

RP2350 Datasheet

A microcontroller
by Raspberry Pi

Colophon

© 2023-2024 Raspberry Pi Ltd

This documentation is licensed under a Creative Commons [Attribution-NoDerivatives 4.0 International \(CC BY-ND\)](#).

build-date: 2024-07-25

build-version: ed3a521-clean

Legal disclaimer notice

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME ("RESOURCES") ARE PROVIDED BY RASPBERRY PI LTD ("RPL") "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). RPL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPL's [Standard Terms](#). RPL's provision of the RESOURCES does not expand or otherwise modify RPL's [Standard Terms](#) including but not limited to the disclaimers and warranties expressed in them.

Table of contents

Colophon	1
Legal disclaimer notice	1
1. Introduction	11
1.1. Why is the chip called RP2350?	12
1.2. The Chip	12
1.3. Pinout Reference	14
1.3.1. Pin Locations	14
1.3.2. Pin Descriptions	15
1.3.3. GPIO Functions (Bank 0)	16
1.3.4. GPIO Functions (Bank 1)	20
2. System Bus	22
2.1. Bus Fabric	22
2.1.1. Bus Priority	23
2.1.2. Bus Security Filtering	23
2.1.3. Atomic Register Access	23
2.1.4. APB Bridge	24
2.1.5. Narrow IO Register Writes	24
2.1.6. Global Exclusive Monitor	25
2.1.7. Bus Performance Counters	27
2.2. Address Map	28
2.2.1. ROM	28
2.2.2. XIP	28
2.2.3. SRAM	28
2.2.4. APB Registers	29
2.2.5. AHB Registers	30
2.2.6. Core-local Peripherals (SIO)	31
2.2.7. Cortex-M33 Private Peripherals	31
3. Processor Subsystem	32
3.1. SIO	32
3.1.1. Secure and Non-secure SIO	33
3.1.2. CPUID	34
3.1.3. GPIO Control	35
3.1.4. Hardware Spinlocks	37
3.1.5. Inter-processor FIFOs (Mailboxes)	37
3.1.6. Doorbells	38
3.1.7. Integer Divider	38
3.1.8. RISC-V Platform Timer	39
3.1.9. TMDS Encoder	39
3.1.10. Interpolator	40
3.1.11. List of Registers	49
3.2. Interrupts	78
3.2.1. Non-maskable Interrupt (NMI)	79
3.3. Event Signals (Arm)	79
3.4. Event Signals (RISC-V)	79
3.5. Debug	80
3.5.1. Connecting to the SW-DP	81
3.5.2. Arm Debug	81
3.5.3. RISC-V Debug	82
3.5.4. Debug Power Domains	82
3.5.5. Software control of SWD pins	83
3.5.6. Self-hosted Debug	83
3.5.7. Trace	84
3.5.8. Rescue Reset	84
3.5.9. Security	84
3.5.10. RP-AP	86

3.6. Cortex-M33 Coprocessors	95
3.6.1. GPIO Coprocessor (GPIOC)	95
3.6.2. Double-precision Coprocessor (DCP)	99
3.6.3. Redundancy Coprocessor (RCP)	107
3.6.4. Floating Point Unit	118
3.7. Cortex-M33 Processor	118
3.7.1. Features	119
3.7.2. Configuration	119
3.7.3. Compliance	123
3.7.4. Programmer's model	126
3.7.5. List of Registers	136
3.8. Hazard3 Processor	223
3.8.1. Instruction Reference	223
3.8.2. Interrupts and Exceptions	254
3.8.3. Debug	257
3.8.4. Custom Extensions	259
3.8.5. Instruction Cycle Counts	268
3.8.6. Configuration	273
3.8.7. Control and Status Registers	275
3.9. Arm/RISC-V Architecture Switching	305
3.9.1. Automatic Switching	306
3.9.2. Mixed Architecture Combinations	306
4. Memory	307
4.1. ROM	307
4.2. SRAM	308
4.2.1. Other On-chip Memory	309
4.3. Boot RAM	309
4.3.1. List of Registers	309
4.4. External Flash and PSRAM (XIP)	312
4.4.1. XIP Cache	313
4.4.2. QSPI Memory Interface (QMI)	316
4.4.3. Streaming DMA Interface	316
4.4.4. Performance Counters	317
4.4.5. List of XIP_CTRL Registers	317
4.4.6. List of XIP_AUX Registers	322
4.5. OTP	324
5. Bootrom	325
5.1. Bootrom Concepts	325
5.1.1. Image Definitions	325
5.1.2. Partition Tables	326
5.1.3. Flash Permissions	326
5.1.4. Blocks And Block Loops	326
5.1.5. Versioning	327
5.1.6. Anti-Rollback	327
5.1.7. Packaged Binaries	328
5.1.8. A/B Partitions	328
5.1.9. UF2 Targeting	328
5.1.10. Hashing and Signing	329
5.1.11. Flash IMAGE_DEF Boot	329
5.1.12. Boot Slots	330
5.1.13. Flash PARTITION_TABLE Boot	330
5.1.14. Flash IMAGE_DEF + PARTITION_TABLE Boot	330
5.1.15. Flash Update / Version Downgrade	331
5.1.16. Try Before You Buy	331
5.1.17. Address Translation	332
5.1.18. Auto Architecture Switching	332
5.2. Example Boot Scenarios	333
5.2.1. Secure Booting	333
5.2.2. Packaged SRAM Booting	333
5.2.3. A/B Booting	334

5.2.4. A/B Booting with Owned Partitions	335
5.2.5. Custom Bootloader	336
5.2.6. OTP Bootloader	338
5.2.7. Rollback Versions with Bootloaders	338
5.2.8. Example Block Loops	338
5.3. Processor-Controlled Boot Sequence	341
5.3.1. POWMAN Boot Vector	342
5.3.2. Watchdog Boot Vector	342
5.3.3. RAM Image Boot	343
5.3.4. OTP Boot	343
5.3.5. Flash Boot	343
5.3.6. BOOTSEL (USB/UART) Boot	344
5.3.7. Boot Configuration (OTP)	345
5.4. Bootrom APIs	345
5.4.1. Locating The API Functions	345
5.4.2. API Function Availability	346
5.4.3. API Function Return Codes	347
5.4.4. API Functions And Exclusive Access	347
5.4.5. SDK Access To The API	348
5.4.6. Alphabetical List Of API Functions / Data	348
5.4.7. Low-Level Flash Access Functions	349
5.4.8. High-level flash access functions	352
5.4.9. Security Related Functions	353
5.4.10. Miscellaneous Functions	354
5.4.11. System Information Functions	356
5.4.12. Boot Related Functions	360
5.4.13. Non-secure-specific Functions	362
5.4.14. RISC-V specific functions	362
5.4.15. Data Values	362
5.5. USB Mass Storage Interface	364
5.5.1. The RP2350 Drive	364
5.5.2. UF2 Format Details	364
5.5.3. UF2 Targeting Rules	366
5.6. USB PICOBLOCK Interface	368
5.6.1. Identifying The Device	368
5.6.2. Identifying The Interface	368
5.6.3. Identifying The Endpoints	369
5.6.4. PICOBLOCK Commands	369
5.6.5. Control Requests	374
5.7. USB White-Labelling	376
5.7.1. USB Device Descriptor	377
5.7.2. USB Device Strings	377
5.7.3. USB Configuration Descriptor	377
5.7.4. MSD Drive	377
5.7.5. UF2 INDEX.HTM File	377
5.7.6. UF2 INFO_UF2.TXT File	378
5.7.7. SCSI Inquiry	378
5.7.8. Volume Label Example	378
5.8. UART Boot	379
5.8.1. Baud Rate and Clock Requirements	379
5.8.2. UART Boot Shell Protocol	379
5.8.3. UART Boot Programming Flow	380
5.8.4. Recovering from Stuck Interface	381
5.8.5. Requirements for UART Boot Binaries	381
5.9. Launching Code On Processor Core 1	381
5.10. Boot Metadata Details	382
5.10.1. Blocks And Block Loops	382
5.10.2. Common Block Items	383
5.10.3. Image Definition	384
5.10.4. Partition Tables	387

5.10.5. Minimum Viable Metadata	390
6. Power	392
6.1. Power Supplies	392
6.2. Core Voltage Regulator	392
6.2.1. Operating Modes	392
6.2.2. Software Control	393
6.2.3. Power Manager Control	393
6.2.4. Status	394
6.2.5. Current Limit	394
6.2.6. Over Temperature Protection	394
6.2.7. Application Circuit	394
6.2.8. External Components and PCB layout requirements	396
6.2.9. List of Registers	400
6.3. Power Control	401
6.3.1. Changes from RP2040	401
6.3.2. Dynamic Power Control	401
6.3.3. Top-level Clock Gates	401
6.3.4. SLEEP State	402
6.3.5. DORMANT State	402
6.3.6. Memory Power Down	402
6.3.7. Programmer's Model	403
6.4. Power Domains	404
6.4.1. Isolation	404
6.5. Power Manager	405
6.5.1. Core Power Domains	405
6.5.2. Power States	405
6.5.3. Power State Transitions	406
6.5.4. Initiating Power State Transitions	407
6.6. Power Management (POWMAN) Registers	409
7. Resets	442
7.1. Overview	442
7.2. Changes from RP2350	442
7.3. Chip Level Resets	443
7.3.1. Chip-Level Reset table	443
7.3.2. Chip-level Reset Destinations	444
7.3.3. Chip-level Reset Sources	444
7.4. System Resets (Power-on State Machine)	445
7.4.1. Reset Sequence	446
7.4.2. Register Control	447
7.4.3. Interaction with Watchdog	447
7.4.4. List of Registers	447
7.5. Subsystem Resets	451
7.5.1. Overview	451
7.5.2. Programmer's Model	451
7.5.3. List of Registers	453
7.6. Power-on Reset & Brownout Detection	456
7.6.1. Power-on Reset (POR)	457
7.6.2. Brownout Detection (BOD)	457
7.6.3. Supply Monitor	460
7.6.4. List of Registers	460
8. Clocks	461
8.1. Overview	461
8.1.1. Clock sources	462
8.1.2. Clock Generators	466
8.1.3. Frequency Counter	469
8.1.4. Resus	470
8.1.5. Programmer's Model	470
8.1.6. List of Registers	476
8.2. Crystal Oscillator (XOSC)	501
8.2.1. Overview	501

8.2.2. Changes from RP2040	502
8.2.3. Usage	503
8.2.4. Startup Delay	503
8.2.5. XOSC Counter	503
8.2.6. DORMANT mode	503
8.2.7. Programmer's Model	504
8.2.8. List of Registers	505
8.3. Ring Oscillator (ROSC)	507
8.3.1. Overview	507
8.3.2. ROSC/XOSC trade-offs	508
8.3.3. Modifying the frequency	508
8.3.4. ROSC divider	509
8.3.5. Random Number Generator	509
8.3.6. ROSC Counter	509
8.3.7. DORMANT mode	510
8.3.8. List of Registers	510
8.4. Low Power Oscillator (LPOSC)	514
8.4.1. Frequency Accuracy and Calibration	515
8.4.2. Using an External Low Power Clock	515
8.4.3. List of Registers	515
8.5. Tick Generators	515
8.5.1. Overview	516
8.5.2. List of Registers	516
8.6. PLL	520
8.6.1. Overview	520
8.6.2. Changes from RP2040	521
8.6.3. Calculating PLL parameters	521
8.6.4. Configuration	524
8.6.5. List of Registers	527
9. GPIO	530
9.1. Overview	530
9.2. Changes from RP2040	531
9.3. Reset State	531
9.4. Function Select	532
9.5. Interrupts	536
9.6. Pads	537
9.7. Pad Isolation Latches	538
9.8. Processor GPIO Controls (SIO)	538
9.9. GPIO Coprocessor Port	539
9.10. Software Examples	539
9.10.1. Select an IO function	539
9.10.2. Enable a GPIO interrupt	545
9.11. List of Registers	546
9.11.1. IO - User Bank	546
9.11.2. IO - QSPI Bank	682
9.11.3. Pad Control - User Bank	704
9.11.4. Pad Control - QSPI Bank	731
10. Security	735
10.1. Overview (Arm)	735
10.1.1. Secure Boot	735
10.1.2. Encrypted Boot	736
10.1.3. Isolating Trusted and Untrusted Software	737
10.2. Processor Security Features (Arm)	738
10.2.1. Background	738
10.2.2. IDAU Address Map	739
10.3. Overview (RISC-V)	740
10.4. Processor Security Features (RISC-V)	740
10.4.1. Secure Boot Enable Procedure	741
10.5. Access Control	741
10.5.1. GPIO Access Control	742

10.5.2. Bus Access Control	743
10.5.3. List of Registers	745
10.6. DMA	792
10.6.1. Channel Security Attributes	792
10.6.2. Memory Protection Unit	792
10.6.3. DREQ Attributes	793
10.6.4. IRQ Attributes	793
10.7. Glitch Detector	793
10.7.1. Theory of Operation	793
10.7.2. Trigger Response	794
10.7.3. List of Registers	794
10.8. Factory Test JTAG	798
10.9. Decommissioning	798
11. PIO	800
12. Peripherals	801
12.1. UART	801
12.1.1. Overview	801
12.1.2. Functional description	802
12.1.3. Operation	804
12.1.4. UART hardware flow control	806
12.1.5. UART DMA Interface	807
12.1.6. Interrupts	809
12.1.7. Programmer's Model	810
12.1.8. List of Registers	812
12.2. I2C	825
12.2.1. Features	825
12.2.2. IP Configuration	826
12.2.3. I2C Overview	826
12.2.4. I2C Terminology	828
12.2.5. I2C Behaviour	829
12.2.6. I2C Protocols	831
12.2.7. Tx FIFO Management and START, STOP and RESTART Generation	834
12.2.8. Multiple Master Arbitration	836
12.2.9. Clock Synchronization	837
12.2.10. Operation Modes	838
12.2.11. Spike Suppression	843
12.2.12. Fast Mode Plus Operation	844
12.2.13. Bus Clear Feature	844
12.2.14. IC_CLK Frequency Configuration	845
12.2.15. DMA Controller Interface	849
12.2.16. Operation of Interrupt Registers	850
12.2.17. List of Registers	850
12.3. SPI	888
12.3.1. Overview	889
12.3.2. Functional Description	890
12.3.3. Operation	892
12.3.4. List of Registers	902
12.4. ADC and Temperature Sensor	909
12.4.1. Changes from RP2040	910
12.4.2. ADC controller	910
12.4.3. SAR ADC	911
12.4.4. ADC ENOB	914
12.4.5. INL and DNL	914
12.4.6. Temperature Sensor	914
12.4.7. List of Registers	915
12.5. PWM	918
12.5.1. Overview	918
12.5.2. Programmer's Model	919
12.5.3. List of Registers	927
12.6. DMA	935

12.6.1. Changes from RP2040	936
12.6.2. Configuring Channels	937
12.6.3. Triggering Channels	939
12.6.4. Data Request (DREQ)	941
12.6.5. Interrupts	943
12.6.6. Security	943
12.6.7. Bus Error Handling	946
12.6.8. Additional Features	948
12.6.9. Example Use Cases	949
12.6.10. List of Registers	953
12.7. USB	984
12.7.1. Overview	984
12.7.2. Changes from RP2040	985
12.7.3. Architecture	987
12.7.4. Programmer's Model	998
12.7.5. List of Registers	1002
12.8. System Timers	1026
12.8.1. Overview	1026
12.8.2. Counter	1026
12.8.3. Alarms	1027
12.8.4. Programmer's Model	1027
12.8.5. List of Registers	1031
12.9. Watchdog	1036
12.9.1. Overview	1036
12.9.2. Changes from RP2040	1036
12.9.3. Watchdog Counter	1037
12.9.4. Control Watchdog Reset Levels	1037
12.9.5. Scratch Registers	1037
12.9.6. Programmer's Model	1037
12.9.7. List of Registers	1039
12.10. Always-On Timer	1041
12.10.1. Overview	1041
12.10.2. Changes from RP2040	1041
12.10.3. Accessing the AON-Timer	1041
12.10.4. Using the Alarm	1042
12.10.5. Selecting the AON-Timer Tick Source	1042
12.10.6. Synchronising the AON-Timer to an External 1Hz Clock	1044
12.10.7. Using an external clock or tick from GPIO	1044
12.10.8. Using a Tick Faster than 1ms	1045
12.10.9. List of Registers	1045
12.11. HSTX	1046
12.11.1. Data FIFO	1046
12.11.2. Output Shift Register	1046
12.11.3. Bit Crossbar	1047
12.11.4. Clock Generator	1049
12.11.5. Command Expander	1050
12.11.6. PIO-to-HSTX Coupled Mode	1051
12.11.7. List of Control Registers	1052
12.11.8. List of FIFO Registers	1059
12.12. TRNG	1060
12.12.1. Overview	1060
12.12.2. Configuration	1060
12.12.3. Operation	1061
12.12.4. Caveats	1061
12.12.5. List of Registers	1061
12.13. SHA-256 Accelerator	1067
12.13.1. Message Padding	1068
12.13.2. Throughput	1068
12.13.3. Data Size and Endianness	1068
12.13.4. DMA DREQ Interface	1068

12.13.5. List of Registers	1069
12.14. QSPI Memory Interface (QMI)	1072
12.14.1. Overview	1072
12.14.2. QSPI Transfers	1074
12.14.3. Timing	1077
12.14.4. Address Translation	1080
12.14.5. Direct Mode	1081
12.14.6. List of Registers	1082
12.15. System Control Registers	1095
12.15.1. SYSINFO	1095
12.15.2. SYSCFG	1097
12.15.3. TBMAN	1100
12.15.4. BUSCTRL	1101
13. OTP	1115
13.1. OTP Address Map	1115
13.1.1. Guarded Reads	1116
13.2. Background: OTP IP Details	1116
13.3. Background: OTP Hardware Architecture	1117
13.3.1. Lock Shim	1117
13.3.2. External Interfaces	1118
13.3.3. OTP Boot Oscillator	1119
13.3.4. Power-up State Machine	1119
13.4. Critical Flags	1120
13.5. Page Locks	1121
13.5.1. Lock Progression	1121
13.5.2. OTP Access Keys	1122
13.5.3. Lock Encoding in OTP	1122
13.5.4. Special Pages	1123
13.5.5. Permissions of Blank Devices	1123
13.6. Error Correction Code (ECC)	1124
13.6.1. Bit repair by polarity (BRP)	1124
13.6.2. Modified Hamming ECC	1125
13.7. Device Decommissioning (RMA)	1125
13.8. List of Registers	1126
13.9. Predefined OTP Data Locations	1138
14. Electrical and Mechanical	1174
Appendix A: Register Field Types	1175
Changes from RP2040	1175
Standard types	1175
RW:	1175
RO:	1175
WO:	1175
Clear types	1175
SC:	1175
WC:	1175
FIFO types	1176
RWF:	1176
RF:	1176
WF:	1176
Appendix B: Units Used in This Document	1177
Memory and Storage Capacity	1177
Physical Quantities	1177
Scale Prefixes	1178
Appendix E: Errata	1180
ACCESSCTRL	1180
RP2350-E3	1180
Bootrom	1180
RP2350-E10	1180
DMA	1181
RP2350-E5	1181

RP2350-E8	1181
GPIO	1182
RP2350-E9	1182
Hazard3	1183
RP2350-E4	1183
RP2350-E6	1183
RP2350-E7	1184
SIO	1184
RP2350-E1	1184
RP2350-E2	1185
Appendix F: Documentation Release History	1186

Chapter 1. Introduction

RP2350 is a new family of microcontrollers from Raspberry Pi that offers significant enhancements over RP2040. Key features include:

- Dual Cortex-M33 or Hazard3 processors at 150MHz
- 520 KiB^a on-chip SRAM, in 10 independent banks
- Extended low-power sleep states with optional SRAM retention: as low as 10 μ A DVDD
- 8 KiB of one-time-programmable storage (OTP)
- Up to 16 MiB of external QSPI flash/PSRAM via dedicated QSPI bus
 - Additional 16 MiB flash/PSRAM accessible via optional second chip-select
- On-chip switched-mode power supply to generate core voltage
 - Low-quiescent-current LDO mode can be enabled for sleep states
- 2x on-chip PLLs for internal or external clock generation
- GPIOs are 5 V-tolerant (powered), and 3.3 V-failsafe (unpowered)
- Security features:
 - Optional boot signing, enforced by on-chip mask ROM, with key fingerprint in OTP
 - Protected OTP storage for optional boot decryption key
 - Global bus filtering based on Arm or RISC-V security/privilege levels
 - Peripherals, GPIOs and DMA channels individually assignable to security domains
 - Hardware mitigations for fault injection attacks
 - Hardware SHA-256 accelerator
- Peripherals:
 - 2x UARTs
 - 2x SPI controllers
 - 2x I2C controllers
 - 24x PWM channels
 - USB 1.1 controller and PHY, with host and device support
 - 12x PIO state machines
 - 1x HSTX peripheral

The RP2350 family of devices is shown in table [Table 1](#), showing options for QFN-80 (10 x 10 mm) and QFN-60 (7 x 7 mm) packages, with and without flash-in-package.

Table 1. RP2350 device family

Product	Package	Internal Flash	GPIO	Analogue inputs
RP2350A	QFN-60	None	30	4
RP2350B	QFN-80	None	48	8
RP2354A	QFN-60	2 MiB ^a	30	4
RP2354B	QFN-80	2 MiB ^a	48	8

^a For more information about the units used in this document, see [Appendix B](#).

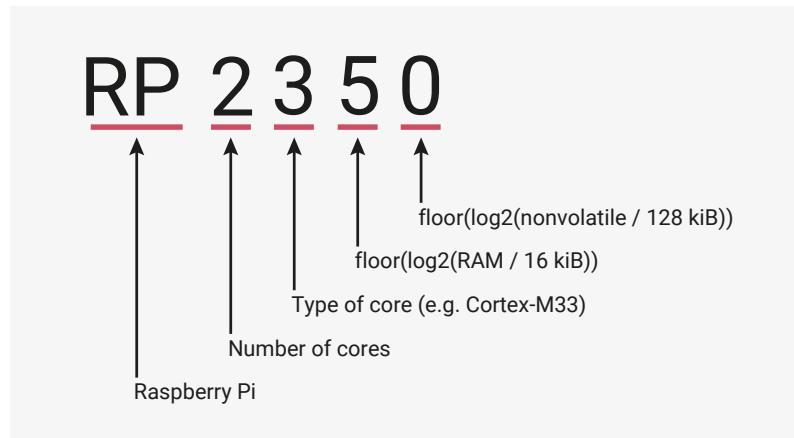
1.1. Why is the chip called RP2350?

The post-fix numeral on RP2350 comes from the following,

1. Number of processor cores (2)
2. Loosely which type of processor (Cortex-M33 or Hazard3)
3. $\log_2 \frac{RAM}{16KiB}$
4. $\log_2 \lfloor \frac{\text{nonvolatile}}{128KiB} \rfloor$ (or 0 if no onboard nonvolatile storage)

see [Figure 1](#).

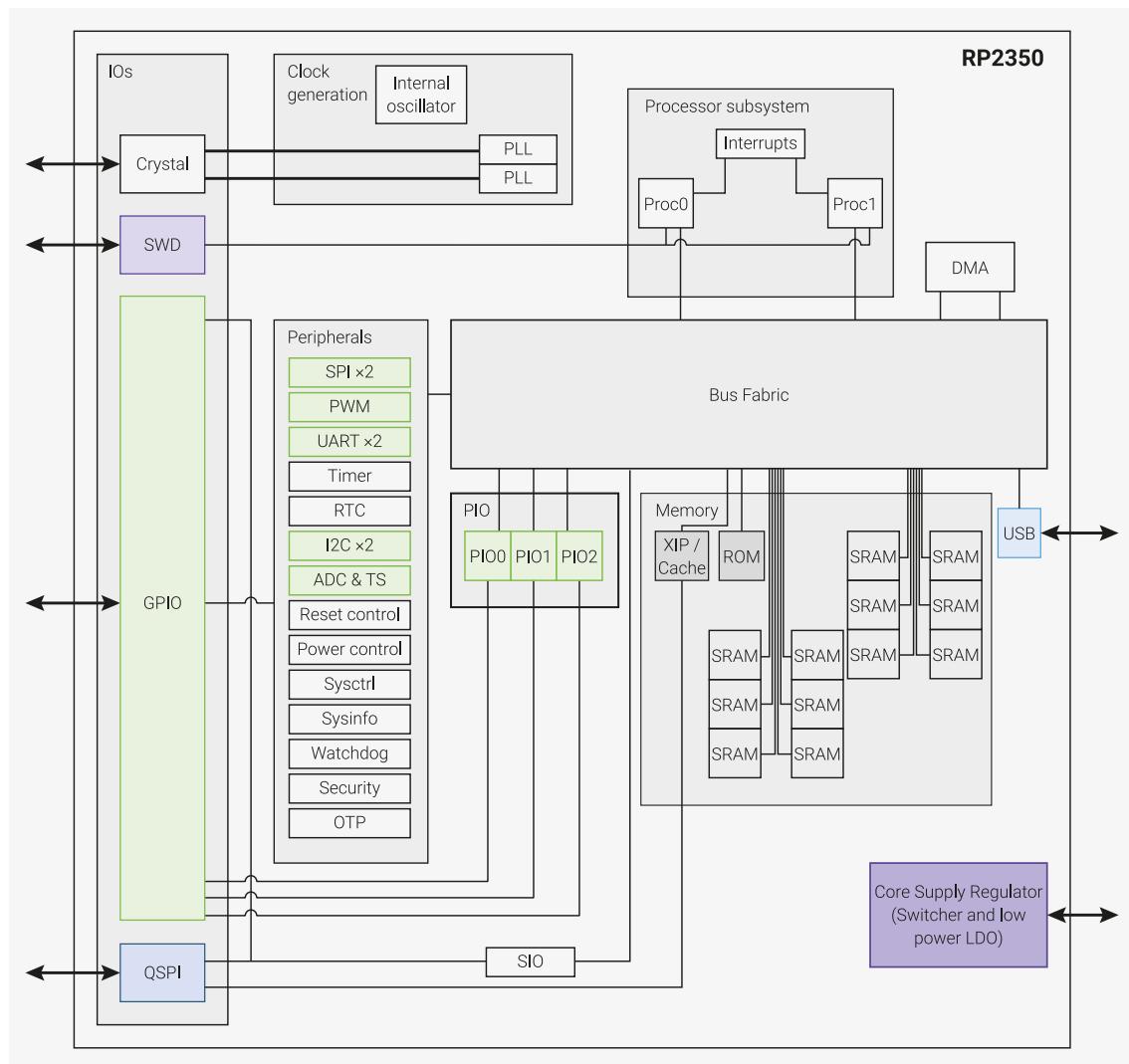
Figure 1. An explanation for the name of the RP2350 chip.



1.2. The Chip

RP2350 contains dual Cortex-M33 or Hazard3 processors, DMA, memory and peripherals, interconnected via AHB/APB bus fabric.

Figure 2. A system overview of the RP2350 chip



Code may be executed directly from external memory through a dedicated QSPI memory interface. A cache improves performance for typical applications. External RAM can also be attached via this interface.

Debug is available via the SWD interface. This allows an external host to load, run, halt and inspect software running on the system.

Internal SRAM can contain code or data. It is addressed as a single 520 kB region, but physically partitioned into 10 banks to allow simultaneous parallel access from different masters.

A system DMA is available to offload repetitive data transfer tasks from the processors.

GPIO pins can be driven directly, or from a variety of dedicated logic functions.

Dedicated hardware peripherals provide fixed IO functions such as SPI, I2C, UART.

Flexible PIO controllers can be used to provide a wider variety of IO functions, or to supplement the number of fixed-function peripherals.

A USB controller with embedded PHY provides FS/LS Host or Device connectivity under software control.

Four or eight ADC inputs (depending on package size) are shared with GPIO pins.

Two PLLs provide a fixed 48MHz clock for USB or ADC, and a flexible system clock up to 150MHz.

An internal voltage regulator supplies the core voltage, so the end product generally only needs supply the IO voltage. The regulator operates as a switched mode buck converter when the system is awake, providing up to 200 mA at a variable output voltage, and can switch to a low-quiescent-current LDO mode when the system is asleep, providing up to 1 mA for state retention.

The system features low power states where unused logic is powered off, supporting wakeup from timer or IO events. The amount of SRAM retained during power-down is configurable.

The internal 8 kB one-time-programmable storage (OTP) contains chip information such as unique identifiers, can be used to configure hardware and bootrom security features, and can be programmed with user-supplied code and data.

The built-in bootrom implements direct boot from flash or OTP, and serial boot from USB or UART. Code signature enforcement is supported for all boot media, using a key fingerprint registered in internal OTP storage.

RISC-V architecture support is implemented by dynamically swapping the Cortex-M33 (Armv8-M) processors with Hazard3 (RV32IMAC+) processors. Both architectures are available on all RP2350 family of devices. The RISC-V cores support debug over SWD, and can be programmed with the same SDK as the Arm cores.

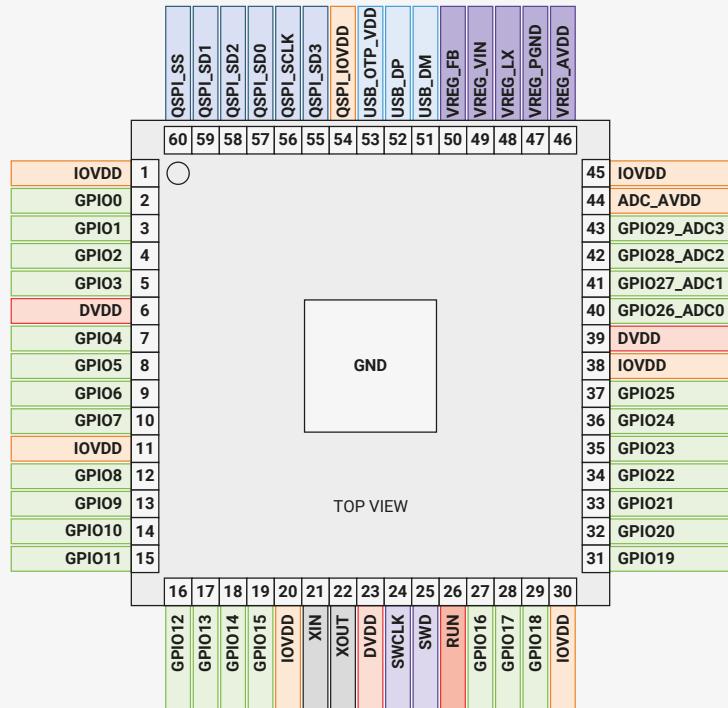
1.3. Pinout Reference

This section provides a quick reference for pinout and pin functions. Full details, including electrical specifications and package drawings, can be found in [Chapter 14](#).

1.3.1. Pin Locations

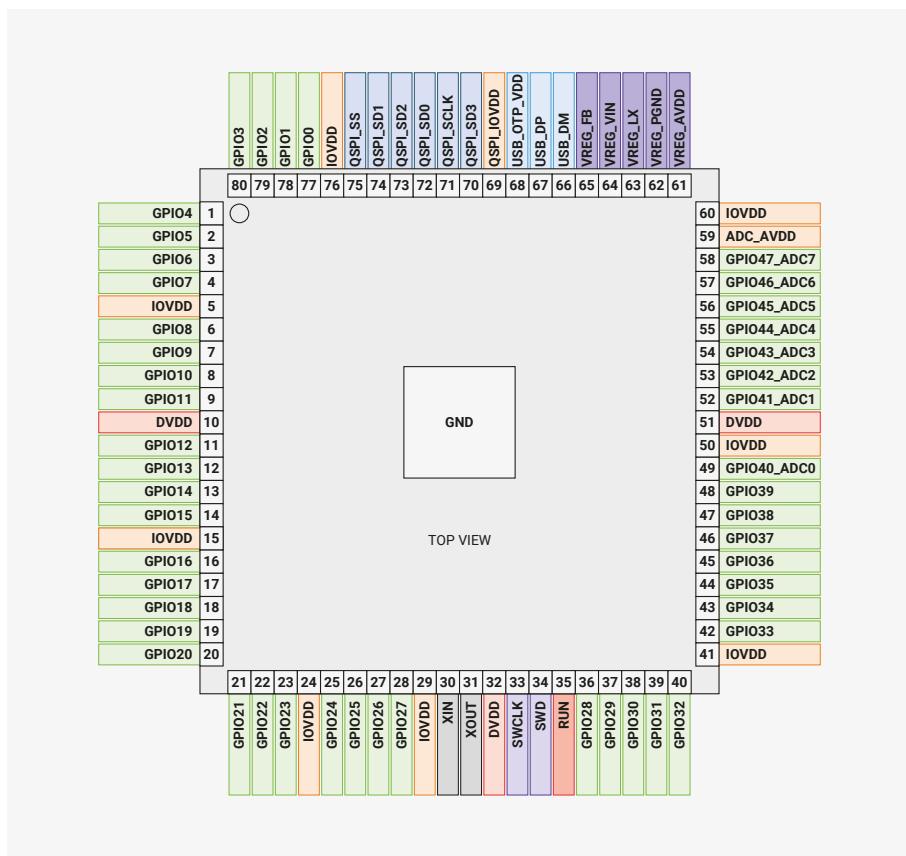
1.3.1.1. QFN-60 (RP2350A)

Figure 3. RP2350
Pinout for QFN-60
7x7mm (reduced ePad
size)



1.3.1.2. QFN-80 (RP2350B)

Figure 4. RP2350
Pinout for QFN-80
10x10mm (reduced
ePad size)



1.3.2. Pin Descriptions

Table 2. The function of each pin is briefly described here. Full electrical specifications can be found in [Chapter 14](#).

Name	Description
GPIOx	General-purpose digital input and output. RP2350 can connect one of a number of internal peripherals to each GPIO, or control GPIOs directly from software.
GPIOx/ADCy	General-purpose digital input and output, with analogue-to-digital converter function. The RP2350 ADC has an analogue multiplexer which can select any one of these pins, and sample the voltage.
QSPIx	Interface to a SPI, Dual-SPI or Quad-SPI flash or PSRAM device, with execute-in-place support. These pins can also be used as software-controlled GPIOs, if they are not required for flash access.
USB_DM and USB_DP	USB controller, supporting Full Speed device and Full/Low Speed host. A 27Ω series termination resistor is required on each pin, but bus pullups and pulldowns are provided internally. These pins can be used as software-controlled GPIOs, if USB is not required.
XIN and XOUT	Connect a crystal to RP2350's crystal oscillator. XIN can also be used as a single-ended CMOS clock input, with XOUT disconnected. The USB bootloader defaults to a 12MHz crystal or 12MHz clock input, but this can be configured via OTP.
RUN	Global asynchronous reset pin. Reset when driven low, run when driven high. If no external reset is required, this pin can be tied directly to IOVDD.
SWCLK and SWD	Access to the internal Serial Wire Debug multi-drop bus. Provides debug access to both processors, and can be used to download code.
GND	Single external ground connection, bonded to a number of internal ground pads on the RP2350 die.

Name	Description
IOVDD	Power supply for digital GPIOs, nominal voltage 1.8V to 3.3V
USB OTP VDD	Power supply for internal USB Full Speed PHY and OTP storage, nominal voltage 3.3V
ADC_AVDD	Power supply for analogue-to-digital converter, nominal voltage 3.3V
QSPI_IOVDD	Power supply for QSPI IOs, nominal voltage 1.8V to 3.3V
VREG_AVDD	Analogue power supply for internal core voltage regulator, nominal voltage 3.3V
VREG_PGND	Power-ground connection for internal core voltage regulator, tie to ground externally
VREG_LX	Switched-mode output for internal core voltage regulator, connected to external inductor. Max current 200 mA, nominal voltage 1.1V after filtering.
VREG_VIN	Power input for internal core voltage regulator, nominal voltage 2.7V to 5.5V
VREG_FB	Voltage feedback for internal core voltage regulator, connect to filtered VREG output (e.g. to DVDD, if the regulator is used to supply DVDD)
DVDD	Digital core power supply, nominal voltage 1.1V. Must be connected externally, either to the voltage regulator output, or an external board-level power supply.

1.3.3. GPIO Functions (Bank 0)

Each individual GPIO pin can be connected to an internal peripheral via the GPIO functions defined below. Some internal peripheral connections appear in multiple places to allow some system level flexibility. SIO, PIO0, PIO1 and PIO2 can connect to all GPIO pins and are controlled by software (or software controlled state machines) so can be used to implement many functions.

Table 3. General Purpose Input/Output (GPIO) Bank 0 Functions

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
0		SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2	XIP_CS1n	USB OVCUR DET	
1		SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2	TRACECLK	USB VBUS DET	
2		SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2	TRACEDATA0	USB VBUS EN	UART0 TX
3		SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	TRACEDATA1	USB OVCUR DET	UART0 RX
4		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	TRACEDATA2	USB VBUS DET	
5		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	TRACEDATA3	USB VBUS EN	
6		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART1 TX
7		SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 RX
8		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	XIP_CS1n	USB VBUS EN	
9		SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	
10		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 TX
11		SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART1 RX
12	HSTX	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB OVCUR DET	
13	HSTX	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB VBUS DET	
14	HSTX	SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO1	USB VBUS EN	UART0 TX
15	HSTX	SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB OVCUR DET	UART0 RX
16	HSTX	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
17	HSTX	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	
18	HSTX	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART0 TX
19	HSTX	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	XIP_CS1n	USB VBUS DET	UART0 RX
20		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB VBUS EN	
21		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB OVCUR DET	
22		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO1	USB VBUS DET	UART1 TX

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
23		SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB VBUS EN	UART1 RX
24		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT2	USB OVCUR DET	
25		SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT3	USB VBUS DET	
26		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART1 TX
27		SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART1 RX
28		SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
29		SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	
GPIOs 30 through 47 are QFN-80 only:												
30		SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART0 TX
31		SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART0 RX
32		SPI0 RX	UART0 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2		USB VBUS EN	
33		SPI0 CSn	UART0 RX	I2C0 SCL	PWM8 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	
34		SPI0 SCK	UART0 CTS	I2C1 SDA	PWM9 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART0 TX
35		SPI0 TX	UART0 RTS	I2C1 SCL	PWM9 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART0 RX
36		SPI0 RX	UART1 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	
37		SPI0 CSn	UART1 RX	I2C0 SCL	PWM10 B	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
38		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM11 A	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART1 TX
39		SPI0 TX	UART1 RTS	I2C1 SCL	PWM11 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART1 RX
40		SPI1 RX	UART1 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
41		SPI1 CSn	UART1 RX	I2C0 SCL	PWM8 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	
42		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM9 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART1 TX
43		SPI1 TX	UART1 RTS	I2C1 SCL	PWM9 B	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 RX
44		SPI1 RX	UART0 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2		USB VBUS EN	

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
45		SPI1 CSn	UART0 RX	I2C0 SCL	PWM10 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	
46		SPI1 SCK	UART0 CTS	I2C1 SDA	PWM11 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART0 TX
47		SPI1 TX	UART0 RTS	I2C1 SCL	PWM11 B	SIO	PIO0	PIO1	PIO2	XIP_CS1n	USB VBUS EN	UART0 RX

Table 4. GPIO bank 0 function descriptions

Function Name	Description
SPIx	Connect one of the internal PL022 SPI peripherals to GPIO
UARTx	Connect one of the internal PL011 UART peripherals to GPIO
I2Cx	Connect one of the internal DW I2C peripherals to GPIO
PWMx A/B	Connect a PWM slice to GPIO. There are twelve PWM slices, each with two output channels (A/B). The B pin can also be used as an input, for frequency and duty cycle measurement.
SIO	Software control of GPIO, from the single-cycle IO (SIO) block. The SIO function (F5) must be selected for the processors to drive a GPIO, but the input is always connected, so software can check the state of GPIOs at any time.
PIOx	Connect one of the programmable IO blocks (PIO) to GPIO. PIO can implement a wide variety of interfaces, and has its own internal pin mapping hardware, allowing flexible placement of digital interfaces on bank 0 GPIOs. The PIO function (F6, F7, F8) must be selected for PIO to drive a GPIO, but the input is always connected, so the PIOs can always see the state of all pins.
HSTX	Connect the high-speed transmit peripheral (HSTX) to GPIO
CLOCK GPINx	General purpose clock inputs. Can be routed to a number of internal clock domains on RP2350, e.g. to provide a 1Hz clock for the AON-Timer, or can be connected to an internal frequency counter.
CLOCK GPOUTx	General purpose clock outputs. Can drive a number of internal clocks (including PLL outputs) onto GPIOs, with optional integer divide.
TRACECLK, TRACEDATAx	CoreSight execution trace output from Cortex-M33 processors (Arm-only)
USB OVCUR DET/VBUS DET/VBUS EN	USB power control signals to/from the internal USB controller
QMI CSxn	Auxiliary chip select for QSPI bus, to allow execute-in-place from an additional flash or PSRAM device

NOTE

GPIOs 0 through 29 are available in all package variants. GPIOs 30 through 47 are available only in QFN-80 (RP2350B) package.

NOTE

Analogue input is available on GPIOs 26 through 29 in the QFN-60 package (RP2350A), for a total of four inputs, and on GPIOs 40 through 47 in the QFN-80 package (RP2350B), for a total of eight inputs.

1.3.4. GPIO Functions (Bank 1)

GPIO functions are also available on the six dedicated QSPI pins, which are usually used for flash execute-in-place, and on the USB DP/DM pins. These may become available for general-purpose use depending on the use case, for example, QSPI pins may not be needed for code execution if RP2350 is booting from internal OTP storage, or being controlled externally via SWD.

Table 5. GPIO Bank 1 Functions

Pin	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
USB DP			UART1 TX	I2C0 SDA		SIO						

Pin	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
USB DM			UART1 RX	I2C0 SCL		SIO						
QSPI SCK	QMI SCK		UART1 CTS	I2C1 SDA		SIO						UART1 TX
QSPI CSn	QMI CS0n		UART1 RTS	I2C1 SCL		SIO						UART1 RX
QSPI SD0	QMI SD0		UART0 TX	I2C0 SDA		SIO						
QSPI SD1	QMI SD1		UART0 RX	I2C0 SCL		SIO						
QSPI SD2	QMI SD2		UART0 CTS	I2C1 SDA		SIO						UART0 TX
QSPI SD3	QMI SD3		UART0 RTS	I2C1 SCL		SIO						UART0 RX

Table 6. GPIO bank 1 function descriptions

Function Name	Description
UARTx	Connect one of the internal PL011 UART peripherals to GPIO
I2Cx	Connect one of the internal DW I2C peripherals to GPIO
SIO	Software control of GPIO, from the single-cycle IO (SIO) block. The SIO function (F5) must be selected for the processors to drive a GPIO, but the input is always connected, so software can check the state of GPIOs at any time.
QMI	QSPI memory interface peripheral, used for execute-in-place from external QSPI flash or PSRAM memory devices.

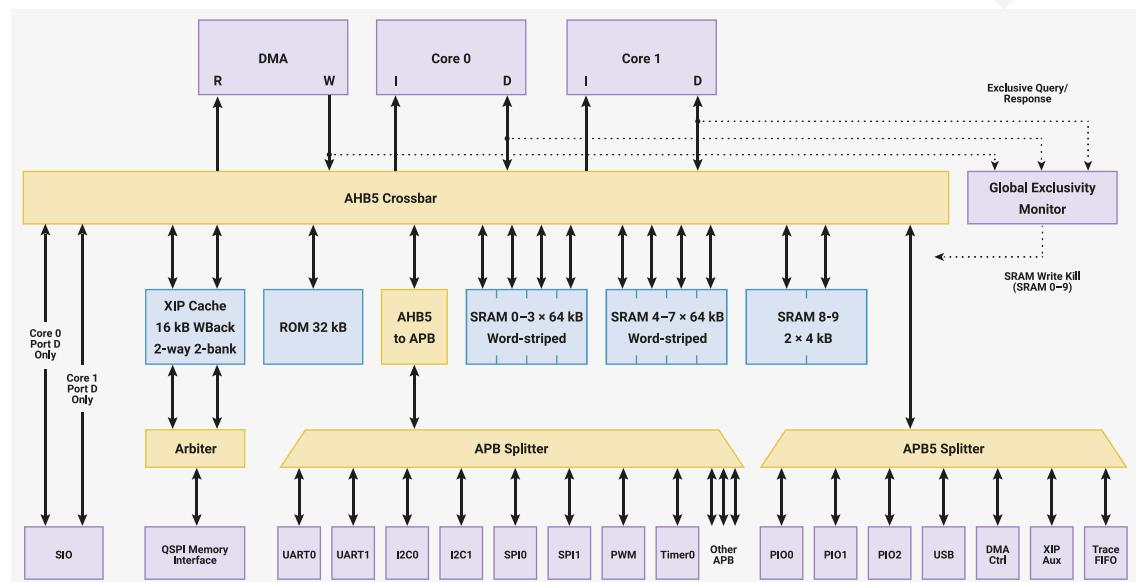
Chapter 2. System Bus

2.1. Bus Fabric

The RP2350 bus fabric routes addresses and data across the chip.

Figure 5 shows the high-level structure of the bus fabric. The main AHB5 crossbar routes addresses and data between its 6 upstream ports and 17 downstream ports, with up to six bus transfers taking place each cycle. All data paths are 32 bits wide. Memories connect to multiple dedicated ports on the main crossbar, for the best possible memory bandwidth. High-bandwidth AHB peripherals share a port on the crossbar. An APB bridge provides access to system control registers and lower-bandwidth peripherals. The SIO peripherals are accessed via a dedicated path from each processor.

Figure 5. RP2350 bus fabric overview.



The bus fabric connects 6 AHB5 managers, i.e. bus ports which generate addresses:

- Core 0: Code port (instruction fetch), and System port (load/store access)
- Core 1: Code port (instruction fetch), and System port (load/store access)
- DMA controller: Read port, Write port

The following 13 downstream ports are symmetrically accessible from all 6 upstream ports:

- Boot ROM (1 port)
- XIP (2 ports, striped)
- SRAM (10 ports, striped)

Additionally, the following 2 ports are accessible for processor load/store and DMA read/write only:

- 1 shared port for fast AHB5 peripherals: PIO0, PIO1, PIO2, USB, DMA control registers, XIP DMA FIFOs, HSTX FIFO, CoreSight trace DMA FIFO
- 1 port for the APB bridge, to all APB peripherals and control registers

NOTE

Instruction fetch from peripherals is *physically disconnected*, to avoid this IDAU-Exempt region ever becoming both Non-secure-writable and Secure-executable. This includes USB RAM, OTP and boot RAM. More details in [Section 10.2.2](#).

The SIO block, which was connected to the Cortex-M0+ IOPORT on RP2040, provides two AHB ports, each dedicated to load/store access from one core.

The six managers can access any six *different* crossbar ports simultaneously. So, at a system clock of 150MHz, the maximum sustained bus bandwidth is 3.6 GB/s.

2.1.1. Bus Priority

The main AHB5 crossbar implements a two-level bus priority scheme. Priority levels are configured separately for core 0, core 1, DMA read and DMA write, using the [BUS_PRIORITY](#) register in the BUSCTRL register block.

When a downstream subordinate receives multiple simultaneous access requests, the port serves high-priority (priority level 1) managers before serving any requests from low-priority (priority 0) managers. If all requests come from managers with the same priority level, the port applies a round-robin tie break, granting access to each manager in turn.

NOTE

Priority arbitration only applies when multiple managers attempt to access the **same** subordinate on the same cycle. When multiple managers access different subordinates, e.g. different SRAM banks, the requests proceed simultaneously.

A subordinate with zero wait states can be accessed once per system clock cycle. When accessing a subordinate with zero wait states (e.g. SRAM), high-priority managers never experience delays caused by accesses from low-priority managers. This guarantees latency and throughput for real-time use cases. However, it also means that low-priority managers may stall until there is a free cycle.

2.1.2. Bus Security Filtering

Every point where the fabric connects to a downstream AHB or APB peripheral is interposed by a bus security filter, which enforces the following access control lists as defined by the ACCESSCTRL registers ([Section 10.5](#)):

- A list of who can access the port: core 0, core 1, DMA, debugger
- A list of the security states from which the port can be accessed: the four combinations of Secure/Non-secure and Privileged/Unprivileged.

Accesses which fail either check are prevented from accessing the downstream port, and return a bus error upstream.

There are two exceptions, which do not implement bus security filters because they implement their own security filtering internally:

- The ACCESSCTRL block itself, which is always world-readable, but filters writes on security and privilege
- Boot RAM, which is hardwired to Secure access only

2.1.3. Atomic Register Access

Each peripheral register block is allocated 4kB of address space, with registers accessed using one of 4 methods, selected by address decode.

- **Addr + 0x0000** : normal read write access

- `Addr + 0x1000` : atomic XOR on write
- `Addr + 0x2000` : atomic bitmask set on write
- `Addr + 0x3000` : atomic bitmask clear on write

This allows software to modify individual fields of a control register without performing a read-modify-write sequence. Instead, the peripheral itself modifies its contents in-place. Without this capability, it is difficult to safely access IO registers when an interrupt service routine is concurrent with code running in the foreground, or when the two processors run code in parallel.

The four atomic access aliases occupy a total of 16kB. Native atomic writes take the same number of clock cycles as normal writes. Most peripherals on RP2350 provide this functionality natively, but some peripherals (I2C, UART, SPI and SSI) add this functionality using a bus interposer. The bus interposer translates upstream atomic writes into downstream read-modify-write sequences at the boundary of the peripheral, at the cost of additional clock cycles. Atomic writes that use a bus interposer take two additional clock cycles compared to normal writes.

The following registers do not support atomic register access:

- SIO ([Section 3.1](#)), though some individual registers (e.g. GPIO) have set, clear, and XOR aliases
- Any register accessed through the self-hosted CoreSight window, including Arm Mem-APs and the RISC-V Debug Module
- Standard Arm control registers on the Cortex-M33 private peripheral bus (PPB), except for Raspberry Pi-specific registers on the EPPB
- OTP programming registers accessed through the SBPI bridge

2.1.4. APB Bridge

The APB bridge provides an interface between the high-speed main AHB5 interconnect and the lower-bandwidth peripherals. Unlike the AHB5 fabric, which offers zero wait state accesses everywhere, APB accesses take a minimum of three cycles for a read, and four cycles for a write.

As a result, the throughput of the APB portion of the bus fabric is lower than the AHB5 portion. However, there is more than sufficient bandwidth to saturate the APB serial peripherals.

The following APB ports contain asynchronous bus crossings, which insert additional stall cycles on top of the typical cost of a read or write in the APB bridge:

- ADC
- HSTX_CTRL
- OTP
- POWMAN

2.1.5. Narrow IO Register Writes

The majority of memory-mapped IO registers on RP2350 ignore the width of bus read/write accesses. They treat all writes as though they were 32 bits in size. This means software cannot use byte or halfword writes to modify part of an IO register: any write to an address where the 30 address MSBs match the register address affects the contents of the entire register.

To update part of an IO register without a read-modify-write sequence, the best solution on RP2350 is atomic set/clear/XOR (see [Section 2.1.3](#)). This is more flexible than byte or halfword writes, as any combination of fields can be updated in one operation.

Upon a 8-bit or 16-bit write (such as a `strb` instruction on the Cortex-M33), the narrow value is replicated multiple times across the 32-bit data bus, so that it is broadcast to all 8-bit or 16-bit segments of the destination register:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/system/narrow_io_write/narrow_io_write.c Lines 19 - 62

```

19 int main() {
20     stdio_init_all();
21
22     // We'll use WATCHDOG_SCRATCH0 as a convenient 32 bit read/write register
23     // that we can assign arbitrary values to
24     io_rw_32 *scratch32 = &watchdog_hw->scratch[0];
25     // Alias the scratch register as two halfwords at offsets +0x0 and +0x2
26     volatile uint16_t *scratch16 = (volatile uint16_t *) scratch32;
27     // Alias the scratch register as four bytes at offsets +0x0, +0x1, +0x2, +0x3:
28     volatile uint8_t *scratch8 = (volatile uint8_t *) scratch32;
29
30     // Show that we can read/write the scratch register as normal:
31     printf("Writing 32 bit value\n");
32     *scratch32 = 0xdeadbeef;
33     printf("Should be 0xdeadbeef: 0x%08x\n", *scratch32);
34
35     // We can do narrow reads just fine -- IO registers treat this as a 32 bit
36     // read, and the processor/DMA will pick out the correct byte lanes based
37     // on transfer size and address LSBs
38     printf("\nReading back 1 byte at a time\n");
39     // Little-endian!
40     printf("Should be ef be ad de: %02x ", scratch8[0]);
41     printf("%02x ", scratch8[1]);
42     printf("%02x ", scratch8[2]);
43     printf("%02x\n", scratch8[3]);
44
45     // The Cortex-M0+ and the RP2040 DMA replicate byte writes across the bus,
46     // and IO registers will sample the entire write bus always.
47     printf("\nWriting 8 bit value 0xa5 at offset 0\n");
48     scratch8[0] = 0xa5;
49     // Read back the whole scratch register in one go
50     printf("Should be 0xa5a5a5a5: 0x%08x\n", *scratch32);
51
52     // The IO register ignores the address LSBs [1:0] as well as the transfer
53     // size, so it doesn't matter what byte offset we use
54     printf("\nWriting 8 bit value at offset 1\n");
55     scratch8[1] = 0x3c;
56     printf("Should be 0x3c3c3c3c: 0x%08x\n", *scratch32);
57
58     // Halfword writes are also replicated across the write data bus
59     printf("\nWriting 16 bit value at offset 0\n");
60     scratch16[0] = 0xf00d;
61     printf("Should be 0xf00df00d: 0x%08x\n", *scratch32);
62 }

```

To disable this behaviour on RP2350, set bit 14 of the address by accessing the peripheral at an offset of `+0x4000`. This causes invalid byte lanes to be driven to zero, rather than being driven with replicated data. In some situations, such as DMA of 8-bit values to the PWM peripheral, the default replication behaviour is not desirable.

2.1.6. Global Exclusive Monitor

The Global Exclusive Monitor enables standard Arm and RISC-V atomic instructions to safely access shared variables in SRAM from both cores. This underpins software libraries for manipulating shared variables, such as `stdatomic.h` in C11. For detailed rules governing the monitor's operation, see the [Arm v8-M Architecture Reference Manual](#).

Arm describes exclusive monitor interactions in terms of a *processing element*, PE, which performs a sequence of bus accesses. For RP2350 purposes, this is one AHB5 manager out of the following three: core 0 load/store, core 1 load/store, and DMA write. The DMA does not itself perform exclusive accesses, but its writes are monitored with

respect to exclusive sequences on either processor. No distinction is made between debugger and non-debugger accesses from a processor.

The monitor observes all transfers on SRAM initiated by the DMA write and processor load/store ports, and pays particular attention to two types of transfer:

- AHB5 *exclusive reads*: Arm `ldrex*` instructions, RISC-V `lr.w` instructions, and the read phase of RISC-V AMOs (The Hazard3 cores on RP2350 implement AMOs as an exclusive read/write pair which retries until the write succeeds)
- AHB5 *exclusive writes*: Arm `strex*` instructions, RISC-V `sc.w` instructions, and the writeback phase of RISC-V AMOs

Based on these observations, the monitor enforces that a matching pair (formed of an exclusive read followed by a successful exclusive write by the same PE) is not interleaved with another PE's successful write (exclusive or not) to the same reservation granule, where a granule is defined as any 16-byte, naturally aligned area of SRAM. An exclusive write succeeds when all of the following are true:

- It is preceded by an exclusive read by the same PE
- No other exclusive writes were performed by this PE since that exclusive read
- The exclusive read was to the same reservation granule
- The exclusive read was of the same size (byte/halfword/word)
- The exclusive read was from the same security and privilege state

If the above conditions are *not* met, the Global Exclusive Monitor shoots down the exclusive write before SRAM can commit the write data. The failure is reported to the originating PE, for example by a non-zero return value from an Arm `strex` instruction.

This implementation of the Armv8-M Global Exclusive Monitor also meets the requirements for RISC-V `lr/sc` and `amo*` instructions, with the caveat that the *RsrvEventual* PMA is not supported. (In practice, whilst it is quite easy to come up with contrived examples of starvation such as the DMA writing to a shared variable on every single cycle, bounded LR/SC and AMO sequences will generally complete quickly.)

⚠ CAUTION

Secure software should avoid shared variables in Non-secure-accessible memory. Such variables are vulnerable to deliberate starvation from exclusive accesses by repeatedly performing non-exclusive writes.

Exclusive accesses are only supported on SRAM. The system treats exclusive accesses to other memory regions as normal reads and writes, reporting exclusivity failure to the originating PE, for example by a non-zero return value from an Arm `strex` instruction.

2.1.6.1. Implementation-defined Monitor Behaviour

The [Armv8-M Architecture Reference Manual](#) leaves several aspects of the Global Exclusive Monitor up to the implementation. For completeness, the RP2350 implementation defines them as follows:

- The reservation granule size is fixed at 16 bytes
- A single reservation is tracked per PE
- The Arm `clrex` instruction does not affect global monitor state
- Any exclusive write by a PE clears that PE's global reservation
- A non-exclusive write by a PE does *not* clear that PE's global reservation, no matter the address

Only the following updates a PE's reservation tag, setting its reservation state to **Exclusive**:

- An exclusive read on SRAM

Only the following changes a PE's reservation state from **Exclusive** to **Open**:

- A successful exclusive write from another PE to this PE's reservation
- A non-exclusive write from another PE to this PE's reservation
- Any exclusive write by this PE
- An exclusive read by this PE, *not* on SRAM

A reservation granule can span multiple SRAM banks, so multiple operations on the same reservation granule may complete on the same cycle. This can result in the following problematic situations:

- Multiple exclusive writes to the same reservation granule, reserved on each PE: in this case the lowest-numbered PE succeeds (in the order DMA < core 0 < core 1), and all others fail.
- A mixture of non-exclusive and exclusive writes to the same reservation granule on the same cycle: in this case, the exclusive writes fail.
- One PE x can write to a reservation granule on the same cycle that another PE y attempts to reserve the same reservation granule via exclusive load: in this case, y's reservation is granted (i.e. the write takes place logically before the load).
- One PE x can write to a reservation granule reserved by another PE y, on the same cycle that PE y makes a new reservation on a *different* reservation granule: in this case, again, y's reservation is granted.

These rules can be summarised by a *logical* ordering of all possible events on a reservation granule that can occur on the same cycle: first all normal writes in arbitrary order, then all exclusive writes in ascending PE order (DMA, core 0, core1), then all loads in arbitrary order.

2.1.6.2. Regions Without Exclusives Support

The Global Exclusive monitor only supports exclusive transactions on certain address ranges. The main system SRAM supports exclusive transactions throughout its entire range: `0x20000000` through `0x20082000`. Within ranges that support exclusive transactions, the Global Exclusive monitor:

- tracks exclusive sequences across all participating PEs
- drives the exclusive success/failure response correctly based on the observed ordering
- shoots down failing exclusive writes so that they have no effect

Exclusive transactions are not supported outside of this range: all exclusive accesses report exclusive failure (both exclusive reads and exclusive writes), and exclusive writes will not be suppressed.

Outside of regions with exclusive transaction support, load/store exclusive loops run forever while still affecting SRAM contents. This applies to both Arm processors performing exclusive reads/writes and RISC-V processors performing `lr.w/sc.w` instructions. However, an `amo*.w` instruction on Hazard3 will result in a Store/AMO Fault, as the hardware detects the failed exclusive read and bails out to avoid an infinite loop.

It is recommended not to perform exclusive accesses on regions outside of main SRAM. Shared variables outside of main SRAM can be protected using either lock variables in main SRAM, the SIO spinlocks, or a locking protocol that does not require exclusive accesses, such as a lock-free queue.

2.1.7. Bus Performance Counters

Bus performance counters automatically count accesses to the main AHB5 crossbar arbiters. These counters can help diagnose high-traffic performance issues.

There are four performance counters, starting at `PERFCTR0`. Each is a 24-bit saturating counter. Counter values can be read from `BUSCTRL_PERFCTRx` and cleared by writing any value to `BUSCTRL_PERFCTRx`. Each counter can count one of the 20 available events at a time, as selected by `BUSCTRL_PERFSELx`. For more information, see [Section 12.15.4](#).

2.2. Address Map

The address map for the device is split into sections as shown in [Table 7](#). Details are shown in the following sections. Unmapped address ranges raise a bus error when accessed.

Each link in the left-hand column of [Table 7](#) goes to a detailed address map for that address range. The detailed address maps have a link for each address to the relevant documentation for that address.

Rough address decode is first performed on bits 31:28 of the address:

Table 7. Address Map Summary

Bus Segment	Base Address
ROM	0x00000000
XIP	0x10000000
SRAM	0x20000000
APB Peripherals	0x40000000
AHB Peripherals	0x50000000
Core-local Peripherals (SIO)	0xd0000000
Cortex-M33 private registers	0xe0000000

2.2.1. ROM

ROM is accessible to DMA, processor load/store, and processor instruction fetch.

Table 8. Address zero points to the bootrom, which is the starting point for both processors when the device is reset.

2.2.2. XIP

XIP is accessible to DMA, processor load/store, and processor instruction fetch.

Table 9. This address range contains various mirrors of a 64 MB space which is mapped to external memory devices. On RP2350 the lower 32 MB is occupied by the QSPI Memory Interface (QMI), and the remainder is reserved. QMI controls are in the APB register section.

NOTE

XIP_SRAM_BASE no longer exists as a separate address range. Cache-as-SRAM is now achieved by pinning cache lines within the cached XIP address space.

2.2.3. SRAM

SRAM is accessible to DMA, processor load/store, and processor instruction fetch.

SRAM0-3 and SRAM4-7 are always striped on bits 3:2 of the address:

Table 10. There are two striped regions, each 256 kB in size, and each striped over 4 SRAM banks. SRAM0-3 are in the SRAM0 power domain, and SRAM4-7 are in the SRAM1 power domain.

Bus Endpoint	Base Address
SRAM_BASE	0x20000000
SRAM_STRIPE_BASE	0x20000000
SRAM0_BASE	0x20000000
SRAM4_BASE	0x20040000
SRAM_STRIPE_END	0x20080000

SRAM 8-9 are always non-striped:

Table 11. These smaller blocks of SRAM are useful for hoisting high-bandwidth data structures like the processor stacks. They are in the SRAM1 power domain.

Bus Endpoint	Base Address
SRAM8_BASE	0x20080000
SRAM9_BASE	0x20081000
SRAM_END	0x20082000

2.2.4. APB Registers

APB peripheral registers are accessible to processor load/store and DMA only. Instruction fetch will always fail.

Table 12. The APB peripheral segment provides access to control and configuration registers, as well as data access for lower-bandwidth peripherals. APB writes cost a minimum of four cycles, and APB reads a minimum of three.

Bus Endpoint	Base Address
SYSINFO_BASE	0x40000000
SYSCFG_BASE	0x40008000
CLOCKS_BASE	0x40010000
PSM_BASE	0x40018000
RESETS_BASE	0x40020000
IO_BANK0_BASE	0x40028000
IO_QSPI_BASE	0x40030000
PADS_BANK0_BASE	0x40038000
PADS_QSPI_BASE	0x40040000
XOSC_BASE	0x40048000
PLL_SYS_BASE	0x40050000
PLL_USB_BASE	0x40058000
ACCESSCTRL_BASE	0x40060000
BUSCTRL_BASE	0x40068000
UART0_BASE	0x40070000
UART1_BASE	0x40078000
SPI0_BASE	0x40080000
SPI1_BASE	0x40088000
I2C0_BASE	0x40090000
I2C1_BASE	0x40098000
ADC_BASE	0x400a0000

Bus Endpoint	Base Address
PWM_BASE	0x400a8000
TIMER0_BASE	0x400b0000
TIMER1_BASE	0x400b8000
HSTX_CTRL_BASE	0x400c0000
XIP_CTRL_BASE	0x400c8000
XIP_QMI_BASE	0x400d0000
WATCHDOG_BASE	0x400d8000
BOOTRAM_BASE	0x400e0000
ROSC_BASE	0x400e8000
TRNG_BASE	0x400f0000
SHA256_BASE	0x400f8000
POWMAN_BASE	0x40100000
TICKS_BASE	0x40108000
OTP_BASE	0x40120000
OTP_DATA_BASE	0x40130000
OTP_DATA_RAW_BASE	0x40134000
OTP_DATA_GUARDED_BASE	0x40138000
OTP_DATA_RAW_GUARDED_BASE	0x4013c000
CORESIGHT_PERIPH_BASE	0x40140000
CORESIGHT_ROMTABLE_BASE	0x40140000
CORESIGHT_AHB_AP_CORE0_BASE	0x40142000
CORESIGHT_AHB_AP_CORE1_BASE	0x40144000
CORESIGHT_TIMESTAMP_GEN_BASE	0x40146000
CORESIGHT_ATB_FUNNEL_BASE	0x40147000
CORESIGHT_TPIU_BASE	0x40148000
CORESIGHT_CTL_BASE	0x40149000
CORESIGHT_APB_AP_RISCV_BASE	0x4014a000
GLITCH_DETECTOR_BASE	0x40158000
TBMAN_BASE	0x40160000

2.2.5. AHB Registers

AHB peripheral registers are accessible to processor load/store and DMA only. Instruction fetch will always fail.

Table 13. The AHB peripheral segment provides access to higher-bandwidth peripherals. The minimum read/write cost is one cycle, and

Bus Endpoint	Base Address
DMA_BASE	0x50000000
USBCTRL_BASE	0x50100000

Bus Endpoint	Base Address
USBCTRL_DPRAM_BASE	0x50100000
USBCTRL_REGS_BASE	0x50110000
PIO0_BASE	0x50200000
PIO1_BASE	0x50300000
PIO2_BASE	0x50400000
XIP_AUX_BASE	0x50500000
HSTX_FIFO_BASE	0x50600000
CORESIGHT_TRACE_BASE	0x50700000

2.2.6. Core-local Peripherals (SIO)

SIO is accessible to processor load/store only.

Table 14. The SIO contains registers which need single-cycle access from both cores concurrently, such as the GPIO registers. Access is always zero-wait-state.

Bus Endpoint	Base Address
SIO_BASE	0xd0000000
SIO_NONSEC_BASE	0xd0020000

2.2.7. Cortex-M33 Private Peripherals

The PPB is accessible to processor load/store only.

These addresses are only accessible to Arm processors: RISC-V processors will return a bus fault.

Table 15. The PPB region contains standard control registers defined by Arm, Non-secure aliases of some of those registers, and a handful of other core-local registers defined by Raspberry Pi (the EPPB)

Bus Endpoint	Base Address
PPB_BASE	0xe0000000
PPB_NONSEC_BASE	0xe0020000
EPPB_BASE	0xe0080000

peripherals may insert up to one wait state.

Chapter 3. Processor Subsystem

The RP2350 processor subsystem consists of two Arm Cortex-M33 processors – each with its standard internal Arm CPU peripherals – alongside external peripherals for GPIO access and inter-core communication. Details of the Arm Cortex-M33 processors, including the specific feature configuration used on RP2350, can be found in [Section 3.7](#).

NOTE

The terms *core0* and *core1*, *proc0* and *proc1* are used interchangeably in RP2350's registers and documentation to refer to processor 0, and processor 1 respectively.

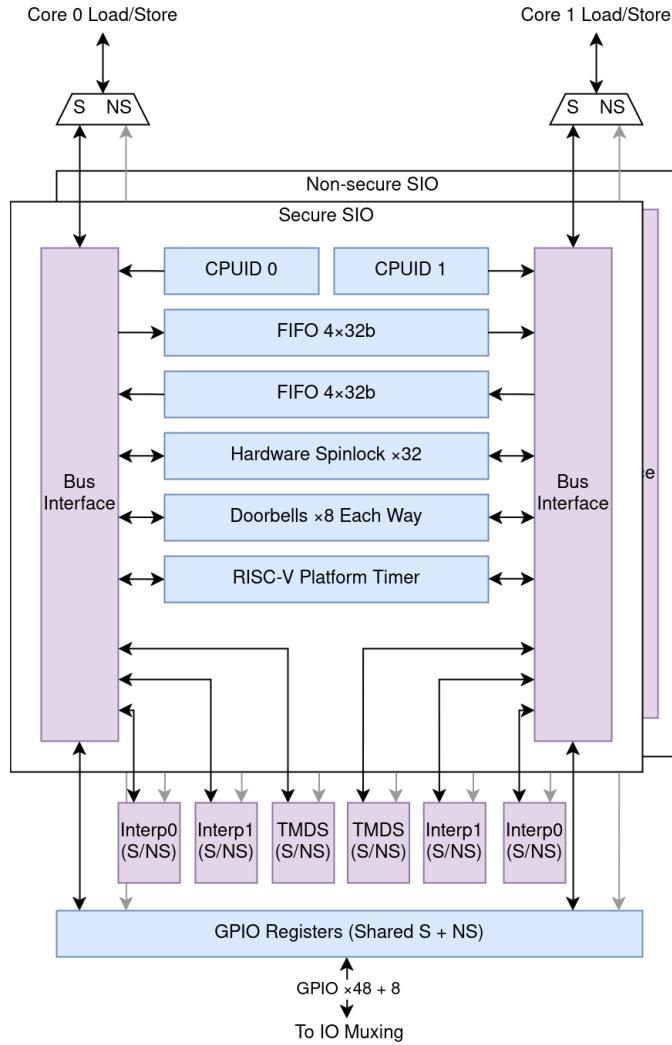
The processors use a number of interfaces to communicate with the rest of the system:

- Each processor uses its own independent code and data 32-bit AHB5 buses to access memory and memory-mapped peripherals (for more information, see [Section 2.1](#)).
- System-level interrupts route to both processors.
- A Serial Wire Debug bus provides debug access to both processors from an external debug host.

3.1. SIO

The Single-cycle IO subsystem (SIO) contains peripherals that require low-latency, deterministic access from the processors. It is accessed via the AHB Fabric. The SIO has a dedicated bus interface for each processor, as shown in [Figure 6](#).

Figure 6. The single-cycle IO block contains registers which processors must access quickly. FIFOs, doorbells and spinlocks support message passing and synchronisation between the two cores. The shared GPIO registers provide fast, direct access to GPIO-capable pins. Interpolators can accelerate common software tasks. Most SIO hardware is banked (duplicated) for Secure and Non-secure access. Grey arrows show bus connections for Non-secure access.



The SIO contains:

- CPUID registers which read as 0/1 on core 0/1 ([Section 3.1.2](#))
- Mailbox FIFOs for passing ordered messages between cores ([Section 3.1.5](#))
- Doorbells for interrupting the opposite core on cumulative and unordered events ([Section 3.1.6](#))
- Hardware spinlocks for implementing critical sections without using exclusive bus accesses ([Section 3.1.4](#))
- Interpolators ([Section 3.1.10](#)) and TMDS encoders ([Section 3.1.9](#))
- Standard RISC-V 64-bit platform timer ([Section 3.1.8](#)) which is usable by both Arm and RISC-V software
- GPIO registers for fast software bitbanging ([Section 3.1.3](#)), with shared access from both cores

Most SIO hardware is duplicated for Secure/Non-secure access. Non-secure access to the FIFO registers will see a physically different FIFO than Secure access to the same address, so that messages belonging to Secure and Non-secure software are not mixed: [Section 3.1.1](#) describes this Secure/Non-secure banking in more detail.

3.1.1. Secure and Non-secure SIO

To allow isolation of Secure and Non-secure software, whilst keeping a consistent programming model for software written to run in either domain, the SIO is duplicated into a Secure and a Non-secure bank. Most hardware is duplicated between the two banks, including:

- Mailbox FIFOs
- Doorbell registers
- Interrupt outputs to processors
- Spinlocks

For example, Non-secure code on core 0 can pass messages to Non-secure code on core 1 through the Non-secure instance of the mailbox FIFO. In turn, this message will generate a Non-secure interrupt, which is separate from the Secure FIFO interrupt line. This does not interfere with any Secure message passing which may be going on at the same time, and Non-secure code can not snoop Secure messages because it does not have access to the Secure mailboxes. The software running in the Secure and Non-secure domain can be identical, and the processors' bus accesses to the SIO will automatically be routed to the Secure or Non-secure version of the mailbox registers.

The following hardware is *not* duplicated:

- The GPIO registers are shared, and Non-secure accesses are filtered on a per-GPIO basis by the Non-secure GPIO mask defined in the ACCESSCTRL `GPIO_NSMASK0` and `GPIO_NSMASK1` registers
- The RISC-V standard platform timer (`MTIME`, `MTIMEH`), which is also usable by Arm processors, is present only in the Secure SIO, as it is a Machine-mode peripheral on RISC-V
- The interpolator and TMDS encoder peripherals are assignable to either the Secure or Non-secure SIO using the `PERI_NONSEC` register

Accesses to the SIO register address range, starting at `0xd0000000` (`SIO_BASE`), are mapped to the SIO bank which matches the security attribute of the bus access. This means accesses from the Arm Secure state, or RISC-V Machine mode, will access the Secure SIO bank, and accesses from the Arm Non-secure state, or RISC-V User mode, will access the Non-secure SIO bank.

Additionally, Secure accesses can use the mirrored address range starting at `0xd0020000` (`SIO_NONSEC_BASE`) to access the Non-secure view of SIO, for example, using the Non-secure doorbells to interrupt Non-secure code running on the other core. Attempting to access this address range from Non-secure code will generate a bus fault.

ⓘ NOTE

The `0x2000` offset of the Secure-to-Non-secure mirror matches the PPB mirrors at `0xe0000000` (`PPB_BASE`) and `0xe0020000` (`PPB_NONSEC_BASE`), which function similarly.

ⓘ NOTE

Debug access is mapped to the Secure/Non-secure SIO using the security attribute of the debugger's bus access, which may differ from the security state that the core was halted in.

3.1.2. CPUID

The `CPUID` SIO register returns a value of 0 when read by core 0, and 1 when read by core 1. This helps software identify the core running the current application. The initial boot sequence also relies on this check: both cores start running simultaneously, core 1 goes into a deep sleep state, and core 0 continues the main boot sequence.

❗ IMPORTANT

Don't confuse the SIO `CPUID` register with the Cortex-M33 `CPUID` register on each processor's internal Private Peripheral Bus, which lists the processor's part number and version.

Reading the `mhartid` CSR on each core returns the same values as `CPUID`: 0 on core 0, and 1 on core 1.

3.1.3. GPIO Control

The SIO GPIO registers control GPIOs which have the SIO function selected (function 5). This function is supported on the following pins:

- all user GPIOs (GPIOs 0 through 29, or 0 through 47, depending on package option)
- QSPI pins
- USB DP/DM pins

All SIO GPIO control registers come in pairs. The lower-addressed register in each pair (e.g. [GPIO_IN](#)) is connected to GPIOs 0 through 31, and the higher-addressed register in each pair (e.g. [GPIO_HI_IN](#)) is connected to GPIOs 32 through 47, the QSPI pins, and the USB DP/DM pins.

 NOTE

To drive a pin with the SIO's GPIO registers, the GPIO multiplexer for this pin must first be configured to select the SIO GPIO function. See [Table 646](#).

These GPIO registers are *shared* between the two cores: both cores can access them simultaneously. There are three groups of registers:

- Output registers, [GPIO_OUT](#) and [GPIO_HI_OUT](#) set the output level of the GPIO. 0 for low output, 1 for high output.
- Output enable registers, [GPIO_OE](#) and [GPIO_HI_OE](#), are used to enable the output driver. 0 for high-impedance, 1 for drive high or low based on [GPIO_OUT](#) and [GPIO_HI_OUT](#).
- Input registers, [GPIO_IN](#) and [GPIO_HI_IN](#), allow the processor to sample the current state of the GPIOs.

Reading [GPIO_IN](#) returns up to 32 input values in a single read, and software then masks out individual pins it is interested in.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h Lines 857 - 867

```

857 static inline bool gpio_get(uint gpio) {
858 #if NUM_BANK0_GPIOS <= 32
859     return !!(1ul << gpio) & sio_hw->gpio_in);
860 #else
861     if (gpio < 32) {
862         return !!(1ul << gpio) & sio_hw->gpio_in);
863     } else {
864         return !!(1ul << (gpio - 32)) & sio_hw->gpio_hi_in);
865     }
866 #endif
867 }
```

The [OUT](#) and [OE](#) registers also have atomic [SET](#), [CLR](#), and [XOR](#) aliases. This allows software to update a subset of the pins in one operation. This ensures safety for parallel GPIO access, both between the two cores and between a single core's interrupt handler and foreground code.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h Lines 906 - 912

```

906 static inline void gpio_set_mask(uint32_t mask) {
907 #if PICO_USE_GPIO_COPROCESSOR
908     gpioc_lo_out_set(mask);
909 #else
910     sio_hw->gpio_set = mask;
911 #endif
912 }
```

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h Lines 953 - 959

```

953 static inline void gpio_clr_mask(uint32_t mask) {
954 #if PICO_USE_GPIO_COPROCESSOR
955     gpioc_lo_out_clr(mask);
956 #else
957     sio_hw->gpio_clr = mask;
958 #endif
959 }

```

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_gpio/include/hardware/gpio.h Lines 1143 - 1168

```

1143 static inline void gpio_put(uint gpio, bool value) {
1144 #if PICO_USE_GPIO_COPROCESSOR
1145     gpioc_bit_out_put(gpio, value);
1146 #elif NUM_BANK0_GPIOS <= 32
1147     uint32_t mask = 1ul << gpio;
1148     if (value)
1149         gpio_set_mask(mask);
1150     else
1151         gpio_clr_mask(mask);
1152 #else
1153     uint32_t mask = 1ul << (gpio & 0x1fu);
1154     if (gpio < 32) {
1155         if (value) {
1156             sio_hw->gpio_set = mask;
1157         } else {
1158             sio_hw->gpio_clr = mask;
1159         }
1160     } else {
1161         if (value) {
1162             sio_hw->gpio_hi_set = mask;
1163         } else {
1164             sio_hw->gpio_hi_clr = mask;
1165         }
1166     }
1167 #endif
1168 }

```

If both processors write to an **OUT** or **OE** register (or any of its **SET/CLR/XOR** aliases) on the same clock cycle, the result is as though core 0 wrote first, then core 1 wrote immediately afterward. For example, if core 0 SETs a bit and core 1 XORs it on the same clock cycle, the bit ends up with a value of **0**.

NOTE

This is a conceptual model for the result produced when two cores write to a GPIO register simultaneously. The register never contains the intermediate values at any point. In the previous example, if the pin is initially **0**, and core 0 performs a **SET** while core 1 performs a **XOR**, the GPIO output remains low throughout the clock cycle.

As well as being shared between cores, the GPIO registers are also shared between security domains. The Secure and Non-secure SIO offer alternative views of the same GPIO registers, which are always mapped as GPIO function **5**. However, the Non-secure SIO can only access pins which are enabled in the GPIO Non-secure mask configured by the ACCESSCTRL registers **GPIO_NSMASK0** and **GPIO_NSMASK1**. The layout of the NSMASK registers matches the layout of the SIO registers – for example, **QSPI_SCK** is bit **26** in both **GPIO_HI_IN** and **GPIO_NSMASK1**.

When a pin is not enabled in Non-secure code:

- writes to the corresponding GPIO registers from a Non-secure context have no effect

- reads from a Non-secure context return zeroes
- reads and writes from a Secure context function as usual using the Secure bank

The GPIO coprocessor port (Section 3.6.1) provides dedicated instructions for accessing the SIO GPIO registers from the Cortex-M33 processors. This includes the ability to read and write 64 bits in a single operation.

3.1.4. Hardware Spinlocks

The SIO provides 32 hardware spinlocks, which can be used to manage mutually-exclusive access to shared software resources. Each spinlock is a one-bit flag, mapped to a different address (from [SPINLOCK0](#) to [SPINLOCK31](#)). Software interacts with each spinlock with one of the following operations:

- Read: Attempt to claim the lock. Read value is non-zero if the lock was successfully claimed, or zero if the lock had already been claimed by a previous read.
- Write (any value): Release the lock. The next attempt to claim the lock will succeed.

If both cores try to claim the same lock on the same clock cycle, core 0 succeeds.

Generally software will acquire a lock by repeatedly polling the lock bit ("spinning" on the lock) until it is successfully claimed. This is inefficient if the lock is held for long periods, so generally the spinlocks should be used to protect short critical sections of higher-level primitives such as mutexes, semaphores and queues.

For debugging purposes, the current state of all 32 spinlocks can be observed via [SPINLOCK_ST](#).

ⓘ NOTE

RP2350 has separate spinlocks for Secure and Non-secure SIO banks because sharing these registers would allow Non-secure code to deliberately starve Secure code that attempts to acquire a lock. See Section 3.1.1.

ⓘ NOTE

The processors on RP2350 support standard atomic/exclusive access instructions which, in concert with the global exclusive monitor (Section 2.1.6), allow both cores to safely share variables in SRAM. The SIO spinlocks are still included for compatibility with RP2040.

3.1.5. Inter-processor FIFOs (Mailboxes)

The SIO contains two FIFOs for passing data, messages or ordered events between the two cores. Each FIFO is 32 bits wide and four entries deep. One of the FIFOs can only be written by core 0 and read by core 1. The other can only be written by core 1 and read by core 0.

Each core writes to its outgoing FIFO by writing to [FIFO_WR](#) and reads from its incoming FIFO by reading from [FIFO_RD](#). A status register, [FIFO_ST](#), provides the following status signals:

- Incoming FIFO contains data ([VLD](#)).
- Outgoing FIFO has room for more data ([RDY](#)).
- The incoming FIFO was read from while empty at some point in the past ([ROE](#)).
- The outgoing FIFO was written to while full at some point in the past ([WOF](#)).

Writing to the outgoing FIFO while full, or reading from the incoming FIFO while empty, does not affect the FIFO state. The current contents and level of the FIFO is preserved. However, this does represent some loss of data or reception of invalid data by the software accessing the FIFO, so a sticky error flag is raised ([ROE](#) or [WOF](#)).

The SIO has a FIFO IRQ output for each core to notify the core that it has received FIFO data. This is a *core-local interrupt*, mapped to the same IRQ number on each core ([SIO_IRQ_FIFO](#), interrupt number [25](#)). Non-secure FIFO interrupts use a separate interrupt line, ([SIO_IRQ_FIFO_NS](#), interrupt number [27](#)). It is not possible to interrupt on the opposite core's

FIFO.

Each IRQ output is the logical OR of the `VLD`, `ROE` and `WOF` bits in that core's `FIFO_ST` register: that is, the IRQ is asserted if any of these three bits is high, and clears again when they are all low. To clear the `ROE` and `WOF` flags, write any value to `FIFO_ST`. To clear the `VLD` flag, read data from the FIFO until it is empty.

If the corresponding interrupt line is enabled in the processor's interrupt controller, the processor takes an interrupt each time data appears in its FIFO, or if it has performed some invalid FIFO operation (read on empty, write on full).

ⓘ NOTE

`ROE` and `WOF` only become set if software misbehaves in some way. Generally, the interrupt handler triggers when data appears in the FIFO, raising the `VLD` flag. Then, the interrupt handler clears the IRQ by reading data from the FIFO until `VLD` goes low once more.

The inter-processor FIFOs and the Event signals are used by the bootrom ([Chapter 5](#)) `wait_for_vector` routine, where core 1 remains in a sleep state until it is woken, and provided with its initial stack pointer, entry point and vector table through the FIFO.

ⓘ NOTE

RP2350 has separate FIFOs and interrupts for Secure and Non-secure SIO banks. See [Section 3.1.1](#)

3.1.6. Doorbells

The doorbell registers raise an interrupt on the opposite core. There are 8 doorbell flags in each direction, combined into a single doorbell interrupt per core. This is a core-local interrupt: the same interrupt number on each core (`SIO_IRQ_BELL`, interrupt number [26](#)) notifies that core of incoming doorbell interrupts.

Whereas the mailbox FIFOs are used for cross-core events whose count and order is important, doorbells are used for events which are accumulative (i.e. may post multiple times, but only answered once) and which can be responded to in any order.

Writing a non-zero value to the `DOORBELL_OUT_SET` register raises the opposite core's doorbell interrupt. The interrupt remains raised until all bits are cleared. Generally, the opposite core enters its doorbell interrupt handler, reads its `DOORBELL_IN_CLR` register to get the mask of active doorbell flags, and then writes back to acknowledge and clear the interrupt.

The `DOORBELL_IN_SET` register allows a processor to ring its own doorbell. This is useful when the routine which rings a doorbell can be scheduled on either core. Likewise, for symmetry, a processor can clear the opposite core's doorbell flags using the `DOORBELL_OUT_CLR` register: this is useful for setup code, but should be avoided in general because of the potential for race conditions when acknowledging interrupts meant for the opposite core.

At any time, a core can read back its `DOORBELL_OUT_SET` or `DOORBELL_OUT_CLR` register (they return the same result) to see the status of doorbell interrupts posted to the opposite core. Likewise, reading either `DOORBELL_IN_SET` or `DOORBELL_IN_CLR` returns the status of doorbell interrupts posted to this core.

ⓘ NOTE

RP2350 has separate per-core doorbell interrupt signals and doorbell registers for Secure and Non-secure SIO banks. Non-secure doorbells are posted on `SIO_IRQ_BELL_NS`, interrupt number [28](#). See [Section 3.1.1](#).

3.1.7. Integer Divider

RP2040's memory-mapped integer divider peripheral is not present on RP2350, since the processors support divide instructions. The address space previously allocated for the divider registers is now reserved.

3.1.8. RISC-V Platform Timer

This 64-bit timer is a standard peripheral described in the RISC-V privileged specification, usable equally by the Arm and RISC-V processors on RP2350. It drives the per-core [SIO_IRQ_MTIMECMP](#) system-level interrupt (Section 3.2), as well as the [mip.mtip](#) timer interrupt on the RISC-V processors.

There is a single 64-bit counter, shared between both cores. The low and high half can be accessed through the [MTIME](#) and [MTIMEH](#) SIO registers. Use the following procedure to safely read the 64-bit time using 32-bit register accesses:

1. Read the upper half, [MTIMEH](#).
2. Read the lower half, [MTIME](#).
3. Read the upper half again.
4. Loop if the two upper-half reads returned different values.

This is similar to the procedure for reading RP2350 system timers (Section 12.8). The loop should only happen once, when the timer is read at exactly the instant of a 32-bit rollover, and even this is only occasional. If you require constant-time operation, you can instead zero the lower half when the two upper-half reads differ.

Timer interrupts are generated based on a per-core 64-bit time comparison value, accessed through the [MTIMECMP](#) and [MTIMECMPPH](#) SIO registers. Each core gets its own copy of these registers, accessed at the same address. The per-core interrupt is asserted whenever the current time indicated in the [MTIME](#) registers is greater than or equal to that core's [MTIMECMP](#). Use the following sequence to write a new 64-bit timer comparison value without causing spurious interrupts:

1. Write all-ones to [MTIMECMP](#) (guaranteed greater than or equal to the old value, *and* the lower half of the target value).
2. Write the upper half of the target value to [MTIMECMPPH](#) (combined 64-bit value is still greater than or equal to the target value).
3. Write the lower half of the target value to [MTIMECMP](#).

The RISC-V timer can count either ticks from the system-level tick generator (Section 8.5), or system clock cycles, selected by the [MTIME_CTRL](#) register. Use a 1 microsecond time base for compatibility with most RISC-V software.

3.1.9. TMDS Encoder

Each core is equipped with an implementation of the TMDS encode algorithm described in chapter 3 of the DVI 1.0 specification. In general, the HSTX peripheral (Section 12.11) supports lower processor overhead for DVI-D output as well as a wider range of pixel formats, but the SIO TMDS encoders are included for use with non-HSTX-capable GPIOs.

The [TMDS_CTRL](#) register allows configuration of a number of input pixel formats, from 16-bit RGB down to 1-bit monochrome. Once the encoder has been set up, the processor writes 32 bits of colour data at a time to [TMDS_WDATA](#), and then reads TMDS data symbols from the output registers. Depending on the pixel format, there may be multiple TMDS symbols read for each write to [TMDS_WDATA](#). There are no stalls: encoding is limited entirely by the processor's load/store bandwidth, up to one 32-bit read or write per cycle per core.

To allow for framebuffer/scanbuffer resolution lower than the display resolution, the output registers have both peek and pop aliases (e.g. [TMDS_PEEK_SINGLE](#) and [TMDS_POP_SINGLE](#)). Reading either register advances the encoder's DC balance counter, but only the pop alias shifts the colour data in [TMDS_WDATA](#) so that multiple correctly-DC-balanced TMDS symbols can be generated from the same input pixel.

The TMDS encoder peripherals are not duplicated over security domains. They are assigned to the Secure SIO at reset, and can be reassigned to the Non-secure SIO using the [PERI_NONSEC](#) register.

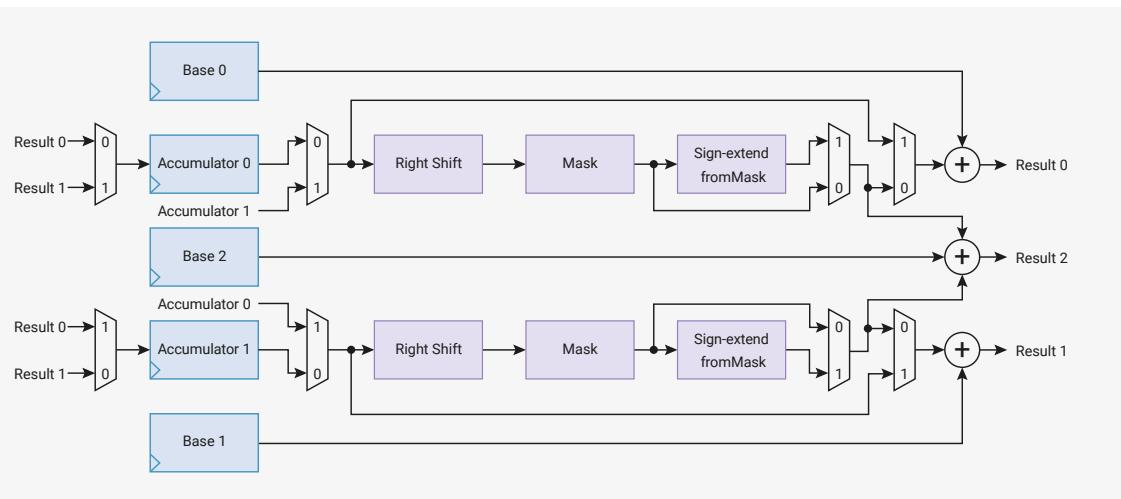
3.1.10. Interpolator

Each core is equipped with two *interpolators* (`INTERP0` and `INTERP1`) which can accelerate tasks by combining certain pre-configured operations into a single processor cycle. Intended for cases where the pre-configured operation repeats many times, interpolators result in code which uses both fewer CPU cycles and fewer CPU registers in time-critical sections.

The interpolators already accelerate audio operations within the SDK. Their flexible configuration makes it possible to optimise many other tasks, including:

- quantization
- dithering
- table lookup address generation
- affine texture mapping
- decompression
- linear feedback

Figure 7. An interpolator. The two accumulator registers and three base registers have single-cycle read/write access from the processor. The interpolator is organised into two lanes, which perform masking, shifting and sign-extension operations on the two accumulators. This produces three possible results, by adding the intermediate shift/mask values to the three base registers. From left to right, the multiplexers on each lane are controlled by the following flags in the `CTRL` registers: `CROSS_RESULT`, `CROSS_INPUT_SIGNED`, and `ADD_RAW`.



The processor can write or read any interpolator register in one cycle, and the results are ready on the next cycle. The processor can also perform an addition on one of the two accumulators `ACCUM0` or `ACCUM1` by writing to the corresponding `ACCUMx_ADD` register.

The three results are available in the read-only locations `PEEK0`, `PEEK1`, `PEEK2`. Reading from these locations does not change the state of the interpolator. The results are also aliased at the locations `POP0`, `POP1`, `POP2`; reading from a `POPx` alias returns the same result as the corresponding `PEEKx`, and simultaneously writes back the lane results to the accumulators. Use the `POPx` aliases to advance the state of interpolator each time a result is read.

You can adjust interpolator behaviour with the following operational modes:

- fractional blending between two values
- clamping values to restrict them within a given range.

The following example shows a trivial example of *popping* a lane result to produce simple iterative feedback.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/hello_interp/hello_interp.c Lines 11 - 23

```

11 void times_table() {
12     puts("9 times table:");
13
14     // Initialise lane 0 on interp0 on this core
15     interp_config cfg = interp_default_config();
16     interp_set_config(interp0, 0, &cfg);

```

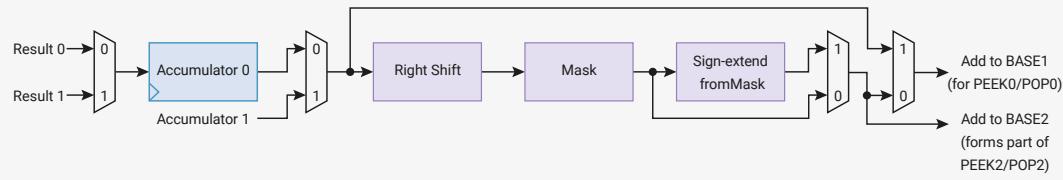
```

17
18     interp0->accum[0] = 0;
19     interp0->base[0] = 9;
20
21     for (int i = 0; i < 10; ++i)
22         printf("%d\n", interp0->pop[0]);
23 }

```

3.1.10.1. Lane Operations

Figure 8. Each lane of each interpolator can be configured to perform mask, shift and sign-extension on one of the accumulators. This is fed into adders which produce final results, which may optionally be fed back into the accumulators with each read. The datapath can be configured using a handful of 32-bit multiplexers. From left to right, these are controlled by the following CTRL flags: CROSS_RESULT, CROSS_INPUT, SIGNED, and ADD_RAW.



Each lane performs these three operations, in sequence:

- A right shift by `CTRL_LANEx_SHIFT` (0 to 31 bits)
- A mask of bits from `CTRL_LANEx_MASK_LSB` to `CTRL_LANEx_MASK_MSB` inclusive (each ranging from bit 0 to bit 31)
- A sign extension from the top of the mask, i.e. take bit `CTRL_LANEx_MASK_MSB` and OR it into all more-significant bits, if `CTRL_LANEx_SIGNED` is set

For example, if:

- `ACCUM0` = `0xdeadbeef`
- `CTRL_LANE0_SHIFT` = 8
- `CTRL_LANE0_MASK_LSB` = 4
- `CTRL_LANE0_MASK_MSB` = 7
- `CTRL_SIGNED` = 1

Then lane 0 would produce the following results at each stage:

- Right shift by 8 to produce `0x00deadbe`
- Mask bits 7 to 4 to produce `0x00deadbe & 0x000000f0` = `0x000000b0`
- Sign-extend up from bit 7 to produce `0xffffffffb0`

In software:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/interp.hello_interp/interp.hello_interp.c Lines 25 - 46

```

25 void moving_mask() {
26     interp_config cfg = interp_default_config();
27     interp0->accum[0] = 0x1234abcd;
28
29     puts("Masking:");
30     printf("ACCUM0 = %08x\n", interp0->accum[0]);
31     for (int i = 0; i < 8; ++i) {
32         // LSB, then MSB. These are inclusive, so 0,31 means "the entire 32 bit register"
33         interp_config_set_mask(&cfg, i * 4, i * 4 + 3);
34         interp_set_config(interp0, 0, &cfg);
35         // Reading from ACCUMx_ADD returns the raw lane shift and mask value, without BASEx
36         // added
37         printf("Nibble %d: %08x\n", i, interp0->add_raw[0]);
38 }

```

```

37     }
38
39     puts("Masking with sign extension:");
40     interp_config_set_signed(&cfg, true);
41     for (int i = 0; i < 8; ++i) {
42         interp_config_set_mask(&cfg, i * 4, i * 4 + 3);
43         interp_set_config(interp0, 0, &cfg);
44         printf("Nibble %d: %08x\n", i, interp0->add_raw[0]);
45     }
46 }
```

The above example should print the following:

```

ACCUM0 = 1234abcd
Nibble 0: 0000000d
Nibble 1: 000000c0
Nibble 2: 00000b00
Nibble 3: 0000a000
Nibble 4: 00040000
Nibble 5: 00300000
Nibble 6: 02000000
Nibble 7: 10000000
Masking with sign extension:
Nibble 0: ffffffd
Nibble 1: ffffffc0
Nibble 2: fffffb00
Nibble 3: fffffa000
Nibble 4: 00040000
Nibble 5: 00300000
Nibble 6: 02000000
Nibble 7: 10000000
```

Changing the result and input multiplexers can create feedback between the accumulators. This is useful for audio dithering.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/hello_interp/hello_interp.c Lines 48 - 66

```

48 void cross_lanes() {
49     interp_config cfg = interp_default_config();
50     interp_config_set_cross_result(&cfg, true);
51     // ACCUM0 gets lane 1 result:
52     interp_set_config(interp0, 0, &cfg);
53     // ACCUM1 gets lane 0 result:
54     interp_set_config(interp0, 1, &cfg);
55
56     interp0->accum[0] = 123;
57     interp0->accum[1] = 456;
58     interp0->base[0] = 1;
59     interp0->base[1] = 0;
60     puts("Lane result crossover:");
61     for (int i = 0; i < 10; ++i) {
62         uint32_t peek0 = interp0->peek[0];
63         uint32_t pop1 = interp0->pop[1];
64         printf("PEEK0, POP1: %d, %d\n", peek0, pop1);
65     }
66 }
```

This should print the following :

```

PEEK0, POP1: 124, 456
PEEK0, POP1: 457, 124
PEEK0, POP1: 125, 457
PEEK0, POP1: 458, 125
PEEK0, POP1: 126, 458
PEEK0, POP1: 459, 126
PEEK0, POP1: 127, 459
PEEK0, POP1: 460, 127
PEEK0, POP1: 128, 460
PEEK0, POP1: 461, 128

```

3.1.10.2. Blend Mode

Blend mode is available on `INTERP0` on each core, and is enabled by the `CTRL_LANE0_BLEND` control flag. It performs linear interpolation, which we define as follows:

$$x = x_0 + \alpha(x_1 - x_0), \text{ for } 0 \leq \alpha < 1$$

Where x_0 is the register `BASE0`, x_1 is the register `BASE1`, and α is a fractional value formed from the least significant 8 bits of the lane 1 shift and mask value.

Blend mode differs from normal mode in the following ways:

- `PEEK0, POP0` return the 8-bit alpha value (the 8 LSBs of the lane 1 shift and mask value), with zeroes in result bits 31 down to 24.
- `PEEK1, POP1` return the linear interpolation between `BASE0` and `BASE1`
- `PEEK2, POP2` do not include lane 1 result in the addition (i.e. it is `BASE2 + lane 0 shift and mask value`)

The result of the linear interpolation is equal to `BASE0` when the alpha value is 0, and equal to `BASE0 + 255/256 * (BASE1 - BASE0)` when the alpha value is all-ones.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/hello_interp/hello_interp.c Lines 68 - 87

```

68 void simple_blend1() {
69     puts("Simple blend 1:");
70
71     interp_config cfg = interp_default_config();
72     interp_config_set_blend(&cfg, true);
73     interp_set_config(interp0, 0, &cfg);
74
75     cfg = interp_default_config();
76     interp_set_config(interp0, 1, &cfg);
77
78     interp0->base[0] = 500;
79     interp0->base[1] = 1000;
80
81     for (int i = 0; i <= 6; i++) {
82         // set fraction to value between 0 and 255
83         interp0->accum[1] = 255 * i / 6;
84         // ≈ 500 + (1000 - 500) * i / 6;
85         printf("%d\n", (int) interp0->peek[1]);
86     }
87 }

```

This should print the following (note the 255/256 resulting in 998 not 1000):

```
500
582
666
748
832
914
998
```

`CTRL_LANE1_SIGNED` controls whether `BASE0` and `BASE1` are sign-extended for this interpolation (this sign extension is required because the interpolation produces an intermediate product value 40 bits in size). `CTRL_LANE0_SIGNED` continues to control the sign extension of the lane 0 intermediate result in `PEEK2`, `POP2` as normal.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/hello_interp/hello_interp.c Lines 90 - 121

```
90 void print_simple_blend2_results(bool is_signed) {
91     // lane 1 signed flag controls whether base 0/1 are treated as signed or unsigned
92     interp_config cfg = interp_default_config();
93     interp_config_set_signed(&cfg, is_signed);
94     interp_set_config(interp0, 1, &cfg);
95
96     for (int i = 0; i <= 6; i++) {
97         interp0->accum[1] = 255 * i / 6;
98         if (is_signed) {
99             printf("%d\n", (int) interp0->peek[1]);
100        } else {
101            printf("0x%08x\n", (uint) interp0->peek[1]);
102        }
103    }
104 }
105
106 void simple_blend2() {
107     puts("Simple blend 2:");
108
109     interp_config cfg = interp_default_config();
110     interp_config_set_blend(&cfg, true);
111     interp_set_config(interp0, 0, &cfg);
112
113     interp0->base[0] = (uint32_t) -1000;
114     interp0->base[1] = 1000;
115
116     puts("signed:");
117     print_simple_blend2_results(true);
118
119     puts("unsigned:");
120     print_simple_blend2_results(false);
121 }
```

This should print the following:

```
signed:
-1000
-672
-336
-8
328
656
992
unsigned:
```

```
0xfffffc18
0xd5ffd60
0xaafffeb0
0x80fffff8
0x56000148
0x2c000290
0x010003e0
```

Finally, in blend mode when using the `BASE_1AND0` register to send a 16-bit value to each of `BASE0` and `BASE1` with a single 32-bit write, the sign-extension of these 16-bit values to full 32-bit values during the write is controlled by `CTRL_LANE1_SIGNED` for both bases, as opposed to non-blend-mode operation, where `CTRL_LANE0_SIGNED` affects extension into `BASE0` and `CTRL_LANE1_SIGNED` affects extension into `BASE1`.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/hello_interp/hello_interp.c Lines 124 - 145

```
124 void simple_blend3() {
125     puts("Simple blend 3:");
126
127     interp_config cfg = interp_default_config();
128     interp_config_set_blend(&cfg, true);
129     interp_set_config(interp0, 0, &cfg);
130
131     cfg = interp_default_config();
132     interp_set_config(interp0, 1, &cfg);
133
134     interp0->accum[1] = 128;
135     interp0->base01 = 0x30005000;
136     printf("0x%08x\n", (int) interp0->peek[1]);
137     interp0->base01 = 0xe000f000;
138     printf("0x%08x\n", (int) interp0->peek[1]);
139
140     interp_config_set_signed(&cfg, true);
141     interp_set_config(interp0, 1, &cfg);
142
143     interp0->base01 = 0xe000f000;
144     printf("0x%08x\n", (int) interp0->peek[1]);
145 }
```

This should print the following:

```
0x00004000
0x0000e800
0xfffffe800
```

3.1.10.3. Clamp Mode

Clamp mode is available on `INTERP1` on each core. To enable clamp mode, set the `CTRL_LANE0_CLAMP` control flag to high. In clamp mode, the `PEEK0/POP0` result is the lane value (shifted, masked, sign-extended `ACCUM0`) clamped between `BASE0` and `BASE1`. In other words, if the lane value is less than `BASE0`, a value of `BASE0` is produced; if greater than `BASE1`, a value of `BASE1` is produced; otherwise, the value passes through. No addition is performed. The signedness of these comparisons is controlled by the `CTRL_LANE0_SIGNED` flag.

Other than this, the interpolator behaves the same as in normal mode.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/Hello_Interp/Hello_Interp.c Lines 193 - 211

```

193 void clamp() {
194     puts("Clamp:");
195     interp_config cfg = interp_default_config();
196     interp_config_set_clamp(&cfg, true);
197     interp_config_set_shift(&cfg, 2);
198     // set mask according to new position of sign bit..
199     interp_config_set_mask(&cfg, 0, 29);
200     // ...so that the shifted value is correctly sign extended
201     interp_config_set_signed(&cfg, true);
202     interp_set_config(interp1, 0, &cfg);
203
204     interp1->base[0] = 0;
205     interp1->base[1] = 255;
206
207     for (int i = -1024; i <= 1024; i += 256) {
208         interp1->accum[0] = i;
209         printf("%d\t%d\n", i, (int) interp1->peek[0]);
210     }
211 }
```

This should print the following:

```

-1024 0
-768 0
-512 0
-256 0
0 0
256 64
512 128
768 192
1024 255
```

3.1.10.4. Sample Use Case: Linear Interpolation

Linear interpolation combines blend mode with other interpolator functionality. In this example, `ACCUM0` tracks a fixed-point (integer/fraction) position within a list of values to be interpolated. Lane 0 is used to produce an address into the value array for the integer part of the position. The fractional part of the position is shifted to produce a value from 0-255 for the blend. The blend is performed between two consecutive values in the array.

Finally the fractional position is updated via a single write to `ACCUM0_ADD_RAW`.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/Hello_Interp/Hello_Interp.c Lines 147 - 191

```

147 void linear_interpolation() {
148     puts("Linear interpolation:");
149     const int uv_fractional_bits = 12;
150
151     // for lane 0
152     // shift and mask XXXX XXXX XXXX XXXX XXXX FFFF FFFF FFFF (accum 0)
153     // to          0000 0000 000X XXXX XXXX XXXX XXXX XXX0
154     // i.e. non fractional part times 2 (for uint16_t)
155     interp_config cfg = interp_default_config();
156     interp_config_set_shift(&cfg, uv_fractional_bits - 1);
157     interp_config_set_mask(&cfg, 1, 32 - uv_fractional_bits);
158     interp_config_set_blend(&cfg, true);
159     interp_set_config(interp0, 0, &cfg);
```

```

160
161    // for lane 1
162    // shift XXXX XXXX XXXX XXXX XXXX FFFF FFFF FFFF (accum 0 via cross input)
163    // to    0000 XXXX XXXX XXXX XXXX FFFF FFFF FFFF
164
165    cfg = interp_default_config();
166    interp_config_set_shift(&cfg, uv_fractional_bits - 8);
167    interp_config_set_signed(&cfg, true);
168    interp_config_set_cross_input(&cfg, true); // signed blending
169    interp_set_config(interp0, 1, &cfg);
170
171    int16_t samples[] = {0, 10, -20, -1000, 500};
172
173    // step is 1/4 in our fractional representation
174    uint step = (1 << uv_fractional_bits) / 4;
175
176    interp0->accum[0] = 0; // initial sample_offset;
177    interp0->base[2] = (uintptr_t) samples;
178    for (int i = 0; i < 16; i++) {
179        // result2 = samples + (lane0 raw result)
180        // i.e. ptr to the first of two samples to blend between
181        int16_t *sample_pair = (int16_t *) interp0->peek[2];
182        interp0->base[0] = sample_pair[0];
183        interp0->base[1] = sample_pair[1];
184        uint32_t peek1 = interp0->peek[1];
185        uint32_t add_raw1 = interp0->add_raw[1];
186        printf("%d\t%d% between %d and %d)\n", (int) peek1,
187               100 * (add_raw1 & 0xff) / 0xff,
188               sample_pair[0], sample_pair[1]);
189        interp0->add_raw[0] = step;
190    }
191 }

```

This should print the following:

```

0      (0% between 0 and 10)
2      (25% between 0 and 10)
5      (50% between 0 and 10)
7      (75% between 0 and 10)
10     (0% between 10 and -20)
2      (25% between 10 and -20)
-5     (50% between 10 and -20)
-13    (75% between 10 and -20)
-20    (0% between -20 and -1000)
-265   (25% between -20 and -1000)
-510   (50% between -20 and -1000)
-755   (75% between -20 and -1000)
-1000  (0% between -1000 and 500)
-625   (25% between -1000 and 500)
-250   (50% between -1000 and 500)
125    (75% between -1000 and 500)

```

This method is used for fast approximate audio upscaling in the SDK.

3.1.10.5. Sample Use Case: Simple Affine Texture Mapping

Simple affine texture mapping can be implemented by using fixed-point arithmetic for texture coordinates, and stepping a fixed amount in each coordinate for every pixel in a scanline. The integer parts of the texture coordinates form an

address into the texture. Reading from `POP2` adds the offset to the texture base pointer. The processor loads the resulting address to sample a pixel colour from the texture.

By using two lanes, all three base values, and the `CTRL_LANEx_ADD_RAW` flag, you can use the interpolator to reduce an expensive CPU operation to a single cycle iteration.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/interp/hello_interp/hello_interp.c Lines 214 - 272

```

214 void texture_mapping_setup(uint8_t *texture, uint texture_width_bits, uint
215                             texture_height_bits,
216                             uint uv_fractional_bits) {
217     interp_config cfg = interp_default_config();
218     // set add_raw flag to use raw (un-shifted and un-masked) lane accumulator value when
219     // adding
220     // it to the the lane base to make the lane result
221     interp_config_set_add_raw(&cfg, true);
222     interp_config_set_shift(&cfg, uv_fractional_bits);
223     interp_config_set_mask(&cfg, 0, texture_width_bits - 1);
224     interp_set_config(interp0, 0, &cfg);
225
226     interp_config_set_shift(&cfg, uv_fractional_bits - texture_width_bits);
227     interp_config_set_mask(&cfg, texture_width_bits, texture_width_bits +
228                           texture_height_bits - 1);
229     interp_set_config(interp0, 1, &cfg);
230
231     interp0->base[2] = (uintptr_t) texture;
232 }
233
234 void texture_mapped_span(uint8_t *output, uint32_t u, uint32_t v, uint32_t du, uint32_t dv,
235                          uint count) {
236     // u, v are texture coordinates in fixed point with uv_fractional_bits fractional bits
237     // du, dv are texture coordinate steps across the span in same fixed point.
238     interp0->accum[0] = u;
239     interp0->base[0] = du;
240     interp0->accum[1] = v;
241     interp0->base[1] = dv;
242     for (uint i = 0; i < count; i++) {
243         // equivalent to
244         // uint32_t sm_result0 = (accum0 >> uv_fractional_bits) & (1 << (texture_width_bits -
245         // 1));
246         // uint32_t sm_result1 = (accum1 >> uv_fractional_bits) & (1 << (texture_height_bits -
247         // 1));
248         // uint8_t *address = texture + sm_result0 + (sm_result1 << texture_width_bits);
249         // output[i] = *address;
250         // accum0 = du + accum0;
251         // accum1 = dv + accum1;
252
253         // result2 is the texture address for the current pixel;
254         // popping the result advances to the next iteration
255         output[i] = *(uint8_t *) interp0->pop[2];
256     }
257 }
258
259 void texture_mapping() {
260     puts("Affine Texture mapping (with texture wrap):");
261
262     uint8_t texture[] = {
263         0x00, 0x01, 0x02, 0x03,
264         0x10, 0x11, 0x12, 0x13,
265         0x20, 0x21, 0x22, 0x23,
266         0x30, 0x31, 0x32, 0x33,
267     };
268     // 4x4 texture

```

```

263     texture_mapping_setup(texture, 2, 2, 16);
264     uint8_t output[12];
265     uint32_t du = 65536 / 2; // step of 1/2
266     uint32_t dv = 65536 / 3; // step of 1/3
267     texture_mapped_span(output, 0, 0, du, dv, 12);
268
269     for (uint i = 0; i < 12; i++) {
270         printf("0x%02x\n", output[i]);
271     }
272 }
```

This should print the following:

```

0x00
0x00
0x01
0x01
0x12
0x12
0x13
0x23
0x20
0x20
0x31
0x31
```

3.1.11. List of Registers

The SIO registers start at a base address of **0xd0000000** (defined as **SIO_BASE** in SDK).

Table 16. List of SIO registers

Offset	Name	Info
0x000	CPUID	Processor core identifier
0x004	GPIO_IN	Input value for GPIO0...31. In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) appear as zero.
0x008	GPIO_HI_IN	Input value on GPIO32...47, QSPI IOs and USB pins In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) appear as zero.
0x010	GPIO_OUT	GPIO0...31 output value

Offset	Name	Info
0x014	GPIO_HI_OUT	<p>Output value for GPIO32...47, QSPI IOs and USB pins.</p> <p>Write to set output level (1/0 → high/low). Reading back gives the last value written, NOT the input value from the pins. If core 0 and core 1 both write to GPIO_HI_OUT simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result.</p> <p>In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) ignore writes, and their output status reads back as zero. This is also true for SET/CLR/XOR aliases of this register.</p>
0x018	GPIO_OUT_SET	GPIO0...31 output value set
0x01c	GPIO_HI_OUT_SET	<p>Output value set for GPIO32..47, QSPI IOs and USB pins.</p> <p>Perform an atomic bit-set on GPIO_HI_OUT, i.e. <code>GPIO_HI_OUT = wdata</code></p>
0x020	GPIO_OUT_CLR	GPIO0...31 output value clear
0x024	GPIO_HI_OUT_CLR	<p>Output value clear for GPIO32..47, QSPI IOs and USB pins.</p> <p>Perform an atomic bit-clear on GPIO_HI_OUT, i.e. <code>GPIO_HI_OUT &=~wdata</code></p>
0x028	GPIO_OUT_XOR	GPIO0...31 output value XOR
0x02c	GPIO_HI_OUT_XOR	<p>Output value XOR for GPIO32..47, QSPI IOs and USB pins.</p> <p>Perform an atomic bitwise XOR on GPIO_HI_OUT, i.e. <code>GPIO_HI_OUT ^= wdata</code></p>
0x030	GPIO_OE	GPIO0...31 output enable
0x034	GPIO_HI_OE	<p>Output enable value for GPIO32...47, QSPI IOs and USB pins.</p> <p>Write output enable (1/0 → output/input). Reading back gives the last value written. If core 0 and core 1 both write to GPIO_HI_OE simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result.</p> <p>In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) ignore writes, and their output status reads back as zero. This is also true for SET/CLR/XOR aliases of this register.</p>
0x038	GPIO_OE_SET	GPIO0...31 output enable set
0x03c	GPIO_HI_OE_SET	<p>Output enable set for GPIO32...47, QSPI IOs and USB pins.</p> <p>Perform an atomic bit-set on GPIO_HI_OE, i.e. <code>GPIO_HI_OE = wdata</code></p>
0x040	GPIO_OE_CLR	GPIO0...31 output enable clear
0x044	GPIO_HI_OE_CLR	<p>Output enable clear for GPIO32...47, QSPI IOs and USB pins.</p> <p>Perform an atomic bit-clear on GPIO_HI_OE, i.e. <code>GPIO_HI_OE &=~wdata</code></p>
0x048	GPIO_OE_XOR	GPIO0...31 output enable XOR
0x04c	GPIO_HI_OE_XOR	<p>Output enable XOR for GPIO32..47, QSPI IOs and USB pins.</p> <p>Perform an atomic bitwise XOR on GPIO_HI_OE, i.e. <code>GPIO_HI_OE ^= wdata</code></p>

Offset	Name	Info
0x050	FIFO_ST	Status register for inter-core FIFOs (mailboxes).
0x054	FIFO_WR	Write access to this core's TX FIFO
0x058	FIFO_RD	Read access to this core's RX FIFO
0x05c	SPINLOCK_ST	Spinlock state
0x080	INTERP0_ACCUM0	Read/write access to accumulator 0
0x084	INTERP0_ACCUM1	Read/write access to accumulator 1
0x088	INTERP0_BASE0	Read/write access to BASE0 register.
0x08c	INTERP0_BASE1	Read/write access to BASE1 register.
0x090	INTERP0_BASE2	Read/write access to BASE2 register.
0x094	INTERP0_POP_LANE0	Read LANE0 result, and simultaneously write lane results to both accumulators (POP).
0x098	INTERP0_POP_LANE1	Read LANE1 result, and simultaneously write lane results to both accumulators (POP).
0x09c	INTERP0_POP_FULL	Read FULL result, and simultaneously write lane results to both accumulators (POP).
0x0a0	INTERP0_PEEK_LANE0	Read LANE0 result, without altering any internal state (PEEK).
0x0a4	INTERP0_PEEK_LANE1	Read LANE1 result, without altering any internal state (PEEK).
0x0a8	INTERP0_PEEK_FULL	Read FULL result, without altering any internal state (PEEK).
0x0ac	INTERP0_CTRL_LANE0	Control register for lane 0
0x0b0	INTERP0_CTRL_LANE1	Control register for lane 1
0x0b4	INTERP0_ACCUM0_ADD	Values written here are atomically added to ACCUM0
0x0b8	INTERP0_ACCUM1_ADD	Values written here are atomically added to ACCUM1
0x0bc	INTERP0_BASE_1AND0	On write, the lower 16 bits go to BASE0, upper bits to BASE1 simultaneously.
0x0c0	INTERP1_ACCUM0	Read/write access to accumulator 0
0x0c4	INTERP1_ACCUM1	Read/write access to accumulator 1
0x0c8	INTERP1_BASE0	Read/write access to BASE0 register.
0x0cc	INTERP1_BASE1	Read/write access to BASE1 register.
0x0d0	INTERP1_BASE2	Read/write access to BASE2 register.
0x0d4	INTERP1_POP_LANE0	Read LANE0 result, and simultaneously write lane results to both accumulators (POP).
0x0d8	INTERP1_POP_LANE1	Read LANE1 result, and simultaneously write lane results to both accumulators (POP).
0x0dc	INTERP1_POP_FULL	Read FULL result, and simultaneously write lane results to both accumulators (POP).
0x0e0	INTERP1_PEEK_LANE0	Read LANE0 result, without altering any internal state (PEEK).
0x0e4	INTERP1_PEEK_LANE1	Read LANE1 result, without altering any internal state (PEEK).
0x0e8	INTERP1_PEEK_FULL	Read FULL result, without altering any internal state (PEEK).

Offset	Name	Info
0x0ec	INTERP1_CTRL_LANE0	Control register for lane 0
0x0f0	INTERP1_CTRL_LANE1	Control register for lane 1
0x0f4	INTERP1_ACCUM0_ADD	Values written here are atomically added to ACCUM0
0x0f8	INTERP1_ACCUM1_ADD	Values written here are atomically added to ACCUM1
0x0fc	INTERP1_BASE_1AND0	On write, the lower 16 bits go to BASE0, upper bits to BASE1 simultaneously.
0x100	SPINLOCK0	Spinlock register 0
0x104	SPINLOCK1	Spinlock register 1
0x108	SPINLOCK2	Spinlock register 2
0x10c	SPINLOCK3	Spinlock register 3
0x110	SPINLOCK4	Spinlock register 4
0x114	SPINLOCK5	Spinlock register 5
0x118	SPINLOCK6	Spinlock register 6
0x11c	SPINLOCK7	Spinlock register 7
0x120	SPINLOCK8	Spinlock register 8
0x124	SPINLOCK9	Spinlock register 9
0x128	SPINLOCK10	Spinlock register 10
0x12c	SPINLOCK11	Spinlock register 11
0x130	SPINLOCK12	Spinlock register 12
0x134	SPINLOCK13	Spinlock register 13
0x138	SPINLOCK14	Spinlock register 14
0x13c	SPINLOCK15	Spinlock register 15
0x140	SPINLOCK16	Spinlock register 16
0x144	SPINLOCK17	Spinlock register 17
0x148	SPINLOCK18	Spinlock register 18
0x14c	SPINLOCK19	Spinlock register 19
0x150	SPINLOCK20	Spinlock register 20
0x154	SPINLOCK21	Spinlock register 21
0x158	SPINLOCK22	Spinlock register 22
0x15c	SPINLOCK23	Spinlock register 23
0x160	SPINLOCK24	Spinlock register 24
0x164	SPINLOCK25	Spinlock register 25
0x168	SPINLOCK26	Spinlock register 26
0x16c	SPINLOCK27	Spinlock register 27
0x170	SPINLOCK28	Spinlock register 28
0x174	SPINLOCK29	Spinlock register 29

Offset	Name	Info
0x178	SPINLOCK30	Spinlock register 30
0x17c	SPINLOCK31	Spinlock register 31
0x180	DOORBELL_OUT_SET	<p>Trigger a doorbell interrupt on the opposite core.</p> <p>Write 1 to a bit to set the corresponding bit in DOORBELL_IN on the opposite core. This raises the opposite core's doorbell interrupt.</p> <p>Read to get the status of the doorbells currently asserted on the opposite core. This is equivalent to that core reading its own DOORBELL_IN status.</p>
0x184	DOORBELL_OUT_CLR	<p>Clear doorbells which have been posted to the opposite core. This register is intended for debugging and initialisation purposes.</p> <p>Writing 1 to a bit in DOORBELL_OUT_CLR clears the corresponding bit in DOORBELL_IN on the opposite core. Clearing all bits will cause that core's doorbell interrupt to deassert. Since the usual order of events is for software to send events using DOORBELL_OUT_SET, and acknowledge incoming events by writing to DOORBELL_IN_CLR, this register should be used with caution to avoid race conditions.</p> <p>Reading returns the status of the doorbells currently asserted on the other core, i.e. is equivalent to that core reading its own DOORBELL_IN status.</p>
0x188	DOORBELL_IN_SET	Write 1s to trigger doorbell interrupts on this core. Read to get status of doorbells currently asserted on this core.
0x18c	DOORBELL_IN_CLR	<p>Check and acknowledge doorbells posted to this core. This core's doorbell interrupt is asserted when any bit in this register is 1.</p> <p>Write 1 to each bit to clear that bit. The doorbell interrupt deasserts once all bits are cleared. Read to get status of doorbells currently asserted on this core.</p>
0x190	PERI_NONSEC	<p>Detach certain core-local peripherals from Secure SIO, and attach them to Non-secure SIO, so that Non-secure software can use them. Attempting to access one of these peripherals from the Secure SIO when it is attached to the Non-secure SIO, or vice versa, will generate a bus error.</p> <p>This register is per-core, and is only present on the Secure SIO.</p> <p>Most SIO hardware is duplicated across the Secure and Non-secure SIO, so is not listed in this register.</p>

Offset	Name	Info
0x1a0	RISCV_SOFTIRQ	<p>Control the assertion of the standard software interrupt (MIP.MSIP) on the RISC-V cores.</p> <p>Unlike the RISC-V timer, this interrupt is not routed to a normal system-level interrupt line, so can not be used by the Arm cores.</p> <p>It is safe for both cores to write to this register on the same cycle. The set/clear effect is accumulated across both cores, and then applied. If a flag is both set and cleared on the same cycle, only the set takes effect.</p>
0x1a4	MTIME_CTRL	<p>Control register for the RISC-V 64-bit Machine-mode timer. This timer is only present in the Secure SIO, so is only accessible to an Arm core in Secure mode or a RISC-V core in Machine mode.</p> <p>Note whilst this timer follows the RISC-V privileged specification, it is equally usable by the Arm cores. The interrupts are routed to normal system-level interrupt lines as well as to the MIP.MTIP inputs on the RISC-V cores.</p>
0x1b0	MTIME	<p>Read/write access to the high half of RISC-V Machine-mode timer. This register is shared between both cores. If both cores write on the same cycle, core 1 takes precedence.</p>
0x1b4	MTIMEH	<p>Read/write access to the high half of RISC-V Machine-mode timer. This register is shared between both cores. If both cores write on the same cycle, core 1 takes precedence.</p>
0x1b8	MTIMECMP	<p>Low half of RISC-V Machine-mode timer comparator. This register is core-local, i.e., each core gets a copy of this register, with the comparison result routed to its own interrupt line.</p> <p>The timer interrupt is asserted whenever MTIME is greater than or equal to MTIMECMP. This comparison is unsigned, and performed on the full 64-bit values.</p>
0x1bc	MTIMECMPPH	<p>High half of RISC-V Machine-mode timer comparator. This register is core-local.</p> <p>The timer interrupt is asserted whenever MTIME is greater than or equal to MTIMECMPPH. This comparison is unsigned, and performed on the full 64-bit values.</p>
0x1c0	TMDS_CTRL	Control register for TMDS encoder.
0x1c4	TMDS_WDATA	Write-only access to the TMDS colour data register.
0x1c8	TMDS_PEEK_SINGLE	<p>Get the encoding of one pixel's worth of colour data, packed into a 32-bit value (3x10-bit symbols).</p> <p>The PEEK alias does not shift the colour register when read, but still advances the running DC balance state of each encoder. This is useful for pixel doubling.</p>

Offset	Name	Info
0x1cc	TMDS_POP_SINGLE	<p>Get the encoding of one pixel's worth of colour data, packed into a 32-bit value. The packing is 5 chunks of 3 lanes times 2 bits (30 bits total). Each chunk contains two bits of a TMDS symbol per lane. This format is intended for shifting out with the HSTX peripheral on RP2350.</p> <p>The POP alias shifts the colour register when read, as well as advancing the running DC balance state of each encoder.</p>
0x1d0	TMDS_PEEK_DOUBLE_L0	<p>Get lane 0 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The PEEK alias does not shift the colour register when read, but still advances the lane 0 DC balance state. This is useful if all 3 lanes' worth of encode are to be read at once, rather than processing the entire scanline for one lane before moving to the next lane.</p>
0x1d4	TMDS_POP_DOUBLE_L0	<p>Get lane 0 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The POP alias shifts the colour register when read, according to the values of PIX_SHIFT and PIX2_NOSHIFT.</p>
0x1d8	TMDS_PEEK_DOUBLE_L1	<p>Get lane 1 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The PEEK alias does not shift the colour register when read, but still advances the lane 1 DC balance state. This is useful if all 3 lanes' worth of encode are to be read at once, rather than processing the entire scanline for one lane before moving to the next lane.</p>
0x1dc	TMDS_POP_DOUBLE_L1	<p>Get lane 1 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The POP alias shifts the colour register when read, according to the values of PIX_SHIFT and PIX2_NOSHIFT.</p>
0x1e0	TMDS_PEEK_DOUBLE_L2	<p>Get lane 2 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The PEEK alias does not shift the colour register when read, but still advances the lane 2 DC balance state. This is useful if all 3 lanes' worth of encode are to be read at once, rather than processing the entire scanline for one lane before moving to the next lane.</p>

Offset	Name	Info
0x1e4	TMDS_POP_DOUBLE_L2	Get lane 2 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word. The POP alias shifts the colour register when read, according to the values of PIX_SHIFT and PIX2_NOSHIFT.

SIO: CPUID Register

Offset: 0x000

Description

Processor core identifier

Table 17. CPUID Register

Bits	Description	Type	Reset
31:0	Value is 0 when read from processor core 0, and 1 when read from processor core 1.	RO	-

SIO: GPIO_IN Register

Offset: 0x004

Table 18. GPIO_IN Register

Bits	Description	Type	Reset
31:0	Input value for GPIO0...31. In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) appear as zero.	RO	0x00000000

SIO: GPIO_HI_IN Register

Offset: 0x008

Description

Input value on GPIO32...47, QSPI IOs and USB pins

In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) appear as zero.

Table 19. GPIO_HI_IN Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD	Input value on QSPI SD0 (MOSI), SD1 (MISO), SD2 and SD3 pins	RO	0x0
27	QSPI_CSN	Input value on QSPI CSn pin	RO	0x0
26	QSPI_SCK	Input value on QSPI SCK pin	RO	0x0
25	USB_DM	Input value on USB D- pin	RO	0x0
24	USB_DP	Input value on USB D+ pin	RO	0x0
23:16	Reserved.	-	-	-
15:0	GPIO	Input value on GPIO32...47	RO	0x0000

SIO: GPIO_OUT Register

Offset: 0x010

Description

GPIO0...31 output value

Table 20. *GPIO_OUT* Register

Bits	Description	Type	Reset
31:0	<p>Set output level (1/0 → high/low) for GPIO0...31. Reading back gives the last value written, NOT the input value from the pins.</p> <p>If core 0 and core 1 both write to GPIO_OUT simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result.</p> <p>In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) ignore writes, and their output status reads back as zero. This is also true for SET/CLR/XOR aliases of this register.</p>	RW	0x00000000

SIO: GPIO_HI_OUT Register

Offset: 0x014

Description

Output value for GPIO32...47, QSPI IOs and USB pins.

Write to set output level (1/0 → high/low). Reading back gives the last value written, NOT the input value from the pins. If core 0 and core 1 both write to GPIO_HI_OUT simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result.

In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) ignore writes, and their output status reads back as zero. This is also true for SET/CLR/XOR aliases of this register.

Table 21. *GPIO_HI_OUT* Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD	Output value for QSPI SD0 (MOSI), SD1 (MISO), SD2 and SD3 pins	RW	0x0
27	QSPI_CS_N	Output value for QSPI CS_N pin	RW	0x0
26	QSPI_SCK	Output value for QSPI SCK pin	RW	0x0
25	USB_DM	Output value for USB D- pin	RW	0x0
24	USB_DP	Output value for USB D+ pin	RW	0x0
23:16	Reserved.	-	-	-
15:0	GPIO	Output value for GPIO32...47	RW	0x0000

SIO: GPIO_OUT_SET Register

Offset: 0x018

Description

GPIO0...31 output value set

Table 22. *GPIO_OUT_SET* Register

Bits	Description	Type	Reset
31:0	Perform an atomic bit-set on GPIO_OUT, i.e. <code>GPIO_OUT = wdata</code>	WO	0x00000000

SIO: GPIO_HI_OUT_SET Register

Offset: 0x01c

Description

Output value set for GPIO32..47, QSPI IOs and USB pins.

Perform an atomic bit-set on GPIO_HI_OUT, i.e. `GPIO_HI_OUT |= wdata`

Table 23.
GPIO_HI_OUT_SET
Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		WO	0x0
27	QSPI_CSN		WO	0x0
26	QSPI_SCK		WO	0x0
25	USB_DM		WO	0x0
24	USB_DP		WO	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		WO	0x0000

SIO: GPIO_OUT_CLR Register

Offset: 0x020

Description

GPIO0...31 output value clear

Table 24.
GPIO_OUT_CLR
Register

Bits	Description	Type	Reset
31:0	Perform an atomic bit-clear on GPIO_OUT, i.e. <code>GPIO_OUT &= ~wdata</code>	WO	0x00000000

SIO: GPIO_HI_OUT_CLR Register

Offset: 0x024

Description

Output value clear for GPIO32..47, QSPI IOs and USB pins.

Perform an atomic bit-clear on GPIO_HI_OUT, i.e. `GPIO_HI_OUT &= ~wdata`

Table 25.
GPIO_HI_OUT_CLR
Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		WO	0x0
27	QSPI_CSN		WO	0x0
26	QSPI_SCK		WO	0x0
25	USB_DM		WO	0x0
24	USB_DP		WO	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		WO	0x0000

SIO: GPIO_OUT_XOR Register

Offset: 0x028

Description

GPIO0...31 output value XOR

Table 26.
GPIO_OUT_XOR
Register

Bits	Description	Type	Reset
31:0	Perform an atomic bitwise XOR on GPIO_OUT, i.e. <code>GPIO_OUT ^= wdata</code>	WO	0x00000000

SIO: GPIO_HI_OUT_XOR Register

Offset: 0x02c

Description

Output value XOR for GPIO32..47, QSPI IOs and USB pins.

Perform an atomic bitwise XOR on GPIO_HI_OUT, i.e. `GPIO_HI_OUT ^= wdata`

Table 27.
GPIO_HI_OUT_XOR
Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		WO	0x0
27	QSPI_CSN		WO	0x0
26	QSPI_SCK		WO	0x0
25	USB_DM		WO	0x0
24	USB_DP		WO	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		WO	0x0000

SIO: GPIO_OE Register

Offset: 0x030

Description

GPIO0...31 output enable

Table 28. GPIO_OE
Register

Bits	Description	Type	Reset
31:0	<p>Set output enable (1/0 → output/input) for GPIO0...31. Reading back gives the last value written.</p> <p>If core 0 and core 1 both write to GPIO_OE simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result.</p> <p>In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) ignore writes, and their output status reads back as zero. This is also true for SET/CLR/XOR aliases of this register.</p>	RW	0x00000000

SIO: GPIO_HI_OE Register

Offset: 0x034

Description

Output enable value for GPIO32..47, QSPI IOs and USB pins.

Write output enable (1/0 → output/input). Reading back gives the last value written. If core 0 and core 1 both write to GPIO_HI_OE simultaneously (or to a SET/CLR/XOR alias), the result is as though the write from core 0 took place first, and the write from core 1 was then applied to that intermediate result.

In the Non-secure SIO, Secure-only GPIOs (as per ACCESSCTRL) ignore writes, and their output status reads back as zero. This is also true for SET/CLR/XOR aliases of this register.

Table 29. GPIO_HI_OE Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD	Output enable value for QSPI SD0 (MOSI), SD1 (MISO), SD2 and SD3 pins	RW	0x0
27	QSPI_CSN	Output enable value for QSPI CSn pin	RW	0x0
26	QSPI_SCK	Output enable value for QSPI SCK pin	RW	0x0
25	USB_DM	Output enable value for USB D- pin	RW	0x0
24	USB_DP	Output enable value for USB D+ pin	RW	0x0
23:16	Reserved.	-	-	-
15:0	GPIO	Output enable value for GPIO32...47	RW	0x0000

SIO: GPIO_OE_SET Register

Offset: 0x038

Description

GPIO0...31 output enable set

Table 30. GPIO_OE_SET Register

Bits	Description	Type	Reset
31:0	Perform an atomic bit-set on GPIO_OE, i.e. <code>GPIO_OE = wdata</code>	WO	0x00000000

SIO: GPIO_HI_OE_SET Register

Offset: 0x03c

Description

Output enable set for GPIO32...47, QSPI IOs and USB pins.

Perform an atomic bit-set on GPIO_HI_OE, i.e. `GPIO_HI_OE |= wdata`

Table 31. GPIO_HI_OE_SET Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		WO	0x0
27	QSPI_CSN		WO	0x0
26	QSPI_SCK		WO	0x0
25	USB_DM		WO	0x0
24	USB_DP		WO	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		WO	0x0000

SIO: GPIO_OE_CLR Register

Offset: 0x040

Description

GPIO0...31 output enable clear

Table 32.
GPIO_OE_CLR Register

Bits	Description	Type	Reset
31:0	Perform an atomic bit-clear on GPIO_OE, i.e. <code>GPIO_OE &= ~wdata</code>	WO	0x00000000

SIO: GPIO_HI_OE_CLR Register

Offset: 0x044

Description

Output enable clear for GPIO32...47, QSPI IOs and USB pins.

Perform an atomic bit-clear on GPIO_HI_OE, i.e. `GPIO_HI_OE &= ~wdata`

Table 33.
GPIO_HI_OE_CLR Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		WO	0x0
27	QSPI_CSN		WO	0x0
26	QSPI_SCK		WO	0x0
25	USB_DM		WO	0x0
24	USB_DP		WO	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		WO	0x0000

SIO: GPIO_OE_XOR Register

Offset: 0x048

Description

GPIO0...31 output enable XOR

Table 34.
GPIO_OE_XOR Register

Bits	Description	Type	Reset
31:0	Perform an atomic bitword XOR on GPIO_OE, i.e. <code>GPIO_OE ^= wdata</code>	WO	0x00000000

SIO: GPIO_HI_OE_XOR Register

Offset: 0x04c

Description

Output enable XOR for GPIO32...47, QSPI IOs and USB pins.

Perform an atomic bitword XOR on GPIO_HI_OE, i.e. `GPIO_HI_OE ^= wdata`

Table 35.
GPIO_HI_OE_XOR Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		WO	0x0
27	QSPI_CSN		WO	0x0
26	QSPI_SCK		WO	0x0
25	USB_DM		WO	0x0
24	USB_DP		WO	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		WO	0x0000

SIO: FIFO_ST Register

Offset: 0x050**Description**

Status register for inter-core FIFOs (mailboxes).

There is one FIFO in the core 0 → core 1 direction, and one core 1 → core 0. Both are 32 bits wide and 8 words deep.

Core 0 can see the read side of the 1 → 0 FIFO (RX), and the write side of 0 → 1 FIFO (TX).

Core 1 can see the read side of the 0 → 1 FIFO (RX), and the write side of 1 → 0 FIFO (TX).

The SIO IRQ for each core is the logical OR of the VLD, WOF and ROE fields of its FIFO_ST register.

Table 36. FIFO_ST Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	ROE	Sticky flag indicating the RX FIFO was read when empty. This read was ignored by the FIFO.	WC	0x0
2	WOF	Sticky flag indicating the TX FIFO was written when full. This write was ignored by the FIFO.	WC	0x0
1	RDY	Value is 1 if this core's TX FIFO is not full (i.e. if FIFO_WR is ready for more data)	RO	0x1
0	VLD	Value is 1 if this core's RX FIFO is not empty (i.e. if FIFO_RD is valid)	RO	0x0

SIO: FIFO_WR Register**Offset:** 0x054

Table 37. FIFO_WR Register

Bits	Description	Type	Reset
31:0	Write access to this core's TX FIFO	WF	0x00000000

SIO: FIFO_RD Register**Offset:** 0x058

Table 38. FIFO_RD Register

Bits	Description	Type	Reset
31:0	Read access to this core's RX FIFO	RF	-

SIO: SPINLOCK_ST Register**Offset:** 0x05C

Table 39. SPINLOCK_ST Register

Bits	Description	Type	Reset
31:0	Spinlock state A bitmap containing the state of all 32 spinlocks (1=locked). Mainly intended for debugging.	RO	0x00000000

SIO: INTERP0_ACCUM0 Register**Offset:** 0x080

Table 40.
INTERP0_ACCUM0
Register

Bits	Description	Type	Reset
31:0	Read/write access to accumulator 0	RW	0x00000000

SIO: INTERP0_ACCUM1 Register

Offset: 0x084

Table 41.
INTERP0_ACCUM1
Register

Bits	Description	Type	Reset
31:0	Read/write access to accumulator 1	RW	0x00000000

SIO: INTERP0_BASE0 Register

Offset: 0x088

Table 42.
INTERP0_BASE0
Register

Bits	Description	Type	Reset
31:0	Read/write access to BASE0 register.	RW	0x00000000

SIO: INTERP0_BASE1 Register

Offset: 0x08c

Table 43.
INTERP0_BASE1
Register

Bits	Description	Type	Reset
31:0	Read/write access to BASE1 register.	RW	0x00000000

SIO: INTERP0_BASE2 Register

Offset: 0x090

Table 44.
INTERP0_BASE2
Register

Bits	Description	Type	Reset
31:0	Read/write access to BASE2 register.	RW	0x00000000

SIO: INTERP0_POP_LANE0 Register

Offset: 0x094

Table 45.
INTERP0_POP_LANE0
Register

Bits	Description	Type	Reset
31:0	Read LANE0 result, and simultaneously write lane results to both accumulators (POP).	RO	0x00000000

SIO: INTERP0_POP_LANE1 Register

Offset: 0x098

Table 46.
INTERP0_POP_LANE1
Register

Bits	Description	Type	Reset
31:0	Read LANE1 result, and simultaneously write lane results to both accumulators (POP).	RO	0x00000000

SIO: INTERP0_POP_FULL Register

Offset: 0x09c

Table 47.
INTERP0_POP_FULL
Register

Bits	Description	Type	Reset
31:0	Read FULL result, and simultaneously write lane results to both accumulators (POP).	RO	0x00000000

SIO: INTERP0_PEEK_LANE0 Register

Offset: 0x0a0

Table 48.
INTERP0_PEEK_LANE
0 Register

Bits	Description	Type	Reset
31:0	Read LANE0 result, without altering any internal state (PEEK).	RO	0x00000000

SIO: INTERP0_PEEK_LANE1 Register

Offset: 0x0a4

Table 49.
INTERP0_PEEK_LANE
1 Register

Bits	Description	Type	Reset
31:0	Read LANE1 result, without altering any internal state (PEEK).	RO	0x00000000

SIO: INTERP0_PEEK_FULL Register

Offset: 0x0a8

Table 50.
INTERP0_PEEK_FULL
Register

Bits	Description	Type	Reset
31:0	Read FULL result, without altering any internal state (PEEK).	RO	0x00000000

SIO: INTERP0_CTRL_LANE0 Register

Offset: 0x0ac

Description

Control register for lane 0

Table 51.
INTERP0_CTRL_LANE
0 Register

Bits	Name	Description	Type	Reset
31:26	Reserved.	-	-	-
25	OVERF	Set if either OVERF0 or OVERF1 is set.	RO	0x0
24	OVERF1	Indicates if any masked-off MSBs in ACCUM1 are set.	RO	0x0
23	OVERF0	Indicates if any masked-off MSBs in ACCUM0 are set.	RO	0x0
22	Reserved.	-	-	-
21	BLEND	Only present on INTERP0 on each core. If BLEND mode is enabled: - LANE1 result is a linear interpolation between BASE0 and BASE1, controlled by the 8 LSBs of lane 1 shift and mask value (a fractional number between 0 and 255/256ths) - LANE0 result does not have BASE0 added (yields only the 8 LSBs of lane 1 shift+mask value) - FULL result does not have lane 1 shift+mask value added (BASE2 + lane 0 shift+mask) LANE1 SIGNED flag controls whether the interpolation is signed or unsigned.	RW	0x0

Bits	Name	Description	Type	Reset
20:19	FORCE_MSB	ORed into bits 29:28 of the lane result presented to the processor on the bus. No effect on the internal 32-bit datapath. Handy for using a lane to generate sequence of pointers into flash or SRAM.	RW	0x0
18	ADD_RAW	If 1, mask + shift is bypassed for LANE0 result. This does not affect FULL result.	RW	0x0
17	CROSS_RESULT	If 1, feed the opposite lane's result into this lane's accumulator on POP.	RW	0x0
16	CROSS_INPUT	If 1, feed the opposite lane's accumulator into this lane's shift + mask hardware. Takes effect even if ADD_RAW is set (the CROSS_INPUT mux is before the shift+mask bypass)	RW	0x0
15	SIGNED	If SIGNED is set, the shifted and masked accumulator value is sign-extended to 32 bits before adding to BASE0, and LANE0 PEEK/POP appear extended to 32 bits when read by processor.	RW	0x0
14:10	MASK_MSB	The most-significant bit allowed to pass by the mask (inclusive) Setting MSB < LSB may cause chip to turn inside-out	RW	0x00
9:5	MASK_LSB	The least-significant bit allowed to pass by the mask (inclusive)	RW	0x00
4:0	SHIFT	Right-rotate applied to accumulator before masking. By appropriately configuring the masks, left and right shifts can be synthesised.	RW	0x00

SIO: INTERP0_CTRL_LANE1 Register

Offset: 0x0b0

Description

Control register for lane 1

Table 5.
INTERP0_CTRL_LANE
1 Register

Bits	Name	Description	Type	Reset
31:21	Reserved.	-	-	-
20:19	FORCE_MSB	ORed into bits 29:28 of the lane result presented to the processor on the bus. No effect on the internal 32-bit datapath. Handy for using a lane to generate sequence of pointers into flash or SRAM.	RW	0x0
18	ADD_RAW	If 1, mask + shift is bypassed for LANE1 result. This does not affect FULL result.	RW	0x0
17	CROSS_RESULT	If 1, feed the opposite lane's result into this lane's accumulator on POP.	RW	0x0
16	CROSS_INPUT	If 1, feed the opposite lane's accumulator into this lane's shift + mask hardware. Takes effect even if ADD_RAW is set (the CROSS_INPUT mux is before the shift+mask bypass)	RW	0x0

Bits	Name	Description	Type	Reset
15	SIGNED	If SIGNED is set, the shifted and masked accumulator value is sign-extended to 32 bits before adding to BASE1, and LANE1 PEEK/POP appear extended to 32 bits when read by processor.	RW	0x0
14:10	MASK_MSB	The most-significant bit allowed to pass by the mask (inclusive) Setting MSB < LSB may cause chip to turn inside-out	RW	0x00
9:5	MASK_LSB	The least-significant bit allowed to pass by the mask (inclusive)	RW	0x00
4:0	SHIFT	Right-rotate applied to accumulator before masking. By appropriately configuring the masks, left and right shifts can be synthesised.	RW	0x00

SIO: INTERP0_ACCUM0_ADD Register

Offset: 0x0b4

Table 53.
INTERP0_ACCUM0_ADD
D Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Values written here are atomically added to ACCUM0 Reading yields lane 0's raw shift and mask value (BASE0 not added).	RW	0x000000

SIO: INTERP0_ACCUM1_ADD Register

Offset: 0x0b8

Table 54.
INTERP0_ACCUM1_ADD
D Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Values written here are atomically added to ACCUM1 Reading yields lane 1's raw shift and mask value (BASE1 not added).	RW	0x000000

SIO: INTERP0_BASE_1AND0 Register

Offset: 0x0bc

Table 55.
INTERP0_BASE_1AND0
Register

Bits	Description	Type	Reset
31:0	On write, the lower 16 bits go to BASE0, upper bits to BASE1 simultaneously. Each half is sign-extended to 32 bits if that lane's SIGNED flag is set.	WO	0x00000000

SIO: INTERP1_ACCUM0 Register

Offset: 0x0c0

Table 56.
INTERP1_ACCUM0
Register

Bits	Description	Type	Reset
31:0	Read/write access to accumulator 0	RW	0x00000000

SIO: INTERP1_ACCUM1 Register

Offset: 0x0c4

Table 57.
INTERP1_ACCUM1
Register

Bits	Description	Type	Reset
31:0	Read/write access to accumulator 1	RW	0x00000000

SIO: INTERP1_BASE0 Register

Offset: 0x0c8

Table 58.
INTERP1_BASE0
Register

Bits	Description	Type	Reset
31:0	Read/write access to BASE0 register.	RW	0x00000000

SIO: INTERP1_BASE1 Register

Offset: 0x0cc

Table 59.
INTERP1_BASE1
Register

Bits	Description	Type	Reset
31:0	Read/write access to BASE1 register.	RW	0x00000000

SIO: INTERP1_BASE2 Register

Offset: 0x0d0

Table 60.
INTERP1_BASE2
Register

Bits	Description	Type	Reset
31:0	Read/write access to BASE2 register.	RW	0x00000000

SIO: INTERP1_POP_LANE0 Register

Offset: 0x0d4

Table 61.
INTERP1_POP_LANE0
Register

Bits	Description	Type	Reset
31:0	Read LANE0 result, and simultaneously write lane results to both accumulators (POP).	RO	0x00000000

SIO: INTERP1_POP_LANE1 Register

Offset: 0x0d8

Table 62.
INTERP1_POP_LANE1
Register

Bits	Description	Type	Reset
31:0	Read LANE1 result, and simultaneously write lane results to both accumulators (POP).	RO	0x00000000

SIO: INTERP1_POP_FULL Register

Offset: 0x0dc

Table 63.
INTERP1_POP_FULL
Register

Bits	Description	Type	Reset
31:0	Read FULL result, and simultaneously write lane results to both accumulators (POP).	RO	0x00000000

SIO: INTERP1_PEEK_LANE0 Register

Offset: 0x0e0

Table 64.
INTERP1_PEEK_LANE
0 Register

Bits	Description	Type	Reset
31:0	Read LANE0 result, without altering any internal state (PEEK).	RO	0x00000000

SIO: INTERP1_PEEK_LANE1 Register

Offset: 0x0e4

Table 65.
INTERP1_PEEK_LANE
1 Register

Bits	Description	Type	Reset
31:0	Read LANE1 result, without altering any internal state (PEEK).	RO	0x00000000

SIO: INTERP1_PEEK_FULL Register

Offset: 0x0e8

Table 66.
INTERP1_PEEK_FULL
Register

Bits	Description	Type	Reset
31:0	Read FULL result, without altering any internal state (PEEK).	RO	0x00000000

SIO: INTERP1_CTRL_LANE0 Register

Offset: 0x0ec

Description

Control register for lane 0

Table 67.
INTERP1_CTRL_LANE
0 Register

Bits	Name	Description	Type	Reset
31:26	Reserved.	-	-	-
25	OVERF	Set if either OVERF0 or OVERF1 is set.	RO	0x0
24	OVERF1	Indicates if any masked-off MSBs in ACCUM1 are set.	RO	0x0
23	OVERF0	Indicates if any masked-off MSBs in ACCUM0 are set.	RO	0x0
22	CLAMP	Only present on INTERP1 on each core. If CLAMP mode is enabled: - LANE0 result is shifted and masked ACCUM0, clamped by a lower bound of BASE0 and an upper bound of BASE1. - Signedness of these comparisons is determined by LANE0_CTRL_SIGNED	RW	0x0
21	Reserved.	-	-	-
20:19	FORCE_MSB	ORed into bits 29:28 of the lane result presented to the processor on the bus. No effect on the internal 32-bit datapath. Handy for using a lane to generate sequence of pointers into flash or SRAM.	RW	0x0
18	ADD_RAW	If 1, mask + shift is bypassed for LANE0 result. This does not affect FULL result.	RW	0x0
17	CROSS_RESULT	If 1, feed the opposite lane's result into this lane's accumulator on POP.	RW	0x0
16	CROSS_INPUT	If 1, feed the opposite lane's accumulator into this lane's shift + mask hardware. Takes effect even if ADD_RAW is set (the CROSS_INPUT mux is before the shift+mask bypass)	RW	0x0

Bits	Name	Description	Type	Reset
15	SIGNED	If SIGNED is set, the shifted and masked accumulator value is sign-extended to 32 bits before adding to BASE0, and LANE0 PEEK/POP appear extended to 32 bits when read by processor.	RW	0x0
14:10	MASK_MSB	The most-significant bit allowed to pass by the mask (inclusive) Setting MSB < LSB may cause chip to turn inside-out	RW	0x00
9:5	MASK_LSB	The least-significant bit allowed to pass by the mask (inclusive)	RW	0x00
4:0	SHIFT	Right-rotate applied to accumulator before masking. By appropriately configuring the masks, left and right shifts can be synthesised.	RW	0x00

SIO: INTERP1_CTRL_LANE1 Register

Offset: 0x0f0

Description

Control register for lane 1

Table 68.
INTERP1_CTRL_LANE
1 Register

Bits	Name	Description	Type	Reset
31:21	Reserved.	-	-	-
20:19	FORCE_MSB	ORed into bits 29:28 of the lane result presented to the processor on the bus. No effect on the internal 32-bit datapath. Handy for using a lane to generate sequence of pointers into flash or SRAM.	RW	0x0
18	ADD_RAW	If 1, mask + shift is bypassed for LANE1 result. This does not affect FULL result.	RW	0x0
17	CROSS_RESULT	If 1, feed the opposite lane's result into this lane's accumulator on POP.	RW	0x0
16	CROSS_INPUT	If 1, feed the opposite lane's accumulator into this lane's shift + mask hardware. Takes effect even if ADD_RAW is set (the CROSS_INPUT mux is before the shift+mask bypass)	RW	0x0
15	SIGNED	If SIGNED is set, the shifted and masked accumulator value is sign-extended to 32 bits before adding to BASE1, and LANE1 PEEK/POP appear extended to 32 bits when read by processor.	RW	0x0
14:10	MASK_MSB	The most-significant bit allowed to pass by the mask (inclusive) Setting MSB < LSB may cause chip to turn inside-out	RW	0x00
9:5	MASK_LSB	The least-significant bit allowed to pass by the mask (inclusive)	RW	0x00
4:0	SHIFT	Right-rotate applied to accumulator before masking. By appropriately configuring the masks, left and right shifts can be synthesised.	RW	0x00

SIO: INTERP1_ACCUM0_ADD Register

Offset: 0x0f4

Table 69.
INTERP1_ACCUM0_AD
D Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Values written here are atomically added to ACCUM0 Reading yields lane 0's raw shift and mask value (BASE0 not added).	RW	0x000000

SIO: INTERP1_ACCUM1_ADD Register

Offset: 0x0f8

Table 70.
INTERP1_ACCUM1_AD
D Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Values written here are atomically added to ACCUM1 Reading yields lane 1's raw shift and mask value (BASE1 not added).	RW	0x000000

SIO: INTERP1_BASE_1AND0 Register

Offset: 0x0fc

Table 71.
INTERP1_BASE_1AND
0 Register

Bits	Description	Type	Reset
31:0	On write, the lower 16 bits go to BASE0, upper bits to BASE1 simultaneously. Each half is sign-extended to 32 bits if that lane's SIGNED flag is set.	WO	0x00000000

SIO: SPINLOCK0, SPINLOCK1, ..., SPINLOCK30, SPINLOCK31 Registers

Offsets: 0x100, 0x104, ..., 0x178, 0x17c

Table 72. SPINLOCK0,
SPINLOCK1, ...,
SPINLOCK30,
SPINLOCK31
Registers

Bits	Description	Type	Reset
31:0	Reading from a spinlock address will: - Return 0 if lock is already locked - Otherwise return nonzero, and simultaneously claim the lock Writing (any value) releases the lock. If core 0 and core 1 attempt to claim the same lock simultaneously, core 0 wins. The value returned on success is 0x1 << lock number.	RW	0x00000000

SIO: DOORBELL_OUT_SET Register

Offset: 0x180

Table 73.
DOORBELL_OUT_SET
Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	<p>Trigger a doorbell interrupt on the opposite core.</p> <p>Write 1 to a bit to set the corresponding bit in DOORBELL_IN on the opposite core. This raises the opposite core's doorbell interrupt.</p> <p>Read to get the status of the doorbells currently asserted on the opposite core. This is equivalent to that core reading its own DOORBELL_IN status.</p>	RW	0x00

SIO: DOORBELL_OUT_CLR Register

Offset: 0x184

Table 74.
DOORBELL_OUT_CLR
Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	<p>Clear doorbells which have been posted to the opposite core. This register is intended for debugging and initialisation purposes.</p> <p>Writing 1 to a bit in DOORBELL_OUT_CLR clears the corresponding bit in DOORBELL_IN on the opposite core. Clearing all bits will cause that core's doorbell interrupt to deassert. Since the usual order of events is for software to send events using DOORBELL_OUT_SET, and acknowledge incoming events by writing to DOORBELL_IN_CLR, this register should be used with caution to avoid race conditions.</p> <p>Reading returns the status of the doorbells currently asserted on the other core, i.e. is equivalent to that core reading its own DOORBELL_IN status.</p>	WC	0x00

SIO: DOORBELL_IN_SET Register

Offset: 0x188

Table 75.
DOORBELL_IN_SET
Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	Write 1s to trigger doorbell interrupts on this core. Read to get status of doorbells currently asserted on this core.	RW	0x00

SIO: DOORBELL_IN_CLR Register

Offset: 0x18c

Table 76.
DOORBELL_IN_CLR
Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	<p>Check and acknowledge doorbells posted to this core. This core's doorbell interrupt is asserted when any bit in this register is 1.</p> <p>Write 1 to each bit to clear that bit. The doorbell interrupt deasserts once all bits are cleared. Read to get status of doorbells currently asserted on this core.</p>	WC	0x00

SIO: PERI_NONSEC Register

Offset: 0x190

Description

Detach certain core-local peripherals from Secure SIO, and attach them to Non-secure SIO, so that Non-secure software can use them. Attempting to access one of these peripherals from the Secure SIO when it is attached to the Non-secure SIO, or vice versa, will generate a bus error.

This register is per-core, and is only present on the Secure SIO.

Most SIO hardware is duplicated across the Secure and Non-secure SIO, so is not listed in this register.

Table 77.
PERI_NONSEC
Register

Bits	Name	Description	Type	Reset
31:6	Reserved.	-	-	-
5	TMDS	If 1, detach TMDS encoder (of this core) from the Secure SIO, and attach to the Non-secure SIO.	RW	0x0
4:2	Reserved.	-	-	-
1	INTERP1	If 1, detach interpolator 1 (of this core) from the Secure SIO, and attach to the Non-secure SIO.	RW	0x0
0	INTERP0	If 1, detach interpolator 0 (of this core) from the Secure SIO, and attach to the Non-secure SIO.	RW	0x0

SIO: RISCV_SOFTIRQ Register

Offset: 0x1a0

Description

Control the assertion of the standard software interrupt (MIP.MSIP) on the RISC-V cores.

Unlike the RISC-V timer, this interrupt is not routed to a normal system-level interrupt line, so can not be used by the Arm cores.

It is safe for both cores to write to this register on the same cycle. The set/clear effect is accumulated across both cores, and then applied. If a flag is both set and cleared on the same cycle, only the set takes effect.

Table 78.
RISCV_SOFTIRQ
Register

Bits	Name	Description	Type	Reset
31:10	Reserved.	-	-	-
9	CORE1_CLR	Write 1 to atomically clear the core 1 software interrupt flag. Read to get the status of this flag.	RW	0x0
8	CORE0_CLR	Write 1 to atomically clear the core 0 software interrupt flag. Read to get the status of this flag.	RW	0x0
7:2	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
1	CORE1_SET	Write 1 to atomically set the core 1 software interrupt flag. Read to get the status of this flag.	RW	0x0
0	CORE0_SET	Write 1 to atomically set the core 0 software interrupt flag. Read to get the status of this flag.	RW	0x0

SIO: MTIME_CTRL Register

Offset: 0x1a4

Description

Control register for the RISC-V 64-bit Machine-mode timer. This timer is only present in the Secure SIO, so is only accessible to an Arm core in Secure mode or a RISC-V core in Machine mode.

Note whilst this timer follows the RISC-V privileged specification, it is equally usable by the Arm cores. The interrupts are routed to normal system-level interrupt lines as well as to the MIP.MTIP inputs on the RISC-V cores.

Table 79.
MTIME_CTRL Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	DBGPAUSE_CORE 1	If 1, the timer pauses when core 1 is in the debug halt state.	RW	0x1
2	DBGPAUSE_CORE 0	If 1, the timer pauses when core 0 is in the debug halt state.	RW	0x1
1	FULLSPEED	If 1, increment the timer every cycle (i.e. run directly from the system clock), rather than incrementing on the system-level timer tick input.	RW	0x0
0	EN	Timer enable bit. When 0, the timer will not increment automatically.	RW	0x1

SIO: MTIME Register

Offset: 0x1b0

Table 80. MTIME Register

Bits	Description	Type	Reset
31:0	Read/write access to the high half of RISC-V Machine-mode timer. This register is shared between both cores. If both cores write on the same cycle, core 1 takes precedence.	RW	0x00000000

SIO: MTIMEH Register

Offset: 0x1b4

Table 81. MTIMEH Register

Bits	Description	Type	Reset
31:0	Read/write access to the high half of RISC-V Machine-mode timer. This register is shared between both cores. If both cores write on the same cycle, core 1 takes precedence.	RW	0x00000000

SIO: MTIMECMP Register

Offset: 0x1b8

Table 82. MTIMECMP Register

Bits	Description	Type	Reset
31:0	<p>Low half of RISC-V Machine-mode timer comparator. This register is core-local, i.e., each core gets a copy of this register, with the comparison result routed to its own interrupt line.</p> <p>The timer interrupt is asserted whenever MTIME is greater than or equal to MTIMECMP. This comparison is unsigned, and performed on the full 64-bit values.</p>	RW	0xffffffff

SIO: MTIMECMPH Register

Offset: 0x1bc

Table 83.
MTIMECMPH Register

Bits	Description	Type	Reset
31:0	<p>High half of RISC-V Machine-mode timer comparator. This register is core-local.</p> <p>The timer interrupt is asserted whenever MTIME is greater than or equal to MTIMECMP. This comparison is unsigned, and performed on the full 64-bit values.</p>	RW	0xffffffff

SIO: TMDS_CTRL Register

Offset: 0x1c0

Description

Control register for TMDS encoder.

Table 84. TMDS_CTRL Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	CLEAR_BALANCE	Clear the running DC balance state of the TMDS encoders. This bit should be written once at the beginning of each scanline.	SC	0x0
27	PIX2_NOSHIFT	<p>When encoding two pixels's worth of symbols in one cycle (a read of a PEEK/POP_DOUBLE register), the second encoder sees a shifted version of the colour data register.</p> <p>This control disables that shift, so that both encoder layers see the same pixel data. This is used for pixel doubling.</p>	RW	0x0

Bits	Name	Description	Type	Reset
26:24	PIX_SHIFT	<p>Shift applied to the colour data register with each read of a POP alias register.</p> <p>Reading from the POP_SINGLE register, or reading from the POP_DOUBLE register with PIX2_NOSHIFT set (for pixel doubling), shifts by the indicated amount.</p> <p>Reading from a POP_DOUBLE register when PIX2_NOSHIFT is clear will shift by double the indicated amount. (Shift by 32 means no shift.)</p> <p>0x0 → Do not shift the colour data register. 0x1 → Shift the colour data register by 1 bit 0x2 → Shift the colour data register by 2 bits 0x3 → Shift the colour data register by 4 bits 0x4 → Shift the colour data register by 8 bits 0x5 → Shift the colour data register by 16 bits</p>	RW	0x0
23	INTERLEAVE	<p>Enable lane interleaving for reads of PEEK_SINGLE/POP_SINGLE.</p> <p>When interleaving is disabled, each of the 3 symbols appears as a contiguous 10-bit field, with lane 0 being the least-significant and starting at bit 0 of the register.</p> <p>When interleaving is enabled, the symbols are packed into 5 chunks of 3 lanes times 2 bits (30 bits total). Each chunk contains two bits of a TMDS symbol per lane, with lane 0 being the least significant.</p>	RW	0x0
22:21	Reserved.	-	-	-
20:18	L2_NBITS	Number of valid colour MSBs for lane 2 (1-8 bits, encoded as 0 through 7). Remaining LSBs are masked to 0 after the rotate.	RW	0x0
17:15	L1_NBITS	Number of valid colour MSBs for lane 1 (1-8 bits, encoded as 0 through 7). Remaining LSBs are masked to 0 after the rotate.	RW	0x0
14:12	L0_NBITS	Number of valid colour MSBs for lane 0 (1-8 bits, encoded as 0 through 7). Remaining LSBs are masked to 0 after the rotate.	RW	0x0
11:8	L2_ROT	<p>Right-rotate the 16 LSBs of the colour accumulator by 0-15 bits, in order to get the MSB of the lane 2 (red) colour data aligned with the MSB of the 8-bit encoder input.</p> <p>For example, for RGB565 (red most significant), red is bits 15:11, so should be right-rotated by 8 bits to align with bits 7:3 of the encoder input.</p>	RW	0x0

Bits	Name	Description	Type	Reset
7:4	L1_ROT	<p>Right-rotate the 16 LSBs of the colour accumulator by 0-15 bits, in order to get the MSB of the lane 1 (green) colour data aligned with the MSB of the 8-bit encoder input.</p> <p>For example, for RGB565, green is bits 10:5, so should be right-rotated by 3 bits to align with bits 7:2 of the encoder input.</p>	RW	0x0
3:0	L0_ROT	<p>Right-rotate the 16 LSBs of the colour accumulator by 0-15 bits, in order to get the MSB of the lane 0 (blue) colour data aligned with the MSB of the 8-bit encoder input.</p> <p>For example, for RGB565 (red most significant), blue is bits 4:0, so should be right-rotated by 13 to align with bits 7:3 of the encoder input.</p>	RW	0x0

SIO: TMDS_WDATA Register

Offset: 0x1c4

Table 85.
TMDS_WDATA
Register

Bits	Description	Type	Reset
31:0	Write-only access to the TMDS colour data register.	WO	0x00000000

SIO: TMDS_PEEK_SINGLE Register

Offset: 0x1c8

Table 86.
TMDS_PEEK_SINGLE
Register

Bits	Description	Type	Reset
31:0	<p>Get the encoding of one pixel's worth of colour data, packed into a 32-bit value (3x10-bit symbols).</p> <p>The PEEK alias does not shift the colour register when read, but still advances the running DC balance state of each encoder. This is useful for pixel doubling.</p>	RF	0x00000000

SIO: TMDS_POP_SINGLE Register

Offset: 0x1cc

Table 87.
TMDS_POP_SINGLE
Register

Bits	Description	Type	Reset
31:0	<p>Get the encoding of one pixel's worth of colour data, packed into a 32-bit value. The packing is 5 chunks of 3 lanes times 2 bits (30 bits total). Each chunk contains two bits of a TMDS symbol per lane. This format is intended for shifting out with the HSTX peripheral on RP2350.</p> <p>The POP alias shifts the colour register when read, as well as advancing the running DC balance state of each encoder.</p>	RF	0x00000000

SIO: TMDS_PEEK_DOUBLE_L0 Register

Offset: 0x1d0

Table 88.
TMDS_PEEK_DOUBLE_L0 Register

Bits	Description	Type	Reset
31:0	<p>Get lane 0 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The PEEK alias does not shift the colour register when read, but still advances the lane 0 DC balance state. This is useful if all 3 lanes' worth of encode are to be read at once, rather than processing the entire scanline for one lane before moving to the next lane.</p>	RF	0x00000000

SIO: TMDS_PEEK_DOUBLE_L0 Register

Offset: 0x1d4

Table 89.
TMDS_POP_DOUBLE_L0 Register

Bits	Description	Type	Reset
31:0	<p>Get lane 0 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The POP alias shifts the colour register when read, according to the values of PIX_SHIFT and PIX2_NOSHIFT.</p>	RF	0x00000000

SIO: TMDS_PEEK_DOUBLE_L1 Register

Offset: 0x1d8

Table 90.
TMDS_PEEK_DOUBLE_L1 Register

Bits	Description	Type	Reset
31:0	<p>Get lane 1 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The PEEK alias does not shift the colour register when read, but still advances the lane 1 DC balance state. This is useful if all 3 lanes' worth of encode are to be read at once, rather than processing the entire scanline for one lane before moving to the next lane.</p>	RF	0x00000000

SIO: TMDS_POP_DOUBLE_L1 Register

Offset: 0x1dc

Table 91.
TMDS_POP_DOUBLE_L1 Register

Bits	Description	Type	Reset
31:0	<p>Get lane 1 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The POP alias shifts the colour register when read, according to the values of PIX_SHIFT and PIX2_NOSHIFT.</p>	RF	0x00000000

SIO: TMDS_PEEK_DOUBLE_L2 Register

Offset: 0x1e0

Table 92.
TMDS_PEEK_DOUBLE_L2 Register

Bits	Description	Type	Reset
31:0	<p>Get lane 2 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The PEEK alias does not shift the colour register when read, but still advances the lane 2 DC balance state. This is useful if all 3 lanes' worth of encode are to be read at once, rather than processing the entire scanline for one lane before moving to the next lane.</p>	RF	0x00000000

SIO: TMDS_POP_DOUBLE_L2 Register

Offset: 0x1e4

Table 93.
TMDS_POP_DOUBLE_L2 Register

Bits	Description	Type	Reset
31:0	<p>Get lane 2 of the encoding of two pixels' worth of colour data. Two 10-bit TMDS symbols are packed at the bottom of a 32-bit word.</p> <p>The POP alias shifts the colour register when read, according to the values of PIX_SHIFT and PIX2_NOSHIFT.</p>	RF	0x00000000

3.2. Interrupts

Each core is equipped with an internal interrupt controller, with 52 interrupt inputs. For the most part each core has exactly the same interrupts routed to it, though there are some exceptions, referred to as *core-local interrupts*, where there is an individual per-core interrupt source mapped to the same interrupt number on each core:

- Cross-core FIFO interrupts: [SIO_IRQ_FIFO](#) and [SIO_IRQ_FIFO_NS](#) (Section 3.1.5)
- Cross-core doorbell interrupts: [SIO_IRQ_BELL](#) and [SIO_IRQ_BELL_NS](#) (Section 3.1.6)
- RISC-V platform timer (also usable by Arm cores): [SIO_IRQ_MTIMECMP](#) (Section 3.1.8)
- GPIO interrupts: [IO_IRQ_BANK0](#), [IRQ_IO_BANK0_NS](#), [IO_IRQ_QSPI](#), [IO_IRQ_QSPI_NS](#) (Section 9.5)

The remaining interrupt inputs have the same interrupt source mirrored identically on both cores. Non-core-local interrupts should only be enabled in the interrupt controller of a single core at a time, and will be serviced by the core whose interrupt controller they are enabled in.

Table 94. Interrupts

IRQ	Interrupt Source	IRQ	Interrupt Source	IRQ	Interrupt Source	IRQ	Interrupt Source	IRQ	Interrupt Source
0	TIMER0_IRQ_0	11	DMA_IRQ_1	22	IO_IRQ_BANK0_NS	33	UART0_IRQ	44	POWMAN_IRQ_POW
1	TIMER0_IRQ_1	12	DMA_IRQ_2	23	IO_IRQ_QSPI	34	UART1_IRQ	45	POWMAN_IRQ_TIMER
2	TIMER0_IRQ_2	13	DMA_IRQ_3	24	IO_IRQ_QSPI_NS	35	ADC_IRQ_FIFO	46	SPAREIRQ_IRQ_0
3	TIMER0_IRQ_3	14	USBCTRL_IRQ	25	SIO_IRQ_FIFO	36	I2C0_IRQ	47	SPAREIRQ_IRQ_1
4	TIMER1_IRQ_0	15	PIO0_IRQ_0	26	SIO_IRQ_BELL	37	I2C1_IRQ	48	SPAREIRQ_IRQ_2
5	TIMER1_IRQ_1	16	PIO0_IRQ_1	27	SIO_IRQ_FIFO_NS	38	OTP_IRQ	49	SPAREIRQ_IRQ_3
6	TIMER1_IRQ_2	17	PIO1_IRQ_0	28	SIO_IRQ_BELL_NS	39	TRNG_IRQ	50	SPAREIRQ_IRQ_4
7	TIMER1_IRQ_3	18	PIO1_IRQ_1	29	SIO_IRQ_MTIMECMP	40	PROC0_IRQ_CTI	51	SPAREIRQ_IRQ_5
8	PWM_IRQ_WRAP_0	19	PIO2_IRQ_0	30	CLOCKS_IRQ	41	PROC1_IRQ_CTI		
9	PWM_IRQ_WRAP_1	20	PIO2_IRQ_1	31	SPI0_IRQ	42	PLL_SYS_IRQ		
10	DMA_IRQ_0	21	IO_IRQ_BANK0	32	SPI1_IRQ	43	PLL_USB_IRQ		

On RP2350, only the lower 46 IRQ signals are connected to system-level interrupt sources, and IRQs 46 to 51 are hardwired to zero (never firing). These six spare interrupts, referred to as `SPAREIRQ IRQ_0` through `SPAREIRQ IRQ_5` in the table, are deliberately reserved for the cores to interrupt themselves (via the Arm `NVIC_ISPRx` registers or the Hazard3 `meifa` CSR), for example, when an interrupt handler wants to schedule a "bottom half" handler for work that must be done after exiting the interrupt handler, but before returning to the code running in the foreground.

Nested interrupts are supported in hardware: a lower-priority interrupt can be pre-empted by a higher-priority interrupt or fault, and will resume once the higher-priority handler returns. The pre-emption priority order is determined by the `NVIC_IPRx` registers (Cortex-M33) or the `MEIPRA` interrupt priority array CSR (Hazard3).

When there is a choice of multiple interrupts to be entered at the same dynamic priority, the interrupt with the lowest IRQ number is chosen as a tie-breaker. The system-level IRQ numbering has been chosen to generally put higher-priority interrupts at lower IRQ numbers for this reason, though the true priority is often dependent on the specific application.

3.2.1. Non-maskable Interrupt (NMI)

The system IRQ signals can be routed to the Cortex-M33 non-maskable interrupt (NMI) input, by setting the bit for that IRQ number in `NMI_MASK0` or `NMI_MASK1`. The non-maskable interrupt ignores the processor's interrupt enable/disable state (PRIMASK), and can pre-empt any other active interrupt. NMIs are generally used for emergent circumstances that require the processor's unconditional attention, such as loss of PLL lock or power supply integrity.

The NMI mask registers are core-local, so each core can have a different combination of interrupts routed to its NMI input. The NMI mask, along with all other EPPB registers, is reset by a warm reset of that core. This avoids an issue on RP2040 where the NMI mask could be left set following a processor reset.

In addition to system-level interrupts, the non-maskable interrupt is asserted when an integrity check is failed in the redundancy coprocessor (RCP, [Section 3.6.3](#)). This behaviour cannot be disabled, but a correctly-programmed RCP does not trigger under normal voltage, frequency, and temperature conditions. Likewise, if user code does not execute any RCP instructions, the RCP will never trigger. The RCP NMI output is asserted on both cores when an integrity check fails, and is deasserted by a warm processor reset.

3.3. Event Signals (Arm)

Using the `WFE` instruction, the Cortex-M33 can enter a sleep state until an "event" (or interrupt) takes place. It can also generate events using the `SEV` instruction. RP2350 cross-wires event signals between the two processors: an event sent by one processor will be received on the other.

NOTE

The event flag is "sticky": if both processors send an event (`SEV`) simultaneously, then enter the sleep state (`WFE`), they will both wake immediately. This prevents the processors from getting stuck in a sleep state in this scenario.

Processors also receive an event signal from the global monitor if their reservation is lost due to a write by a different master, in accordance with Armv8-M architecture requirements.

While in a `WFE` (or `WFI`) sleep state, the processor shuts off its internal clock gates to reduce power consumption. When **both** processors are in a sleep state and the DMA is inactive, all of RP2350 can enter a sleep state, disabling clocks on unused infrastructure such as the bus fabric. The rest of RP2350 wakes automatically when either of the processors wakes. See [Section 6.3.4](#).

3.4. Event Signals (RISC-V)

The Hazard3 `h3.block` instruction halts processor execution until an unblock signal is received. The `h3.unblock` instruction sends an unblock signal to other processors. These NOP-compatible hint instructions are documented in [Section](#)

3.8.4.3.

On RP2350 the Hazard3 unblock in/out signals are cross-connected between the two processors, and each processor's unblock output is also fed back into its input. The global monitor also posts an unblock signal to each core when that core loses a reservation due to an access by another core or the system DMA.

The Hazard3 **MSLEEP** CSR defines how deep a sleep the processor will enter when executing a **h3.block** instruction. By default this is a simple pipeline stall, but the processor can also gate its own clock and negotiate the system-level clock wake/sleep state with the clocks block ([Section 6.3.4](#)).

The **h3.unblock** instruction is "sticky": an **h3.block** will fall through immediately if any unblock signal has been received since the last time the processor executed an **h3.block** instruction.

3.5. Debug

The 2-wire Serial Wire Debug (SWD) port provides access to hardware and software debug features including:

- Loading firmware into SRAM or external flash memory
- Control of processor execution: run/halt, step, set breakpoints, other standard debug functionality
- Access to processor architectural state
- Access to memory and memory-mapped IO via the system bus
- Configuring the CoreSight trace hardware

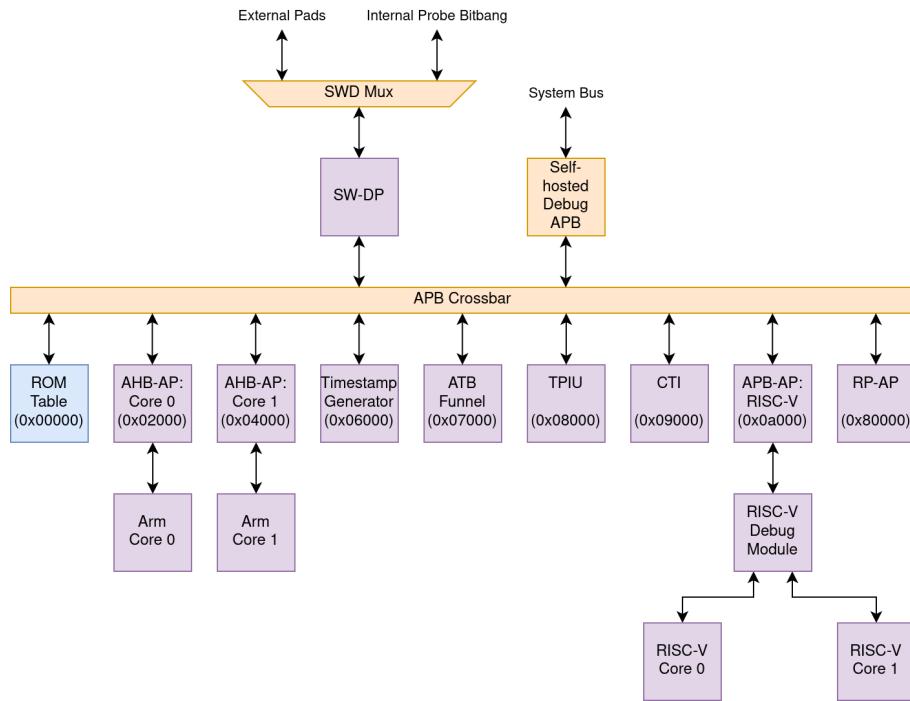
The SWD bus is exposed on two dedicated pins, SWCLK and SWDIO, and is immediately available after power-on.

NOTE

We recommend a max SWD frequency of 24MHz. This depends heavily on your setup. You may need to run much slower (1MHz) depending on the quality and length of your cables.

A single SW-DP provides access to RP2350's debug subsystem from the external SWD pins. The DP is multidrop-capable, but use of multidrop SWD is not mandatory. All hardware in the debug subsystem, with the exception of the RP-AP, can also be accessed directly from the system bus using the self-hosted debug window starting at **CORESIGHT_PERIPH_BASE**.

Figure 9. RP2350 debug topology. An SW-DP connects the external SWD pins to internal debug hardware. The ROM table lists debug components, for automatic discovery. AHB-APs provide debug access to Arm processors, and an APB-AP provides access to a standard RISC-V Debug Module. The RP-AP provides Raspberry Pi-specific controls such as rescue reset and debug key entry. Remaining components are for Arm trace.



The numbers in brackets in Figure 9 are the addresses of the debug components within the debug address space. These correspond to values written to the SW-DP SELECT register for SWD accesses, or offsets from `CORESIGHT_PERIPH_BASE` for self-hosted debug access. All APs are accessible through the SW-DP, and all except the RP-AP are also accessible through self-hosted debug.

3.5.1. Connecting to the SW-DP

The SW-DP defaults to the Dormant state at power-up or assertion of the external reset (RUN) pin. A Dormant-to-SWD sequence must be issued before beginning SWD operations. See the Arm Debug Interface specification, version 6, for details of Dormant/SWD state switching: <https://developer.arm.com/documentation/ih0074/latest/>

After a power-on, the following sequence can be used to connect to the SW-DP:

- At least 8 × SWCLK cycles with SWDIO high.
- The 128-bit Selection Alert sequence: `0x19BC0EA2, 0xE3DDA9E9, 0x86852D95, 0x6209F392`, LSB-first.
- Four SWCLK cycles with SWDIO low.
- SWD activation code sequence : `0x1A`, LSB first.
- At least 50 × SWCLK cycles with SWDIO high (line reset).
- A DPIDR read to exit the Reset state

In order to wake up the system from a low power (P1.x) state, set the CDBGWRUPREQ in the DP CTRL/STAT register, then poll CDBGWRUPACK in the same register until set. In low-power states, only the SW-DP and RP-AP are accessible, as the remaining debug logic is unpowered.

3.5.2. Arm Debug

There are two AHB5 Mem-APs, at offsets `0x02000` and `0x04000` in the debug address space, which are used to debug the two Arm Cortex-M33 processors. Each Mem-AP is a bus master which accesses a 32-bit address space. This is the same address space accessed by a processor's load/store instructions, which includes system-level hardware such as memory and peripherals, and processor-internal hardware on the processor's private peripheral bus (PPB). Certain PPB registers are visible only when accessed from the Mem-AP, not when accessed by software running on the processor.

The AHB5 Mem-AP's own register map is defined in Arm's ADIV6 specification. Generally this is only of interest to those implementing their own debug translator, and the Mem-AP can be thought of simply as a bridge between a DP (such as RP2350's SW-DP) and a downstream address space.

The standard Arm debug registers used to debug software running on the Cortex-M33 can be found documented in the [Armv8-M Architecture Reference Manual](#), or the Cortex-M33 Technical Reference Manual, available from Arm Ltd. This datasheet also documents the core's internal registers in [Section 3.7.5](#).

The Mem-APs can access system peripherals and memory at exactly the same addresses they would be accessed by software running on the processor. However, the privilege and security of Mem-AP accesses may be different from the security state of the software running on the processor at the point it halted: the privilege and security of Mem-AP accesses is configured explicitly via its control and status word (CSW) register. Care must be taken when debugging Non-secure software which accesses the SIO, for example, because by default the debugger may access the Secure alias of the SIO, not the Non-secure alias which software will have been accessing.

The bus filters configured by the ACCESSCTRL bus access permission registers ([Section 10.5.2](#)) treat bus accesses originating from the Mem-APs as distinct from bus accesses originating from software running on the processor. This means it is possible to lock software out from a peripheral, whilst still allowing debugger access.

3.5.3. RISC-V Debug

There is a single APB Mem-AP, at offset `0x0a000` in the debug address space, which provides access only to the RISC-V Debug Module (DM). The DM is a standard component which the debugger uses to enumerate RISC-V harts present in the system, debug software running on each hart, and access the system bus. It is defined in the RISC-V debug specification, of which RP2350 implements version 0.13.2.

From the point of view of the RISC-V debug specification, the SW-DP and APB Mem-AP function jointly as the Debug Transport Module for this system. The DM is located at offset `0x0` in the APB-AP's downstream address space, and the registers are word-sized and byte-addressed, meaning the DM register addresses in the debug specification must be multiplied by 4 to get the correct APB address.

On RP2350, each core possesses exactly one hardware thread (hart). Core 0 has a hart ID of 0, and core 1 has a hart ID of 1. These hart IDs match the hart index used in the DM. This DM is also equipped with the hart array mask select extension, which allows multiple cores to be reset/halted/resumed simultaneously.

The DM is equipped with the System Bus Access (SBA) extension, which allows the debugger to access the system bus without halting either core. This can be used for minimally intrusive debug techniques like Segger RTT. SBA accesses arbitrate with core 1's load/store port to access the system bus, but they are treated as distinct from core 1's accesses for the purpose of bus filtering ([Section 10.5.2](#)), which means it is possible to lock software out of a peripheral whilst retaining debug access. Processor load/stores in Debug mode are also treated as debug accesses for the purpose of bus filtering.

The DM is able to reset each core individually using the `dmcontrol.hartreset` control. This resets *only* the selected processor. The `dmcontrol.ndmreset` resets both processors *only*, which is the minimum requirement in the RISC-V debug specification. A full system reset, which includes the DM, can be performed using the SYSRESETREQ control in the SW-DP, a switched core domain reset configured in POWMAN and initiated by the watchdog, or any full-system reset such as the RUN pin. A PSM reset initiated by the watchdog can reset almost all system-level hardware except for the DM, but note that the DM becomes momentarily inaccessible whilst the system clock's clock generator is reset, which is the reason for `dmcontrol.ndmreset` resetting the processors only.

For details on the processor side of RISC-V debug, see [Section 3.8.3](#). See also the Hazard3 source code at <https://github.com/Wren6991/Hazard3/>, which includes the DM implementation.

3.5.4. Debug Power Domains

The SW-DP and the RP-AP are in the always-on power domain. This means they are available even when the system is in its lowest-power state, with the switched core domain (which includes the processors) fully powered down.

The remainder of the debug hardware is in the switched core domain. This is the same domain as the processors and system peripherals.

Setting the CDBGWRUPREQ bit in the SW-DP's CTRL/STAT register will force a power up of the switched core domain, making the remaining debug hardware available. This power up takes some time, as it is sequenced by the 32 kHz low-power oscillator (Section 8.4), so the CDBGWRUPACK bit must be polled to wait for the system to power up before attempting to access any APs other than the RP-AP. See Arm's ADIV6 specification for the SW-DP's register listing.

Note that the RP-AP is accessible without asserting CDBGWRUPACK.

3.5.5. Software control of SWD pins

The **DBGFORCE** register in SYSCFG can be used to detach the SW-DP from the external debug pads, and instead bitbang the SWD signals directly from software. This is intended for a debug probe running on one core being used to debug the other core. For other use cases it is generally cleaner to use the self-hosted debug access to interface with the APs directly from the system bus.

3.5.6. Self-hosted Debug

All APs shown in Figure 9, except for the RP-AP, have direct memory-mapped access from the system bus. This is known as self-hosted debug, because with care it allows running a debug host (i.e. a debugger) directly on-system. It can also be used to access the trace hardware, which can be used for self-hosted trace using the trace DMA FIFO. By default only Secure access is permitted, as the processor debug presents an opportunity for Non-secure code to interfere with the Secure context and/or perform Secure bus accesses.

The self-hosted debug window starts at address [0x40140000 \(CORESIGHT_PERIPH_BASE\)](#). The offsets of the APs within this window are the same as the APs' addresses when accessed from the SW-DP.

Because of the blocking nature of the AHB-AP's DRW register, and its interactions with the Cortex-M33's arbitration of AHB-AP accesses with load/stores, certain accesses have potential to cause bus lockup due to circular bus stall dependencies. In particular, cores may not access their own AHB-APs through the self-hosted debug window, and AHB-APs may not access AHB-APs through the self-hosted debug window – attempting to do so will immediately return a bus fault. To reduce the opportunities for deadlock, a full APB crossbar is used to connect the SW-DP and the self-hosted debug port to the APs, so that for example self-hosted use of the Arm trace hardware will not interfere with an external debugger attaching via the AHB-APs.

There are some cases where a bus deadlock can not be avoided, such as a core using the other core's AHB-AP, via the self-hosted debug window, to access some other APB peripheral:

1. The access upstream of the APB's DRW register will not complete until the downstream access completes
2. The downstream access will not complete until it is granted access to the system APB bridge
3. Access to the APB bridge will not be granted until the upstream access, which is occupying the system APB bridge, completes
4. See point 1.

This situation can arise when running a self-hosted debugger on one core, and debugging code on the other core which accesses APB addresses. The deadlock is eventually broken when the APB bridge's 65536-cycle timeout expires, abandoning the transfer and returning a bus error to the origin of the upstream access. To avoid this, software should detect when it is about to use an AP to access an APB address (an address starting with [0x4](#)), and perform the access directly instead of using the Mem-AP.

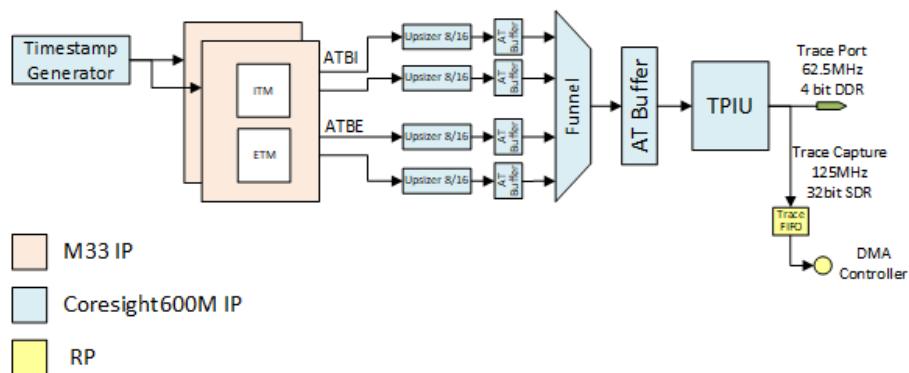
This type of deadlock does not occur when the debugger accesses the bus with RISC-V System Bus Access, because the bus transfer upstream of the DM does not block on completion of the downstream access.

3.5.7. Trace

3.5.7.1. Overview

The ATB trace subsystem is based on the Coresight SoC-600M architecture, as shown in [Figure 10](#).

Figure 10. Trace Subsystem



The trace subsystem captures trace messages from each of the M33 ITM/ETM components, merges them into a single trace bus, and sends off-chip through the 4-bit DDR trace port for subsequent capture and analysis by a trace port analyser.

The trace subsystem comprises the following main components:

- M33 ETM - Embedded Trace Macroel, real-time instruction flow messages
- M33 ITM - Instruction Trace Macrocell, software generated messages
- Timestamp Generator - timestamp fed to the M33 processors to align trace information.
- ATB Funnel - merges the M33 trace sources into a single trace stream.
- TPIU - Trace Port Interface Unit, outputs trace data over trace port pins. The source-synchronous trace interface is 4-bits DDR, 62.5MHz clock, giving a maximum trace data rate of 500Mb/s.
- Trace FIFO - an RP component that optionally captures the 32-bit trace stream for sending to the system SRAM by the DMA Controller. Bus access to the trace FIFO is controlled by the **CORESIGHT_TRACE** register in ACCESSCTRL.

3.5.8. Rescue Reset

A rescue reset is a full system reset, similar to asserting the RUN pin low, which also sets a flag telling the bootrom to halt before running any user software. This is performed over the SWD bus using the RP-AP, and can be performed even when system clocks are stopped and the switched core power domain is powered down. This is used in the case where the chip has locked up, for example if code has been programmed into flash which permanently halts the system clock: since the debugger can no longer communicate with the processors to return the system to a working state, more drastic action is needed. This functionality was provided by the Rescue DP on RP2040, but on RP2350 it is provided by the RP-AP, to avoid mandatory use of multidrop SWD.

A rescue is invoked by setting and then clearing the **CTRL.RESCUE_RESTART** bit in the RP-AP. This causes a hard reset of the chip, and sets **CHIP_RESET.RESCUE_FLAG** to indicate that a rescue reset took place. The bootrom checks this flag almost immediately in the initial boot process (before watchdog, flash or USB boot), acknowledges by clearing the bit, then halts the processor. This leaves the system in a safe state, with the system clock running, so that the debugger can reattach to the cores and load fresh code.

3.5.9. Security

By default, the SWD debug access port allows an external debugger to access all system memory and peripherals, and

to observe and change the execution of software running on the Cortex-M33 processors. If boot signature enforcement is enabled ([Section 10.1.1](#)), debug access becomes a security concern, as it is able to sidestep this protection. To account for this, RP2350 supports progressively locking down the debug port using configuration in on-chip OTP storage.

Conceptually there are two control bits: debug disable, and secure debug disable. Debug disable is intended to completely cut off debug access to the processors and the system bus, whilst the secure debug disable forbids Secure bus accesses, and halting of processors in the Secure state, but still allows Non-secure software to be debugged as normal. There are two ways to set these control bits:

- Setting the relevant OTP critical flag: `CRIT1.DEBUG_DISABLE` or `CRIT1.SECURE_DEBUG_DISABLE` to set the debug disable or secure debug disable, respectively
- Installing a 128-bit fixed debug key as OTP key 5 or 6 ([Section 3.5.9.2](#))

OTP configuration changes take effect at the next reset of the OTP block.

Once debug has been disabled, software can re-enable debug using the OTP `DEBUGEN` register, which allows the secure and overall debug enable to be cleared individually for each processor. For example, Secure software may implement a shell where users can authenticate using a cryptographic challenge to enable debug on systems where it is disabled by default. The `DEBUGEN` register belongs to the processor cold reset domain, so it is preserved over a PSM reset starting from as early as OTP (the second PSM stage). This allows almost a full system reset without losing debug access.

To avoid accidental writes of the `DEBUGEN` register, its bits can be individually locked using the matching bits in `DEBUGEN_LOCK`.

This offers increasing levels of debug protection:

1. Fully open: no keys installed and no OTP debug disable flags are set. This is the most convenient configuration for product development.
2. Access with key only: at least one key is installed, but no OTP debug disable flags are set.
3. No access even with key (an OTP debug disable flag is set), but Secure code can enable debug access by writing to `DEBUGEN`.
4. No access even with key (an OTP debug disable flag is set), and `DEBUGEN` is locked by `DEBUGEN_LOCK`.

3.5.9.1. Effects of Debug Disables

The secure debug disable flag (`CRIT1.SECURE_DEBUG_DISABLE`) has the following effects:

- Set Secure AP enable signals for Arm core 0 and core 1 AHB-APs to 0.
 - This prevents the APs from performing Secure bus accesses (including to the PPB).
 - Status is reported in the `SDeviceEn` flag of the AHB-AP `CSW` register.
- Set the Cortex-M33 `SPIDEN` and `SPNIDEN` signals for both cores to 0.
 - This prevents the cores from being halted or traced whilst in the Secure state.
- Disable the factory test JTAG interface ([Section 10.8](#)).

The debug disable flag (`CRIT1.DEBUG_DISABLE`) has all of the effects of the secure debug disable flag. It also has the following additional effects:

- Set AP enable signals for Arm core 0 and core 1 AHB-APs to 0.
 - This prevents the APs from performing any bus accesses at all (including to the PPB).
 - Status is reported in the `DeviceEn` flag of the AHB-AP `CSW` register.
- Set AP enable signal for RISC-V DM APB-AP to 0.
 - This prevents the AP from accessing the RISC-V Debug Module.

- Status is reported in the [DeviceEn](#) flag of the APB-AP [CSW](#) register.

- Set [DBGEN](#) and [NIDEN](#) signals for the CTI to 0.

On RISC-V [CRIT1.SECURE_DEBUG_DISABLE](#) has no useful effect. Debug-mode accesses from the cores always have Secure and Privileged bus attributes, except when reduced by [FORCE_CORE_NS](#). Likewise, System Bus Access via the Debug Module is always Secure and Privileged, unless [FORCE_CORE_NS.CORE1](#) is set, in which case it is Non-secure and Privileged. Use the [CRIT1.DEBUG_DISABLE](#) flag on RISC-V.

3.5.9.2. Debug Keys

[Section 13.5.2](#) describes the OTP hardware access keys. Hardware reads OTP access keys into hidden registers as part of the OTP power-up sequence which takes place after an OTP reset, and the corresponding OTP locations then become inaccessible. OTP keys 5 and 6 are special in that they control access to the SWD debug hardware in addition to functioning as normal OTP page keys.

A debug key is a 128-bit fixed challenge. Installing a debug key in OTP locks down debug access, and it remains locked until the debug host writes a matching key value through the RP-AP [DBGKEY](#) register. This is a write-only interface.

To install a debug key, first program the OTP locations starting from [KEY5_0](#) or [KEY6_0](#). These locations are ECC-protected. Once you have programmed the 128-bit key value and read it back to confirm the correct value is programmed, write the raw bit pattern [0x010101](#) to [KEY5_VALID](#) or [KEY6_VALID](#) to mark the key as valid. The validity takes effect at the next reset of the OTP block.

Once a key is valid, the OTP storage locations for that key become inaccessible for both reads and writes. Only the OTP power-up state machine ([Section 13.3.4](#)) can read the key.

The effect of installing debug keys depends on which of key 5 and 6 are installed:

- If key 5 or key 6 is valid, and no matching key (either) has been entered through the RP-AP, all debug is disabled. This has the same effect as setting [CRIT1.DEBUG_DISABLE](#).
- If key 5 is valid, and no matching key (key 5 specifically) has been entered through the RP-AP, Secure debug is disabled. This has the same effect as writing [CRIT1.SECURE_DEBUG_DISABLE](#).

When both keys are installed, key 5 provides both Secure and Non-secure debug access, and key 6 provides Non-secure debug access only. When only a single key is installed, that key provides both Secure and Non-secure debug access.

To enter a key over SWD, first write a 1 to [DBGKEY.RESET](#). Then sequentially write 128 bits to [DBGKEY.DATA](#), each accompanied by a 1 written to [DBGKEY.PUSH](#). Write the data LSB-first, starting with the lowest-numbered OTP row.

Assuming you wrote a value that matched one of the installed debug keys, debug unlocks after the 128th push. The [SDeviceEn](#) and [DeviceEn](#) flags in the Mem-AP [CSW](#) registers indicate success or failure.

Failure to supply a matching key through the RP-AP disables debug if it would otherwise be enabled. However, supplying a key does not enable if it is already disabled for other reasons. For example, if [CRIT1.DEBUG_DISABLE](#) is set, and [DEBUGEN](#) is clear, debug is be disabled no matter the state of the debug keys and the RP-AP.

3.5.10. RP-AP

The RP-AP is a small register block which is always accessible over SWD. RP-AP access does not require the switched core domain to be powered up, or any internal system clock generators to be running.

3.5.10.1. List of Registers

The RP-AP registers start at offset [0x80000](#) in the debug address space, which is accessed via address [0x80000](#) in the SW-DP's SELECT register. Unlike the other APs, it can not be accessed directly from the system bus.

Table 95. List of RP_AP registers

Offset	Name	Info
0x000	CTRL	This register is primarily used for DFT but can also be used to overcome some power up problems. However, it should not be used to force power up of domains. Use DBG_POW_OVRD for that.
0x004	DBGKEY	Serial key load interface (write-only)
0x008	DBG_POW_STATE_SWCORE	<p>This register indicates the state of the power sequencer for the switched-core domain.</p> <p>The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.</p> <p>Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.</p> <p>Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.</p> <p>Bits 9-11 describe the states of the power manager clocks which change as clock generators in the switched-core become available following switched-core power up.</p> <p>This bus can be sent to GPIO for debug. See DBG_POW_OUTPUT_TO_GPIO in the DBG_POW_OVRD register.</p>
0x00c	DBG_POW_STATE_XIP	<p>This register indicates the state of the power sequencer for the XIP domain.</p> <p>The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.</p> <p>Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.</p> <p>Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.</p>
0x010	DBG_POW_STATE_SRAM0	<p>This register indicates the state of the power sequencer for the SRAM0 domain.</p> <p>The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.</p> <p>Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.</p> <p>Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.</p>

Offset	Name	Info
0x014	DBG_POW_STATE_SRAM1	<p>This register indicates the state of the power sequencer for the SRAM1 domain.</p> <p>The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.</p> <p>Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.</p> <p>Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.</p>
0x018	DBG_POW_OVRD	<p>This register allows external control of the power sequencer outputs for all the switched power domains. If any of the power sequencers stall at any stage then force power up operation of all domains by running this sequence:</p> <ul style="list-style-type: none"> - set DBG_POW_OVRD = 0x3b to force small power switches on, large power switches off, resets on and isolation on - allow time for the domain power supplies to reach full rail - set DBG_POW_OVRD = 0x3b to force large power switches on - set DBG_POW_OVRD = 0x37 to remove isolation - set DBG_POW_OVRD = 0x17 to remove resets
0x01c	DBG_POW_OUTPUT_TO_GPIO	<p>Send some, or all, bits of DBG_POW_STATE_SWCORE to gpios.</p> <p>Bit 0 sends bit 0 of DBG_POW_STATE_SWCORE to GPIO 34</p> <p>Bit 1 sends bit 1 of DBG_POW_STATE_SWCORE to GPIO 35</p> <p>Bit 2 sends bit 2 of DBG_POW_STATE_SWCORE to GPIO 36</p> <p>.</p> <p>.</p> <p>Bit 11 sends bit 11 of DBG_POW_STATE_SWCORE to GPIO 45</p>
0xdfc	IDR	Standard Coresight ID Register

RP_AP: CTRL Register

Offset: 0x000

Description

This register is primarily used for DFT but can also be used to overcome some power up problems. However, it should not be used to force power up of domains. Use DBG_POW_OVRD for that.

Table 96. CTRL Register

Bits	Name	Description	Type	Reset
31	RESCUE_RESTART	Allows debug of boot problems by restarting the chip with minimal boot code execution. Write to 1 to put the chip in reset then write to 0 to restart the chip with the rescue flag set. The rescue flag is in the POWMAN_CHIP_RESET register and is read by boot code. The rescue flag is cleared by writing 0 to POWMAN_CHIP_RESET_RESCUE_FLAG or by resetting the chip by any means other than RESCUE_RESTART.	RW	0x0
30	SPARE	Unused	RW	0x0
29:7	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
6	DBG_FRCE_GPIO_LPCK	Allows chip start-up when the Low Power Oscillator (LPOSC) is inoperative or malfunctioning and also allows the initial power sequencing rate to be adjusted. Write to 1 to force the LPOSC output to be driven from a GPIO (gpio20 on 80-pin package, gpio34 on the 60-pin package). If the LPOSC is inoperative or malfunctioning it may also be necessary to set the LPOSC_STABLE_FRCE bit in this register. The user must provide a clock on the GPIO. For normal operation use a clock running at around 32kHz. Adjusting the frequency will speed up or slow down the initial power-up sequence.	RW	0x0
5	LPOSC_STABLE_FRCE	Allows the chip to start-up even though the Low Power Oscillator (LPOSC) is failing to set its stable flag. Initial power sequencing is clocked by LPOSC at around 32kHz but does not start until the LPOSC declares itself to be stable. If the LPOSC is otherwise working correctly the chip will boot when this bit is set. If the LPOSC is not working then DBG_FRCE_GPIO_LPCK must be set and an external clock provided.	RW	0x0
4	POWMAN_DFT_IS_O_OFF	Holds the isolation gates between power domains in the open state. This is intended to hold the gates open for DFT and power manager debug. It is not intended to force the isolation gates open. Use the overrides in DBG_POW_OVRD to force the isolation gates open or closed.	RW	0x0
3	POWMAN_DFT_PWRON	Holds the power switches on for all domains. This is intended to keep the power on for DFT and debug, rather than for switching the power on. The power switches are not sequenced and the sudden demand for current could cause the always-on power domain to brown out. This register is in the always-on domain therefore chaos could ensue. It is recommended to use the DBG_POW_OVRD controls instead.	RW	0x0
2	POWMAN_DBGMODE	This prevents the power manager from powering down and resetting the switched-core power domain. It is intended for DFT and for debugging the power manager after the chip has booted. It cannot be used to force initial power on because it simultaneously deasserts the reset.	RW	0x0
1	JTAG_FUNCSEL	Multiplexes the JTAG ports onto GPIO0-3	RW	0x0
0	JTAG_TRSTN	Resets the JTAG module. Active low.	RW	0x0

RP_AP: DBGKEY Register

Offset: 0x004

Description

Serial key load interface (write-only)

Table 97. DBGKEY Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
2	RESET	Reset (before sending a new key)	RW	0x0
1	PUSH		RW	0x0
0	DATA		RW	0x0

RP_AP: DBG_POW_STATE_SWCORE Register

Offset: 0x008

Description

This register indicates the state of the power sequencer for the switched-core domain.

The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.

Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.

Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.

Bits 9-11 describe the states of the power manager clocks which change as clock generators in the switched-core become available following switched-core power up.

This bus can be sent to GPIO for debug. See DBG_POW_OUTPUT_TO_GPIO in the DBG_POW_OVRD register.

Table 98.
DBG_POW_STATE_SW
CORE Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	USING_FAST_POWCK	Indicates the source of the power manager clock. On switched-core power up the clock switches from the LPOSC to clk_ref and this flag will be set. clk_ref will be running from the ROSC initially but will switch to XOSC when it comes available. On switched-core power down the clock switches to LPOSC and this flag will be cleared.	RO	0x0
10	WAITING_POWCK	Indicates the switched-core power sequencer is waiting for the power manager clock to update. On switched-core power up the clock switches from the LPOSC to clk_ref. clk_ref will be running from the ROSC initially but will switch to XOSC when it comes available. On switched-core power down the clock switches to LPOSC. If the switched-core power up sequence stalls with this flag active then it means clk_ref is not running which indicates a problem with the ROSC. If that happens then set DBG_POW_RESTART_FROM_XOSC in the DBG_POW_OVRD register to avoid using the ROSC. If the switched-core power down sequence stalls with this flag active then it means LPOSC is not running. The solution is to not stop LPOSC when the switched-core power domain is powered.	RO	0x0

Bits	Name	Description	Type	Reset
9	WAITING_TIMCK	Indicates that the switched-core power sequencer is waiting for the AON-Timer to update. On switched-core power-up there is nothing to be done. The AON-Timer continues to run from the LPOSC so this flag will not be set. Software decides whether to switch the AON-Timer clock to XOSC (via clk_ref). On switched-core power-down the sequencer will switch the AON-Timer back to LPOSC if software switched it to XOSC. During the switchover the WAITING_TIMCK flag will be set. If the switched-core power down sequence stalls with this flag active then the only recourse is to reset the chip and change software to not select XOSC as the AON-Timer source.	RO	0x0
8	IS_PU	Indicates the power somain is fully powered up.	RO	0x0
7	RESET_FROM_SE_Q	Indicates the state of the reset to the power domain.	RO	0x0
6	ENAB_ACK	Indicates the state of the enable to the power domain.	RO	0x0
5	ISOLATE_FROM_SE_EQ	Indicates the state of the isolation control to the power domain.	RO	0x0
4	LARGE_ACK	Indicates the state of the large power switches for the power domain.	RO	0x0
3	SMALL_ACK2	The small switches are split into 3 chains. In the power up sequence they are switched on separately to allow management of the VDD rise time. In the power down sequence they switch off simultaneously with the large power switches. This bit indicates the state of the last element in small power switch chain 2.	RO	0x0
2	SMALL_ACK1	This bit indicates the state of the last element in small power switch chain 1.	RO	0x0
1	SMALL_ACK0	This bit indicates the state of the last element in small power switch chain 0.	RO	0x0
0	IS_PD	Indicates the power somain is fully powered down.	RO	0x0

RP_AP: DBG_POW_STATE_XIP Register

Offset: 0x00c

Description

This register indicates the state of the power sequencer for the XIP domain.

The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.

Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.

Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.

Table 99.
DBG_POW_STATE_XIP
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	IS_PU	Indicates the power somain is fully powered up.	RO	0x0

Bits	Name	Description	Type	Reset
7	RESET_FROM_SE_Q	Indicates the state of the reset to the power domain.	RO	0x0
6	ENAB_ACK	Indicates the state of the enable to the power domain.	RO	0x0
5	ISOLATE_FROM_SE_EQ	Indicates the state of the isolation control to the power domain.	RO	0x0
4	LARGE_ACK	Indicates the state of the large power switches for the power domain.	RO	0x0
3	SMALL_ACK2	The small switches are split into 3 chains. In the power up sequence they are switched on separately to allow management of the VDD rise time. In the power down sequence they switch off simultaneously with the large power switches. This bit indicates the state of the last element in small power switch chain 2.	RO	0x0
2	SMALL_ACK1	This bit indicates the state of the last element in small power switch chain 1.	RO	0x0
1	SMALL_ACK0	This bit indicates the state of the last element in small power switch chain 0.	RO	0x0
0	IS_PD	Indicates the power domain is fully powered down.	RO	0x0

RP_AP: DBG_POW_STATE_SRAM0 Register

Offset: 0x010

Description

This register indicates the state of the power sequencer for the SRAM0 domain.

The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.

Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.

Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.

Table 100.
DBG_POW_STATE_SR
AM0 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	IS_PU	Indicates the power domain is fully powered up.	RO	0x0
7	RESET_FROM_SE_Q	Indicates the state of the reset to the power domain.	RO	0x0
6	ENAB_ACK	Indicates the state of the enable to the power domain.	RO	0x0
5	ISOLATE_FROM_SE_EQ	Indicates the state of the isolation control to the power domain.	RO	0x0
4	LARGE_ACK	Indicates the state of the large power switches for the power domain.	RO	0x0

Bits	Name	Description	Type	Reset
3	SMALL_ACK2	The small switches are split into 3 chains. In the power up sequence they are switched on separately to allow management of the VDD rise time. In the power down sequence they switch off simultaneously with the large power switches. This bit indicates the state of the last element in small power switch chain 2.	RO	0x0
2	SMALL_ACK1	This bit indicates the state of the last element in small power switch chain 1.	RO	0x0
1	SMALL_ACK0	This bit indicates the state of the last element in small power switch chain 0.	RO	0x0
0	IS_PD	Indicates the power somain is fully powered down.	RO	0x0

RP_AP: DBG_POW_STATE_SRAM1 Register

Offset: 0x014

Description

This register indicates the state of the power sequencer for the SRAM1 domain.

The sequencer timing is managed by the POWMAN_SEQ_* registers. See the header file for those registers for more information on the timing.

Power up of the domain commences by clearing bit 0 (IS_PD) then bits 1-8 are set in sequence. Bit 8 (IS_PU) indicates the sequence is complete.

Power down of the domain commences by clearing bit 8 (IS_PU) then bits 7-1 are cleared in sequence. Bit 0 (IS_PU) is then set to indicate the sequence is complete.

Table 101.
DBG_POW_STATE_SR
AM1 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	IS_PU	Indicates the power somain is fully powered up.	RO	0x0
7	RESET_FROM_SE_Q	Indicates the state of the reset to the power domain.	RO	0x0
6	ENAB_ACK	Indicates the state of the enable to the power domain.	RO	0x0
5	ISOLATE_FROM_SE_EQ	Indicates the state of the isolation control to the power domain.	RO	0x0
4	LARGE_ACK	Indicates the state of the large power switches for the power domain.	RO	0x0
3	SMALL_ACK2	The small switches are split into 3 chains. In the power up sequence they are switched on separately to allow management of the VDD rise time. In the power down sequence they switch off simultaneously with the large power switches. This bit indicates the state of the last element in small power switch chain 2.	RO	0x0
2	SMALL_ACK1	This bit indicates the state of the last element in small power switch chain 1.	RO	0x0
1	SMALL_ACK0	This bit indicates the state of the last element in small power switch chain 0.	RO	0x0
0	IS_PD	Indicates the power somain is fully powered down.	RO	0x0

RP_AP: DBG_POW_OVRD Register

Offset: 0x018

Description

This register allows external control of the power sequencer outputs for all the switched power domains. If any of the power sequencers stall at any stage then force power up operation of all domains by running this sequence:

- set DBG_POW_OVRD = 0x3b to force small power switches on, large power switches off, resets on and isolation on
- allow time for the domain power supplies to reach full rail
- set DBG_POW_OVRD = 0x3b to force large power switches on
- set DBG_POW_OVRD = 0x37 to remove isolation
- set DBG_POW_OVRD = 0x17 to remove resets

Table 102.
DBG_POW_OVRD
Register

Bits	Name	Description	Type	Reset
31:7	Reserved.	-	-	-
6	DBG_POW_RSTA RT_FROM_XOSC	By default the system begins boot as soon as a clock is available from the ROSC, then it switches to the XOSC when it is available. This is done because the XOSC takes several ms to start up. If there is a problem with the ROSC then the default behaviour can be changed to not use the ROSC and wait for XOSC. However, this requires a mask change to modify the reset value of the Power Manager START_FROM_XOSC register. To allow experimentation the default can be temporarily changed by setting this register bit to 1. After setting this bit the core must be reset by a Coresight dprst or a rescue reset (see RESCUE_RESTART in the RP_AP_CTRL register above). A power-on reset, brown-out reset or RUN pin reset will reset this control and revert to the default behaviour.	RW	0x0
5	DBG_POW_RESET	When DBG_POW_OVRD_RESET=1 this register bit controls the resets for all domains. 1 = reset. 0 = not reset.	RW	0x0
4	DBG_POW_OVRD_ RESET	Enables DBG_POW_RESET to control the resets for the power manager and the switched-core. Essentially that is everything except the Coresight 2-wire interface and the RP_AP registers.	RW	0x0
3	DBG_POW_ISO	When DBG_POW_OVRD_ISO=1 this register bit controls the isolation gates for all domains. 1 = isolated. 0 = not isolated.	RW	0x0
2	DBG_POW_OVRD_ ISO	Enables DBG_POW_ISO to control the isolation gates between domains.	RW	0x0
1	DBG_POW_OVRD_ LARGE_REQ	Turn on the large power switches for all domains. This should not be done until sufficient time has been allowed for the small switches to bring the supplies up. Switching the large switches on too soon risks browning out the always-on domain and corrupting these very registers.	RW	0x0

Bits	Name	Description	Type	Reset
0	DBG_POW_OVRD_SMALL_REQ	Turn on the small power switches for all domains. This switches on chain 0 for each domain and switches off chains 2 & 3 and the large power switch chain. This will bring the power up for all domains without browning out the always-on power domain.	RW	0x0

RP_AP: DBG_POW_OUTPUT_TO_GPIO Register

Offset: 0x01c

Description

Send some, or all, bits of DBG_POW_STATE_SWCORE to gpios.

Bit 0 sends bit 0 of DBG_POW_STATE_SWCORE to GPIO 34

Bit 1 sends bit 1 of DBG_POW_STATE_SWCORE to GPIO 35

Bit 2 sends bit 2 of DBG_POW_STATE_SWCORE to GPIO 36

1. +

2. + Bit 11 sends bit 11 of DBG_POW_STATE_SWCORE to GPIO 45

Table 103.
DBG_POW_OUTPUT_T
O_GPIO Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:0	ENABLE		RW	0x000

RP_AP: IDR Register

Offset: 0xdxfc

Table 104. IDR
Register

Bits	Description	Type	Reset
31:0	Standard Coresight ID Register	RO	-

3.6. Cortex-M33 Coprocessors

The Cortex-M33 features a coprocessor port which transfers up to 64 bits per cycle between the processor and certain closely-coupled hardware. The Cortex-M33's built-in floating-point unit is an example of such a coprocessor, but RP2350 adds three device-specific coprocessors to this interface. The following sections document these coprocessors.

Before accessing a coprocessor from Secure code, that coprocessor must first be enabled by setting the corresponding bit in the CPACR. Before accessing from the Non-secure state, the corresponding bits in the NSACR and CPACR_NS registers must be set.

The RISC-V processors on RP2350 do not have access to the Cortex-M33 coprocessors.

3.6.1. GPIO Coprocessor (GPIOC)

Coprocessor port 0 provides low-overhead access from the Cortex-M33 processors to the GPIO registers in the SIO (Section 3.1.3). This enables a single coprocessor instruction to sample all 48 GPIOs, or to set/clear/write any single GPIO, among other functionality.

Non-secure accesses are filtered according to the [GPIO_NSMASK0](#) and [GPIO_NSMASK1](#) registers in ACCESSCTRL. GPIOs not granted for Non-secure use will ignore writes from the Non-secure state, and read back as zero when read

from the Non-secure state.

3.6.1.1. OUT Mask Write Instructions

These instructions write to multiple bits in the SIO [GPIO_OUT](#) and [GPIO_HI_OUT](#) registers.

Mnemonic	Armv8-M Instruction	Operation
gpioc_lo_out_put	<code>mcr p0, #0, Rt, c0, c0</code>	<code>sio_hw>gpio_out = Rt;</code>
gpioc_lo_out_xor	<code>mcr p0, #1, Rt, c0, c0</code>	<code>sio_hw>gpio_togl = Rt;</code>
gpioc_lo_out_set	<code>mcr p0, #2, Rt, c0, c0</code>	<code>sio_hw>gpio_set = Rt;</code>
gpioc_lo_out_clr	<code>mcr p0, #3, Rt, c0, c0</code>	<code>sio_hw>gpio_clr = Rt;</code>
gpioc_hi_out_put	<code>mcr p0, #0, Rt, c0, c1</code>	<code>sio_hw>gpio_hi_out = Rt;</code>
gpioc_hi_out_xor	<code>mcr p0, #1, Rt, c0, c1</code>	<code>sio_hw>gpio_hi_togl = Rt;</code>
gpioc_hi_out_set	<code>mcr p0, #2, Rt, c0, c1</code>	<code>sio_hw>gpio_hi_set = Rt;</code>
gpioc_hi_out_clr	<code>mcr p0, #3, Rt, c0, c1</code>	<code>sio_hw>gpio_hi_clr = Rt;</code>
gpioc_hilo_out_put	<code>mcrr p0, #0, Rt, Rt2, c0</code>	Simultaneously: <code>sio_hw>gpio_out = Rt; sio_hw>gpio_hi_out = Rt2;</code>
gpioc_hilo_out_xor	<code>mcrr p0, #1, Rt, Rt2, c0</code>	Simultaneously: <code>sio_hw>gpio_togl = Rt; sio_hw>gpio_hi_togl = Rt2;</code>
gpioc_hilo_out_set	<code>mcrr p0, #2, Rt, Rt2, c0</code>	Simultaneously: <code>sio_hw>gpio_set = Rt; sio_hw>gpio_hi_set = Rt2;</code>
gpioc_hilo_out_clr	<code>mcrr p0, #3, Rt, Rt2, c0</code>	Simultaneously: <code>sio_hw>gpio_clr = Rt; sio_hw>gpio_hi_clr = Rt2;</code>

3.6.1.2. OE Mask Write Instructions

These instructions write to multiple bits in the SIO [GPIO_OE](#) and [GPIO_HI_OE](#) registers.

Mnemonic	Armv8-M Instruction	Operation
gpioc_lo_oe_put	<code>mcr p0, #0, Rt, c0, c4</code>	<code>sio_hw>gpio_oe = Rt;</code>
gpioc_lo_oe_xor	<code>mcr p0, #1, Rt, c0, c4</code>	<code>sio_hw>gpio_oe_togl = Rt;</code>
gpioc_lo_oe_set	<code>mcr p0, #2, Rt, c0, c4</code>	<code>sio_hw>gpio_oe_set = Rt;</code>
gpioc_lo_oe_clr	<code>mcr p0, #3, Rt, c0, c4</code>	<code>sio_hw>gpio_oe_clr = Rt;</code>
gpioc_hi_oe_put	<code>mcr p0, #0, Rt, c0, c5</code>	<code>sio_hw>gpio_hi_oe = Rt;</code>
gpioc_hi_oe_xor	<code>mcr p0, #1, Rt, c0, c5</code>	<code>sio_hw>gpio_hi_oe_togl = Rt;</code>
gpioc_hi_oe_set	<code>mcr p0, #2, Rt, c0, c5</code>	<code>sio_hw>gpio_hi_oe_set = Rt;</code>
gpioc_hi_oe_clr	<code>mcr p0, #3, Rt, c0, c5</code>	<code>sio_hw>gpio_hi_oe_clr = Rt;</code>
gpioc_hilo_oe_put	<code>mcrr p0, #0, Rt, Rt2, c4</code>	Simultaneously: <code>sio_hw>gpio_oe = Rt; sio_hw>gpio_hi_oe = Rt2;</code>
gpioc_hilo_oe_xor	<code>mcrr p0, #1, Rt, Rt2, c4</code>	Simultaneously: <code>sio_hw>gpio_oe_togl = Rt; sio_hw>gpio_hi_oe_togl = Rt2;</code>
gpioc_hilo_oe_set	<code>mcrr p0, #2, Rt, Rt2, c4</code>	Simultaneously: <code>sio_hw>gpio_oe_set = Rt; sio_hw>gpio_hi_oe_set = Rt2;</code>
gpioc_hilo_oe_clr	<code>mcrr p0, #3, Rt, Rt2, c4</code>	Simultaneously: <code>sio_hw>gpio_oe_clr = Rt; sio_hw>gpio_hi_oe_clr = Rt2;</code>

3.6.1.3. Single-bit Write Instructions

These instructions write to a single, indexed bit in either the `GPIO_OUT` and `GPIO_HI_OUT` registers, or the `GPIO_OE` and `GPIO_HI_OE` registers.

Mnemonic	Armv8-M Instruction	Operation
<code>gpioc_bit_out_put</code>	<code>mcrr p0, #4, Rt, Rt2, c0</code>	Write a 1-bit value to any output. Equivalent to: <code>if (Rt2 & 1) gpioc_hilo_out_set(1ull << Rt); else gpioc_hilo_out_clr(1ull << Rt);</code>
<code>gpioc_bit_out_xor</code>	<code>mcr p0, #5, Rt, c0, c0</code>	Unconditionally toggle any single output. Equivalent to: <code>gpioc_hilo_out_xor(1ull << Rt);</code>
<code>gpioc_bit_out_set</code>	<code>mcr p0, #6, Rt, c0, c0</code>	Unconditionally set any single output. Equivalent to: <code>gpioc_hilo_out_set(1ull << Rt);</code>
<code>gpioc_bit_out_clr</code>	<code>mcr p0, #7, Rt, c0, c0</code>	Unconditionally clear any single output. Equivalent to: <code>gpioc_hilo_out_clr(1ull << Rt);</code>
<code>gpioc_bit_out_xor2</code>	<code>mcrr p0, #5, Rt, Rt2, c0</code>	Conditionally toggle any single output. Equivalent to: <code>gpioc_hilo_out_xor((uint64_t)(Rt2 & 1) << Rt);</code>
<code>gpioc_bit_out_set2</code>	<code>mcrr p0, #6, Rt, Rt2, c0</code>	Conditionally set any single output. Equivalent to: <code>gpioc_hilo_out_set((uint64_t)(Rt2 & 1) << Rt);</code>
<code>gpioc_bit_out_clr2</code>	<code>mcrr p0, #7, Rt, Rt2, c0</code>	Conditionally clear any single output. Equivalent to: <code>gpioc_hilo_out_clr((uint64_t)(Rt2 & 1) << Rt);</code>
<code>gpioc_bit_oe_put</code>	<code>mcrr p0, #4, Rt, Rt2, c4</code>	Write a 1-bit value to any output enable. Equivalent to: <code>if (Rt2 & 1) gpioc_hilo_oe_set(1ull << Rt); else gpioc_hilo_oe_clr(1ull << Rt);</code>
<code>gpioc_bit_oe_xor</code>	<code>mcr p0, #5, Rt, c0, c4</code>	Unconditionally toggle any output enable. Equivalent to: <code>gpioc_hilo_oe_xor(1ull << Rt);</code>
<code>gpioc_bit_oe_set</code>	<code>mcr p0, #6, Rt, c0, c4</code>	Unconditionally set any output enable (set to output). Equivalent to: <code>gpioc_hilo_oe_set(1ull << Rt);</code>
<code>gpioc_bit_oe_clr</code>	<code>mcr p0, #7, Rt, c0, c4</code>	Unconditionally clear any output enable (set to input). Equivalent to: <code>gpioc_hilo_oe_clr(1ull << Rt);</code>
<code>gpioc_bit_oe_xor2</code>	<code>mcrr p0, #5, Rt, Rt2, c4</code>	Conditionally toggle any output enable. Equivalent to: <code>gpioc_hilo_oe_xor((uint64_t)(Rt2 & 1) << Rt);</code>
<code>gpioc_bit_oe_set2</code>	<code>mcrr p0, #6, Rt, Rt2, c4</code>	Conditionally set any output enable (set to output). Equivalent to: <code>gpioc_hilo_oe_set((uint64_t)(Rt2 & 1) << Rt);</code>
<code>gpioc_bit_oe_clr2</code>	<code>mcrr p0, #7, Rt, Rt2, c4</code>	Conditionally clear any output enable (set to input). Equivalent to: <code>gpioc_hilo_oe_clr((uint64_t)(Rt2 & 1) << Rt);</code>

3.6.1.4. Indexed Mask Write Instructions

These instructions write to a single, dynamically selected 32-bit GPIO register.

Mnemonic	Armv8-M Instruction	Operation
<code>gpioc_index_out_put</code>	<code>mcrr p0, #8, Rt, Rt2, c0</code>	Write <code>Rt</code> to a GPIO output register selected by <code>Rt2</code> .
<code>gpioc_index_out_xor</code>	<code>mcrr p0, #9, Rt, Rt2, c0</code>	Toggle bits <code>Rt</code> in a GPIO output register selected by <code>Rt2</code> .
<code>gpioc_index_out_set</code>	<code>mcrr p0, #10, Rt, Rt2, c0</code>	Set bits <code>Rt</code> in a GPIO output register selected by <code>Rt2</code> .
<code>gpioc_index_out_clr</code>	<code>mcrr p0, #11, Rt, Rt2, c0</code>	Clear bits <code>Rt</code> in a GPIO output register selected by <code>Rt2</code> .
<code>gpioc_index_oe_put</code>	<code>mcrr p0, #8, Rt, Rt2, c4</code>	Write <code>Rt</code> to a GPIO output enable register selected by <code>Rt2</code>
<code>gpioc_index_oe_xor</code>	<code>mcrr p0, #9, Rt, Rt2, c4</code>	Toggle bits <code>Rt</code> in a GPIO output enable register selected by <code>Rt2</code> .

Mnemonic	Armv8-M Instruction	Operation
gpioc_index_oe_set	<code>mcr p0, #10, Rt, Rt2, c4</code>	Set bits <code>Rt</code> in a GPIO output enable register selected by <code>Rt2</code> (i.e. set to output).
gpioc_index_oe_clr	<code>mcr p0, #11, Rt, Rt2, c4</code>	Clear bits <code>Rt</code> in a GPIO output enable register selected by <code>Rt2</code> (i.e. set to input).

3.6.1.5. Read Instructions

These instructions read from either the `GPIO_OUT` and `GPIO_HI_OUT` registers; the `GPIO_OE` and `GPIO_HI_OE` registers; or the `GPIO_IN` and `GPIO_HI_IN` registers.

Mnemonic	Armv8-M Instruction	Operation
gpioc_lo_out_get	<code>mrc p0, #0, Rt, c0, c0</code>	Read back the lower 32-bit output register. Equivalent to: <code>Rt = sio_hw>gpio_out;</code>
gpioc_hi_out_get	<code>mrc p0, #0, Rt, c0, c1</code>	Read back the upper 32-bit output register. Equivalent to: <code>Rt = sio_hw>gpio_hi_out;</code>
gpioc_hilo_out_get	<code>mrrc p0, #0, Rt, Rt2, c0</code>	Read back two 32-bit output registers in a single operation. Equivalent to: <code>Rt = sio_hw>gpio_out;</code> and simultaneously <code>Rt2 = sio_hw>gpio_hi_out << 32;</code>
gpioc_lo_oe_get	<code>mrc p0, #0, Rt, c0, c4</code>	Read back the lower 32-bit output enable register. Equivalent to: <code>Rt = sio_hw>gpio_oe;</code>
gpioc_hi_oe_get	<code>mrc p0, #0, Rt, c0, c5</code>	Read back the upper 32-bit output enable register. Equivalent to: <code>Rt = sio_hw>gpio_hi_oe;</code>
gpioc_hilo_oe_get	<code>mrrc p0, #0, Rt, Rt2, c4</code>	Read back two 32-bit output enable registers in a single operation. Equivalent to: <code>Rt = sio_hw>gpio_oe;</code> and simultaneously <code>Rt2 = sio_hw>gpio_hi_oe << 32;</code>
gpioc_lo_in_get	<code>mrc p0, #0, Rt, c0, c8</code>	Sample the lower 32 GPIOs. Equivalent to: <code>Rt = sio_hw>gpio_in;</code>
gpioc_hi_in_get	<code>mrc p0, #0, Rt, c0, c9</code>	Sample the upper 32 GPIOs. Equivalent to: <code>Rt = sio_hw>gpio_hi_in;</code>
gpioc_hilo_in_get	<code>mrrc p0, #0, Rt, Rt2, c8</code>	Sample 64 GPIOs on the same cycle. Equivalent to: <code>Rt = sio_hw>gpio_in;</code> and simultaneously <code>Rt2 = sio_hw>gpio_hi_in << 32;</code>

3.6.1.6. Interpreting Instruction Fields

The type of coprocessor instruction – `mrc`, `mrrc`, `mcr` and `mcrr` – specifies the direction of the transfer (read/write) and the number of Arm registers being transferred (one or two).

Bits 3:2 of the first coprocessor register number field, `CRm`, identify the group of registers being accessed. Values 0, 1 and 2 refer to the output, output enable and input registers respectively.

Bit 0 of the first coprocessor register number field, `CRm`, may be used to distinguish which register in a group is being accessed. Bit 1 is reserved to allow more registers to be indexed on future chips with more GPIOs.

For writes, bits 1:0 of the instruction's `opc1` field specify the type of write operation: values 0, 1, 2, 3 map to normal write, XOR, set and clear operations respectively. Bits 3:2 of the `opc1` field are used to indicate the addressing mode for the register or individual bit being accessed. Their exact interpretation depends on the instruction.

Any combinations not listed in the preceding tables are reserved for future use.

3.6.2. Double-precision Coprocessor (DCP)

Each Cortex-M33 CPU core is equipped with two instances of a double-precision coprocessor that provides acceleration of double-precision floating point operations including add, subtract, multiply, divide and square root. The design is implemented in just a few thousand gates and so occupies much less silicon die area than a full double-precision floating-point unit. Nevertheless, these coprocessors considerably speed up basic double-precision operations compared to pure software implementations. The coprocessors also offer support for some single-precision operations and conversions.

The two coprocessor instances are assigned to the Secure and Non-secure domains. Coprocessor number [4](#) always maps to the coprocessor used for the current processor security state. Coprocessor number [5](#) always maps to the Non-secure coprocessor instance, but is accessible only from Secure code. This duplication avoids saving and restoring the coprocessor context during Secure/Non-secure state transitions.

3.6.2.1. CPU Interface

As with the other coprocessors, the accelerator connects to the CPU over a 64-bit bus. Two words of data can be transferred per cycle over that bus using the following instructions:

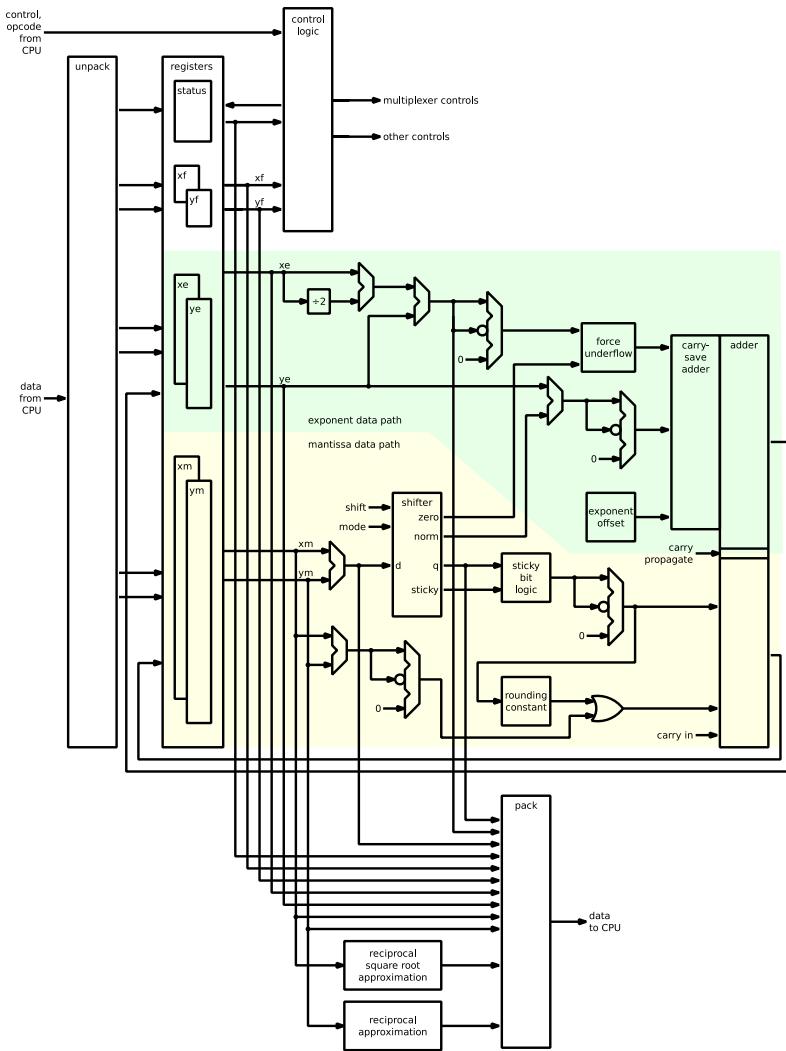
- [MCRR](#): move two integer registers to coprocessor
- [MRRC](#): move two integer registers from coprocessor

There are also single-register versions of these instructions, including ones that allow the CPU's flags to be loaded from the coprocessor. The CPU issues [CDP](#) instructions to trigger operations within the coprocessor without transferring any data.

3.6.2.2. Internal architecture

A block diagram of the accelerator is shown in [Figure 11](#).

Figure 11. Block diagram of double-precision accelerator



At the heart of the design are:

- two sets of registers, each designed to hold an unpacked double-precision value
- a 9-bit status register

Unlike a conventional FPU, the accelerator does not contain a full register bank. Not only does this save die area, it also means that saving and restoring the coprocessor's state is very fast: in fact, the entire state fits within six 32-bit words and hence can be saved to, or restored from, the CPU in three instructions.

The accelerator contains a wide adder, capable of adding two mantissa values and three exponent values simultaneously. There is also a shifter that can either perform a logical right shift by a given amount, or normalise a denormalised mantissa and report the amount of shift required to do so. A considerable amount of hardware in the shifter is shared between these two operating modes.

Control logic, shown at the top of the diagram, decodes coprocessor instructions and configures the accelerator's functional units and datapath multiplexers in order to execute the desired operation. Each coprocessor instruction takes a single cycle, so coprocessor operations cannot stall the CPU.

A floating-point operation such as addition or subtraction is carried out by executing a fixed (or 'canned') sequence of instructions as follows:

1. One or two **MCRR** instructions to write the operands to the coprocessor.
2. A number of **CDP** (and possibly other) instructions that together perform the operation itself.
3. An **MRRC** or **MRC** instruction to read back the result.

The hardware handles special cases involving zeroes, NaNs, and infinities, as well as rounding, underflow and overflow.

The accelerator does not contain a multiplier array, as that would occupy a considerable amount of die area. Instead, the mantissas of the operands of a multiplication operation are brought back into the CPU to take advantage of the fast long multiply instructions available there. The coprocessor handles the processing of exponents.

Division and square root operations also involve data moving back and forth between coprocessor and CPU. To assist with these operations, the coprocessor contains two small lookup tables (implemented as random logic) that provide initial approximations used in the divide and square root algorithms. The coprocessor handles the processing of exponents.

The accelerator is only meant to be used with the canned instruction sequences that implement basic floating-point operations. The state of the accelerator is not guaranteed to be preserved from the end of one canned sequence to the beginning of the next: see the discussion of the 'engaged' flag in the status register below.

3.6.2.3. Registers

X and Y mantissa registers

The X and Y mantissa registers (`xm` and `ym`) are each 64 bits wide. They can be read and written directly by the CPU; the `xm` register can also store the lower part of the result from the adder. When a value is written to the coprocessor using a 'write unpacked' `MCRR` instruction, the top two bits of the mantissa register are set to `01` and the next most significant bits are filled from the mantissa field of the floating-point operand. The low-order bits of the mantissa register are cleared.

X and Y exponent registers

The X and Y exponent registers (`xe` and `ye`) are each 14 bits wide. They can be read and written directly by the CPU; the `xe` register can also store the higher part of the result from the adder. When a value is written to the coprocessor using a 'write unpacked' `MCRR` instruction, the exponent register is set from the exponent field of the floating-point operand.

X and Y flag registers

The X and Y flag registers (`xf` and `yf`) are each four bits wide. They can be read and written directly by the CPU. The flag register stores information about the type of floating-point number represented in the corresponding mantissa and exponent registers: its sign, whether it is a zero, whether it is an infinity, and whether it is a NaN. When a value is written to the coprocessor using a 'write unpacked' `MCRR` instruction, the bits of the flag register are updated according to the type of the floating-point operand.

Status register

The status register contains nine bits. It can be read and written directly by the CPU. The least significant six bits of the register store the shift required to align the two operands of an addition or subtraction; the next two bits indicate whether the value represented by (`xe`, `xm`) is greater than, equal to, or less than the value represented by (`ye`, `ym`) - in other words, whether the magnitude of the value stored in the X registers is greater than, equal to, or less than the magnitude of the value stored in the Y registers. These status bits are set in the first step of an addition, subtraction or comparison operation after the operands have been loaded.

The final bit of the status register indicates whether the coprocessor is 'engaged'. The engaged flag is set by all coprocessor instructions that occur at the beginning or in the middle of the canned instruction sequences. It is cleared by those instructions used at the end of a canned sequence to read back a final result.

3.6.2.4. State save and restore

An interrupt handler can test the engaged flag to determine whether it has pre-empted an in-progress operation on the same coprocessor. If the engaged flag is set, the handler can save (and restore) the coprocessor state before using the coprocessor. If the engaged flag is clear, the save (and restore) step can be skipped. If this approach is implemented, the state of the accelerator must be regarded as undefined when not within one of the canned instruction sequences.

Three `MRRC` instructions are provided to copy the six words of state in the coprocessor into integer registers in the CPU, from where they can, for example, be pushed onto the stack. The last of these instructions clears the engaged flag.

Similarly, three **MCRR** instructions are provided to restore the state of the coprocessor from integer registers, including the state of the engaged flag.

3.6.2.5. Instruction summary

As mentioned above, it is intended that the coprocessor instructions are only used as part of canned sequences. Nevertheless, for completeness, a list of the available instructions is given here with an outline of their effects.

MCRR instructions are shown in [Table 105](#).

Table 105. *MCRR* instructions

Mnemonic	Effect	Used by
WXMD	write xm direct	restore status
WYMD	write ym direct	restore status
WEFD	write $xe, xf, ye, yf, \text{other status}$ direct	restore status
WXUP	write xm, xe, xf unpacked double-precision	double-precision binary operations
WYUP	write ym, ye, yf unpacked double-precision	double-precision binary operations
WXYU	write xm, xe, xf, ym, ye, yf two unpacked single-precision	single-precision binary operations
WXMS	write xm bit 0=0/1 if data zero/nonzero	dmul
WXMO	write xm direct OR into b0, add exponents, XOR signs	dmul
WXDD	write xm direct; subtract exponents, XOR signs	ddiv
WXDQ	write xm direct, offset exponent	dsqrt
WXUC	write X unsigned int $+2^{52}+2^{32}$, Y $=2^{52}+2^{32}$	conversions from unsigned int
WXIC	write X signed int $+2^{52}+2^{32}$, Y $=2^{52}+2^{32}$	conversions from signed int
WXOC	write X unpacked double-precision, Y $=2^{52}+2^{32}$	conversions from double-precision
WXFC	write X unpacked single-precision, Y $=2^{52}+2^{32}$	conversions from single-precision
WXFM	write xm direct, add exponents, XOR signs	fmul
WFXML	write xm direct, subtract exponents, XOR signs	fdiv
WXFQ	write xm direct, offset exponent	fsqrt

CDP instructions are shown in [Table 106](#).

Table 106. *CDP* instructions

Mnemonic	Effect	Used by
INIT	zero all registers	
ADD0	compare X-Y, set status	add, sub, cmp
ADD1	$xm := \pm xm + \pm ym >> s$ or $\pm ym + \pm xm >> s$	add
SUB1	$xm := \pm xm - \pm ym >> s$ or $-\pm ym \pm xm >> s$	sub
SQR0	$xe = xe / 2$, $xm = xm << 0:1$	sqrt

Mnemonic	Effect	Used by
NORM	normalise	
NRDF	normalise and round single-precision	single-precision operations, conversions to single-precision
NRDD	normalise and round double-precision	double-precision operations, conversions to double-precision
NTDC	normalise and truncate double-precision pre-integer conversion	truncating conversions to int
NRDC	normalise and round double-precision pre-integer conversion	rounding conversions to int

MRRC and **MRC** instructions are shown in Table 107.

Table 107. **MRRC** and **MRC** instructions

Mnemonic	Effect	Used by
RXVD	read xf ,VERSION direct	dclassify, check version
RCMP	read processed status	dcmp
RDFA	read FADD result packed from X	fadd
RDFS	read FSUB result packed from X	fsub
RDFM	read FMUL result packed from X	fmul
RDFD	read FDIV result packed from X	fdiv
RDFQ	read FSQRT result packed from X	fsqrt
RDFG	read general float result packed from X	double-precision to single-precision conversion
RDUC	read unsigned integer conversion result from X	conversions to unsigned int
RDIC	read signed integer conversion result from X	conversions to signed int
RXMD	read xm direct	save status
RYMD	read ym direct, engaged=0	save status
REFD	read xe,xf,ye,yf ,other status direct	save status
RXMS	read xm Q62-s	dmul, ddiv, dsqrt
RYMS	read ym Q62-s	dmul, ddiv
RXYH	read ym hi, xm hi	fmul, fdiv
RYMR	read ym hi, recip approximation lo	fdiv, ddiv
RXMQ	read xm hi, rsqrt approximation lo	fsqrt, dsqrt
RDDA	read DADD result packed from X	dadd
RDDS	read DSUB result packed from X	dsub
RDDM	read DMUL result packed from X	dmul
RDDD	read DDIV result packed from X	ddiv

Mnemonic	Effect	Used by
RDDQ	read DSQRT result packed from X	dsqrt
RDDG	read general double result packed from X	single-precision to double-precision conversion

Alongside each **MRRC** and **MRC** instruction is a variant starting **P** (for 'peek') instead of **R** that has the same function but preserves the engaged flag. **RXMD** is identical to **PXMD**; **REFD** is identical to **PEFD**.

The SDK includes macros to generate Arm assembler from the mnemonics above in the file **dcp_instr.inc.S** in the SDK, for example turning **WXUP r0,r1** into **mccr p4,#1,r0,r1,c0**.

3.6.2.6. Example canned sequence

The assembly code sequence to implement a callable double-precision addition operation is shown in [Table 108](#).

Table 108. Assembly code sequence to implement a callable double-precision addition operation

Arm assembler	Coprocessor mnemonic	Action
mccr p4,#1,r0,r1,c0	WXUP r0,r1	write R0 and R1 unpacked double-precision into X
mccr p4,#1,r2,r3,c1	WYUP r2,r3	write R2 and R3 unpacked double-precision into Y
cdp p4,#0,c0,c0,c1,#0	ADD0	compare X and Y ; set status and alignment shift
cdp p4,#1,c0,c0,c1,#0	ADD1	add/subtract (depending on status and signs) xm and ym aligned, write result to xm
cdp p4,#8,c0,c0,c0,#1	NRDD	normalise and round double-precision result
mrrc p4,#1,r0,r1,c0	RDDA r0,r1	read R0 and R1 packed double-precision from X , including special-value processing for addition
bx r14		return from function

Logic in the coprocessor ensures, for example, that the **ADD1** instruction shifts the smaller argument, that **xm** and **ym** are negated as required before being sent to the adder, and that the larger exponent is used as the basis for the subsequent normalisation.

3.6.2.7. Using the coprocessor via the SDK library

The SDK **pico_double** library automatically uses the coprocessor for double-precision floating-point calculations. This is the simplest way to take advantage of the coprocessor, but it entails a few cycles of overhead for each operation. Not only is there the overhead involved in a function call and return, but for safety the general-purpose implementations in the SDK always test the engaged flag, saving and restoring the coprocessor state to and from the stack as needed. That ensures that the functions work correctly in interrupt handlers or in multi-threaded code without additional intervention.

3.6.2.8. Using the coprocessor directly

The SDK includes macros to generate canned sequences for standard operations in the file **dcp_canned.inc.S** in the SDK.

These allow the callable double-precision addition operation listed above, for example, to be written as:

```
dcp_dadd_m r0,r1, r0,r1,r2,r3 @ result in r0,r1; operands in r0,r1 and r2,r3
bx r14
```

`dcp_dadd_m` is a macro which expands into the sequence of coprocessor instructions given above. This macro allows you to specify the integer registers to be used for the operands and the result, which means that using these macros directly not only avoids function call and return overhead, it also avoids the extra overhead associated with argument marshalling.

The more complex macros also require you to specify 'scratch' registers that they can use for storing intermediate results. The following function, which calculates the dot product of two three-element vectors of doubles pointed to by `R0` and `R1`, illustrates this:

```

push {r4-r9,r14}
ldrd r3,r4,[r0],#8          @ load x0
ldrd r5,r6,[r1],#8          @ load y0
dcp_dmull_m r7,r8, r3,r4,r5,r6, r3,r4,r5,r6,r12,r14,r9 @ compute x0y0 ①
ldrd r3,r4,[r0],#8          @ load x1
ldrd r5,r6,[r1],#8          @ load y1
dcp_dmull_m r3,r4, r3,r4,r5,r6, r3,r4,r5,r6,r12,r14,r9 @ compute x1y1 ①
dcp_dadd_m r7,r8, r3,r4,r7,r8          @ compute x0y0+x1y1
ldrd r3,r4,[r0],#8          @ load x2
ldrd r5,r6,[r1],#8          @ load y2
dcp_dmull_m r3,r4, r3,r4,r5,r6, r3,r4,r5,r6,r12,r14,r9 @ compute x2y2 ①
dcp_dadd_m r0,r1, r3,r4,r7,r8          @ compute x0y0+x1y1+x2y2 ②
pop {r4-r9,r15}

```

1. `r3, r4, r5, r6, r12, r14`, and `r9` are scratch registers.

2. stores the result in `r0, r1`.

NOTE

This example does not check the engaged flag. If used in interrupt handlers or in multi-threaded applications, a suitable test would have to be added. For example, see the [SDK implementation of `__aeabi_dadd`](#) for an efficient way to do this. The test only needs to be performed once, at the beginning of the function, so the overhead in this case would be relatively small.

The following example demonstrates how to use the coprocessor:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/dcp/hello_dcp/hello_dcp.c Lines 19 - 109

```

19 extern double dcp_dot          (double*p,double*q,int n);
20 extern double dcp_dotx         (float*p,float*q,int n);
21 extern float dcp_iirx          (float x,float*temp,float*coeff,int order);
22 extern void dcp_butterfly_radix2 (double*x,double*y);
23 extern void dcp_butterfly_radix2_twiddle_dif (double*x,double*y,double*tf);
24 extern void dcp_butterfly_radix2_twiddle_dit (double*x,double*y,double*tf);
25 extern void dcp_butterfly_radix4          (double*w,double*x,double*y,double*z);
26
27 static void dcp_test0() {
28     double u[3]={1,2,3};
29     double v[3]={4,5,6};
30     double w;
31     w=dcp_dot(u,v,3);
32     printf("(1,2,3).(4,5,6)=%g\n",w);
33 }
34
35 static void dcp_test1() {
36     float u[3]={1+pow(2,-20),2,3};
37     float v[3]={1-pow(2,-20),5,6};
38     double w;
39     w=dcp_dotx(u,v,3);

```

```

40     printf("(1+pow(2,-20),2,3).(1-pow(2,-20),5,6)=%.17g\n",w);
41 }
42
43 static void dcp_test2() {
44     int t;
45     float w;
46 // filter coefficients calculated using Octave as follows:
47 // octave> pkg load signal
48 // octave> format long
49 // octave> [b,a]=cheby1(2,1,.5)
50 // b = 0.307043201259064  0.614086402518128  0.307043201259064
51 // a = 1.000000000000000e+00  6.406405700380895e-02  3.139684953186774e-01
52 // and tested as follows:
53 // octave> filter(b,a,[1 zeros(1,19)])
54     float coeff[5]={0.3070432,0.3139685,0.6140864,0.06406406,0.3070432};
55     float temp[4]={0};
56     printf("IIR filter impulse response:\n");
57     for(t=0;t<20;t++) {
58         w=dcp_iirx(t?0:1,temp,coeff,2);
59         printf("y[%2d]=%g\n",t,w);
60     }
61 }
62
63 static void dcp_test3() {
64     double x[2]={2,3};
65     double y[2]={5,7};
66     dcp_butterfly_radix2(x,y);
67     printf("Radix-2 butterfly of (2+3j,5+7j)=(%g%+gj,%g%+gj)\n",x[0],x[1],y[0],y[1]);
68 }
69
70 static void dcp_test4() {
71     double x[2]={2,3};
72     double y[2]={5,7};
73     double t[2]={1.5,2.5};
74     dcp_butterfly_radix2_twiddle_dif(x,y,t);
75     printf("Radix-2 DIF butterfly of (2+3j,5+7j) with twiddle factor
(1.5+2.5j)=(%g%+gj,%g%+gj)\n",x[0],x[1],y[0],y[1]);
76 }
77
78 static void dcp_test5() {
79     double x[2]={2,3};
80     double y[2]={5,7};
81     double t[2]={1.5,2.5};
82     dcp_butterfly_radix2_twiddle_dit(x,y,t);
83     printf("Radix-2 DIT butterfly of (2+3j,5+7j) with twiddle factor
(1.5+2.5j)=(%g%+gj,%g%+gj)\n",x[0],x[1],y[0],y[1]);
84 }
85
86 static void dcp_test6() {
87     double w[2]={2,3};
88     double x[2]={5,7};
89     double y[2]={11,17};
90     double z[2]={41,43};
91     dcp_butterfly_radix4(w,x,y,z);
92     printf("Radix-4 butterfly of (2+3j,5+7j,11+17j,41+43j)=(%g%+gj,%g%+gj,%g%+gj,%g%+gj)\n"
,w[0],w[1],x[0],x[1],y[0],y[1],z[0],z[1]);
93 }
94
95 int main() {
96     stdio_init_all();
97
98     printf("Hello, DCP!\n");
99
100    dcp_test0();

```

```

101     dcp_test1();
102     dcp_test2();
103     dcp_test3();
104     dcp_test4();
105     dcp_test5();
106     dcp_test6();
107
108     return 0;
109 }
```

There are also further examples in the `dcp/` directory in the [Pico Examples](#) repository.

3.6.2.9. IEEE 754 compliance

The canned instruction sequences provide IEEE-compliant operations with the exception that denormals are flushed to zero on input and output. Zeroes, NaNs and infinities are correctly handled. Rounding is to nearest, even on tie.

Faster versions of division and square root operations, named `ddiv_fast` and `dsqrt_fast` respectively, are available. These do not always give correctly rounded results but do have a guaranteed error before rounding of less than 0.5ulp ('units in last place'), which in particular means that if there is an exact representation of the result then that is what is returned.

3.6.2.10. Benchmarks

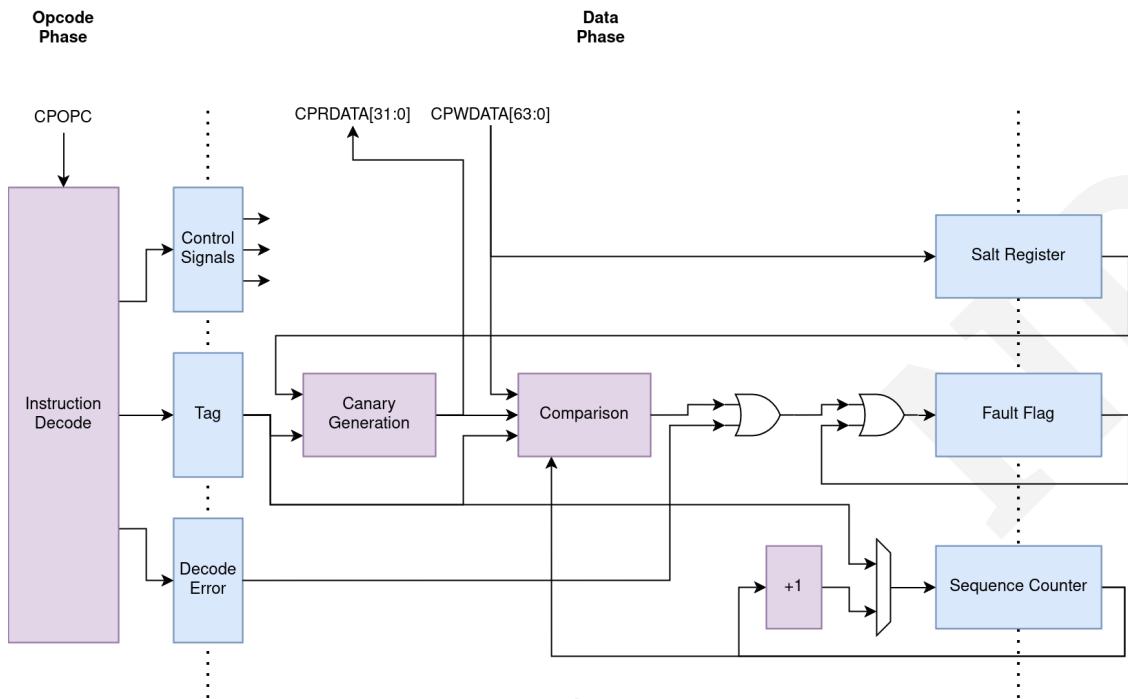
[Table 109](#) gives cycle counts for various floating-point operations using the accelerator with inlined code, compared to some typical ranges of benchmarks for (a) fully-fledged hardware double-precision FPUs; and (b) pure software implementations.

Table 109. Cycle counts for floating-point operations using the accelerator

Operation	Using coprocessor	Full hardware (latency)	Software only
<code>dadd</code>	6	2-6	70-90
<code>dsub</code>	6	2-6	70-90
<code>dmul</code>	17	3-7	75-90
<code>ddiv</code>	51	13-60	135-600
<code>ddiv_fast</code>	32		
<code>dsqrt</code>	49	15-62	130-650
<code>dsqrt_fast</code>	38		
<code>dcmp</code>	4		
<code>declassify</code>	2		
<code>integer to/from double</code>	5		

3.6.3. Redundancy Coprocessor (RCP)

Figure 12. The redundancy coprocessor implements hardware-checked assertions, to aid control flow and data flow integrity checking. Its two-phase pipeline is closely coupled to the Cortex-M33 pipeline. A 64-bit salt register holds a once-per-boot random number, which is used to generate and validate stack canary values and generate pseudorandom delay sequences on RCP instructions. Other comparison functions provide more general hardware-checked assertion support.



The redundancy coprocessor (RCP) is used in the RP2350 bootrom to provide hardware-assisted mitigation against fault injection and return-oriented programming attacks. This includes the following instructions:

- generate and validate stack canary values based on a per-boot random seed
- assert that certain points in the program are executed in the correct order without missing steps
- validate booleans stored as one of two valid bit patterns in a 32-bit word
- validate 32-bit integers stored redundantly in two words with an XOR parity mask
- halt the processor upon reaching a software-detected panic condition

Section 3.6.3.7 lists the RCP instruction set in full. RCP instruction encodings contain a parity bit; executing an invalid instruction or an instruction with bad parity triggers an RCP fault.

Each Cortex-M33 processor is equipped with a single RCP instance, mapped as coprocessor number 7 in the coprocessor opcode space. The two RCP instances are linked: an RCP fault on one core immediately triggers a fault on the other. RCP faults have two steps:

1. The non-maskable interrupt (NMI) is asserted. It remains asserted until a warm reset of the processor.
2. Any further RCP instructions stall the coprocessor port until a warm reset of the processor. This stall cannot be interrupted, as the processor is already in the NMI state.

In the RP2350 bootrom, implements the NMI and HardFault vectors with an `rcp_panic` instruction. This instruction unconditionally stalls the coprocessor port. This prevents the processor from retiring any more instructions until either a debugger connects to reset the processors, or the processors reset through some other mechanism (such as the system watchdog timer). The processor quickly reaches a quiescent state that reduces vulnerability to further fault injection (deliberate or otherwise).

Each core's RCP has a 64-bit seed value (Section 3.6.3.1). The RCP uses this value to generate stack canary values and to add short pseudorandom delays to RCP instructions. Both RCP instances are seeded by core 0 during the early boot path in the bootrom using the system true-random number generator (Section 12.12). Running any RCP instruction before providing a salt value triggers an RCP fault. The use of random data in stack canary values makes it difficult to reuse return-oriented-programming stack payloads across multiple boots.

Figure 12 gives a dataflow-level overview of the RCP hardware. The RCP is structured as a two-phase pipeline which overlays the Cortex-M33 execution pipeline. It exchanges data with the core via a 64-bit incoming bus (CPWDATA) and a 32-bit outgoing bus (CPRDATA). The Cortex-M33 can issue two register reads to the coprocessor in one cycle through

the CPWDATA bus. The RCP leverages this throughput for some of its assertion instructions, such as `rcp_iequal`, which raises a fault when two Arm registers do not contain the same 32-bit value.

The 8-bit **tag** value in [Figure 12](#) is an 8-bit instruction immediate value encoded by the instruction `CRn` and `CRm` fields. These 8-bit values are used to uniquely identify functions for canary value generation so that stack frames are not interchangeable between functions. They also provide 8-bit counter values for `rcp_count_set` and `rcp_count_check` instructions. Encoding the tags using the `CRn` and `CRm` fields makes RCP instruction sequences more compact, as it obviates additional instructions to materialise these small constants in registers and pass them through CPWDATA. This also makes the tag values less vulnerable to glitching, because the instruction opcode fields are available earlier in the cycle than the register values passed on CPWDATA.

RCP instructions may also execute in the Non-secure state, with certain differences to prevent Non-secure code from triggering RCP faults or observing the value of the salt register. This supports Non-secure software executing shared ROM routines which contain RCP instructions, but does not allow probing of the RCP's internal state from a Non-secure context. [Section 3.6.3.2](#) gives further details and rationale for Non-secure execution support.

Certain details are elided from [Figure 12](#) for clarity, such as the delay counter used for pseudorandom instruction delays, and the logic for suppressing faults under Non-secure execution. This behaviour is described in full in the following sections.

3.6.3.1. Salt Register

Each RCP instance is provisioned with a 64-bit **salt register**, which provides a seed for stack canary values and random instruction delays. This is expected to be initialised with a random value early in the boot process: the RP2350 bootrom uses the true random number generator to generate the salt values.

Initially the salt register is in the **invalid** state. This state only allows the following operations:

- Checking the valid state of the salt register, via `rcp_canary_status`
- Writing a salt via `rcp_salt_core0` or `rcp_salt_core1`, which writes a 64-bit value to that core's salt register, and changes its state to **valid**

When the salt register is in the invalid state, *executing any RCP instruction other than those listed above unconditionally triggers an RCP fault*. This makes it difficult to skip RCP initialisation via fault injection, because the RP2350 bootrom contains a high density of RCP instructions.

Similarly, attempting to write to an already-valid RCP salt register triggers an RCP fault. There is no reason to initialise the RCP salt register twice, so this case is detected as an anomaly that indicates loss of control flow integrity.

Core 0's coprocessor port writes the salt registers for both cores' RCP instances to simplify multicore interactions during early boot. In the RP2350 bootrom, core 1's first steps lock down its MPU execute permissions to a small region of the ROM containing its wait-for-launch code, and then poll for its RCP salt to become valid once core 0 has cleared boot memory, performed some minimal hardware setup, and generated the RCP salts.

When core 0 is switched to RISC-V architecture and core 1 is Arm, the core 1 salt register is forcibly marked as valid to permit core 1 to execute the ROM. This has no impact on secure boot because RISC-V cores are only enabled when secure boot is disabled; the ability to set core 0 to RISC-V already implies subversion of secure boot.

3.6.3.2. Access from Non-secure

Setting bit 7 of the Cortex-M33 `NSACR` register permits Non-secure code to set bit 7 of `CPACR_NS`, which in turn enables Non-secure access to the RCP. Non-secure RCP access is useful for executing shared Secure/Non-secure routines which contain RCP instructions. For example, the `memcpy` implementation in the RP2350 bootrom is shared by Secure code in the main boot path, and Non-secure code such as the USB bootloader.

Since an RCP fault is fatal for all software running on the system, Non-secure must not be able to trigger RCP faults at will. Similarly, if Non-secure code were able to read out the RCP salt register, it would make it easier to engineer stack payloads which can control Secure execution without triggering RCP faults. Therefore the RCP handles Non-secure accesses differently from Secure:

- Masks read data to all-zeroes
- Ignores write data: any instruction which would generate a data-dependent RCP fault becomes a no-op
- Reports coprocessor errors instead of RCP faults for invalid instructions, which the processor maps to the Non-secure UNDEFINSTR UsageFault
- Skips the pseudorandom instruction delay: all RCP instructions execute in one cycle, assuming the Cortex-M33 is able to issue them at one instruction per cycle

The lack of pseudorandom instruction delays makes it more difficult for Non-secure code to extract the seed value used to add delays to Secure execution of RCP instructions.

3.6.3.3. Instruction Validation

The RCP applies the following rules to all coprocessor instructions which target coprocessor 7:

- The number of 1 bits in the `Opc1` field, plus the instruction parity bit, must be an even number.
 - For `mcr`, `mrc` and `cdp` instructions, the parity bit must be encoded by bit 0 of the `Opc2` field.
 - For `mcrr`, the parity bit must be encoded by bit 3 of the `CRm` field.
- The instruction must not be an `mrrc` (64-bit coprocessor-to-core)
- For `mcr` instructions (32-bit core-to-coprocessor):
 - The `Opc1` field must be in the range 0 through 6.
 - If there is no 8-bit tag (i.e. any other than `rcp_canary_check`, `rcp_count_check`, `rcp_count_set`), the `CRn` and `CRm` opcode fields must be all-zeroes.
- For `mrc` instructions (32-bit coprocessor-to-core):
 - The `Opc1` field must be in the range 0 through 2.
 - For instructions other than `rcp_canary_get` and `rcp_canary_check`, the `CRn` and `CRm` opcode fields must be all-zeroes.
- For `mcrr` instructions (64-bit core-to-coprocessor):
 - The `Opc1` field must be in the range 0 through 8.
 - For `rcp_salt_core*` instructions, the `CRm` field must be 0 or 1 (referred to as `rcp_salt_core0` and `rcp_salt_core1` respectively).
 - For all other `mcrr` instructions, the `CRm` field must be 0.

The terms `Opc1`, `Opc2`, `CRm` and `CRn` in the description above refer to standard encoding fields in the Arm T32 instruction encoding for coprocessor instructions. See the [Armv8-M Architecture Reference Manual](#) for full details of the encoding and assembler syntax.

Any coprocessor instruction targeting coprocessor 7 that fails these validation rules will result in one of two outcomes, depending on the security domain in which the instruction is executed:

- Secure execution of an invalid instruction is an immediate, unconditional RCP fault. The RCP asserts the core's non-maskable interrupt signal, and any further RCP instructions stall the coprocessor port indefinitely. This continues until the core receives a warm reset. This also triggers RCP faults on other cores: for more information, see [Section 3.6.3.4](#).
- Non-secure execution of an invalid instruction returns an error on the opcode-phase coprocessor interface, which is interpreted as a Non-secure UNDEFINSTR UsageFault by the core. For a full description of this Armv8-M-specific fault, see the [Armv8-M Architecture Reference Manual](#).

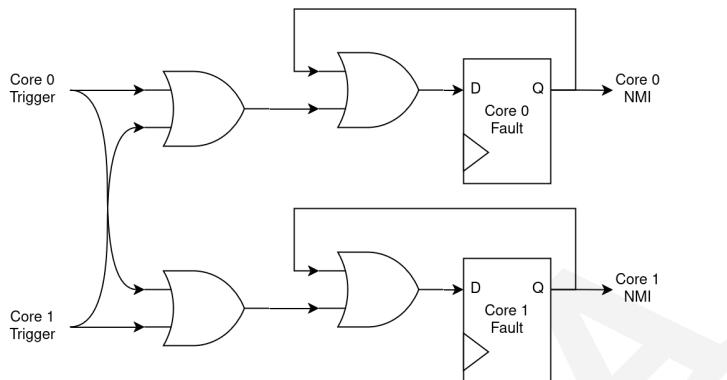
3.6.3.4. Cross-core Triggering

An RCP fault indicates that the integrity of the software environment is compromised. Though the fault may originate on a single processor, all processors which share the same trusted memory may behave unpredictably if they continue to execute, since:

- The physical condition which caused one processor to misexecute in a detectable way, such as low supply voltage, may cause other processors to misexecute in a manner which was not detected.
- The processor which triggered an RCP fault may already have corrupted shared, trusted memory contents in a way that interferes with the other processor's operation, (e.g. corrupting the other core's stack).

Therefore, an RCP fault on one core also triggers an RCP fault on other cores. Because RP2350 has two cores, an RCP fault on core 0 always triggers a fault on core 1, and an RCP fault on core 1 always triggers a fault on core 0.

Figure 13. Triggering an RCP fault on one core also triggers a fault on the other core. Triggers accumulate into a fault register, which remains set until the core resets. The NMI asserts when the fault register is set.



Each core locally ORs in the trigger signal from the other core. The outputs of the two OR gates on the left are logically equivalent, but the gates are kept local to the core to minimise delay routing the core's own fault trigger to its own fault register.

3.6.3.5. Stack Canary Values

Canaries are values written to the stack on function entry and validated on function exit, to assure that:

- The exit matches the entry (i.e. when leaving through the back door, you entered through the front door)
- The stack was not completely overwritten in the course of executing the function

This helps to mitigate two classes of attack:

- Fault injection: any physical fault condition which corrupts the program counter or causes a wild indirect branch is likely to cause the processor to execute a function epilogue which does not match the prologue. Any branch into the middle of a function is likely to eventually reach the epilogue.
- Return-oriented programming: deliberate stack corruption, for example by exploiting missing bounds checks on stack buffer operations. This can redirect control flow through a sequence of function tails which perform arbitrary operations. Random canary values make it difficult to craft such a stack payload.

Return-oriented programming mitigation is particularly important to account for in the bootrom because it exposes an API surface that is mapped at a known location at runtime (the bootrom is physically always mapped at `0x00000000`). This provides a well-known exploit surface similar to the C standard library.

The RCP supports canary value with two canary-specific instructions:

- `rcc_canary_get` generates a 32-bit value for an 8-bit tag as a function of the salt register
- `rcc_canary_check` validates a 32-bit value for an 8-bit tag and raises an RCP fault if the value does not match that produced by an `rcc_canary_get` for the same tag.

The 32-bit canary value is as follows:

- Bits 7:0: all-zero
- Bits 15:8: XOR of bits 7:0 of the salt with (AND of bits 31:24 of the salt with the 8-bit tag)
- Bits 23:16: XOR of bits 15:8 of the salt with (AND of bits 39:32 of the salt with the bitwise NOT of the 8-bit tag)
- Bits 31:24: XOR of bits 23:16 of the salt with the 8-bit tag

The following code demonstrates how you might calculate the 32-bit canary value in C:

```
uint32_t canary_value(uint64_t salt, uint8_t tag) {
    uint32_t tag_expanded =
        (uint32_t)tag |
        ((uint32_t)~tag << 8) |
        ((uint32_t)tag << 16);
    tag_expanded &= (0xff0000u | ((salt >> 24) & 0x00ffffu));
    uint32_t result24 = tag_expanded ^ salt;
    return result24 << 8;
}
```

This canary value is chosen such that:

- Different tags are guaranteed to yield different canary values
- For any two different tags, each is a function of at least one salt bit that the other is not a function of (so it is difficult to calculate canaries for different tags even if one value is known)
- Null-terminated string operations on the stack terminate before reading or writing a canary

Each function should use a different canary tag, to prevent a stack frame for one function being used to return through another function's epilogue. Avoid using canary values for purposes other than stack canaries.

The RP2350 bootrom uses 8-bit tags in the range `0x40` through `0xbf`. The remaining tags are free for use in user code.

3.6.3.6. Pseudorandom Instruction Delays

By default, all RCP instructions execute with a pseudorandom delay in the range of 0 to 127 cycles. These delays make it more difficult for an outside observer to precisely time a fault injection event with respect to an RCP instruction, or the critical code path it protects.

Setting bit 12 of the first halfword of an instruction disables the pseudorandom delay for that instruction only. The instruction executes in a single cycle, assuming the Cortex-M33 does not insert stall cycles due to other micro-architectural constraints. To set this bit, assemble the `*2` variant of any given coprocessor instruction (e.g. `mrc2` rather than `mrc`). In the Non-secure state, RCP instructions always execute without delay.

The RCP implements instruction execution delays by stalling the coprocessor opcode interface during the opcode phase (shown in the [Figure 12](#) pipeline diagram). The Cortex-M33 may choose to abandon a stalled coprocessor instruction due to an interrupt. When this happens, the delay counter continues counting down, waiting for the delay period to elapse. If the Cortex-M33 issues another RCP instruction whilst the delay counter is still running (either in the interrupt, or after returning to the interrupted RCP instruction), this instruction executes once the existing countdown completes. However, if the delay counter of an abandoned instruction has already expired before the next RCP instruction executes, the next instruction samples a pseudorandom delay count, and begins a new countdown.

The pseudorandom delay sequence is a function of bits 63:40 of the salt value. As such, the pattern of delays is unique per-boot, provided each boot writes a different 64-bit value to the salt register.

The pseudorandom number generator (PRNG) used for delays implements a number of small linear feedback shift registers (LFSRs) in bits 63:40 of the salt register, and returns a nonlinear function of the 24-bit state. The LFSR feedback functions on the 24-bit state are:

- Bits 23:20: 4-bit LFSR with taps `0xc`

- Bits 19:15: 5-bit LFSR with taps 0x14
- Bits 14:8: 7-bit LFSR with taps 0x60
- Bits 7:0: 8-bit LFSR with taps 0xb4

The LFSRs are implemented by shifting the XOR reduction of (state AND taps) into the LSB with each state update. When an LFSR's state is all-zeroes, a one bit is shifted into the LSB. The LFSR state advances each time a random number is generated: this happens when executing an instruction with a pseudorandom delay, or when executing a `rcp_random_byte` instruction.

Each bit of the pseudorandom output is the XOR of six bits of the 24-bit state, XORed with the majority-3 vote of three other bits of the state:

Output Bit	XOR Taps							Majority-3 Taps		
7	7	17	6	16	13	8	9	12	21	
6	14	21	19	6	16	13	4	14	6	
5	7	5	2	18	11	1	18	14	7	
4	4	19	17	0	18	7	18	11	3	
3	23	12	7	16	14	5	17	3	15	
2	15	13	20	21	8	12	7	22	9	
1	4	16	11	18	9	6	14	21	16	
0	11	3	4	19	10	14	1	2	9	

Bits 6:0 of this function are used for pseudorandom instruction delays, producing delays in the range of 0 to 127 cycles. The delay is applied in addition to the one-cycle base cost of executing a coprocessor instruction. The full 8-bit result is available through the `rcp_random_byte` instruction.

This is a simple pseudorandom number generator which makes it difficult to recover the initial 24-bit state from a small number of observations. It accomplishes this by making the observation size much smaller than the state size and using a non-linear combination function for the output. It has a number of statistical aberrations which make it unsuitable for general random number generation (not to mention its small state size). For high-quality random number generation, either use the system true-random number generator (TRNG) directly, or use a high-quality software PRNG with a large state seeded from the TRNG.

Note that the 24 MSBs of the salt value used to seed the delay PRNG do not overlap with the 40 LSBs used to generate stack canary values. Therefore measuring the random delays externally provides no information on the canary values.

3.6.3.7. Instruction Listing

The Cortex-M33 processors access the RCP using `mcr`, `mcr2`, `mrc`, and `cdp` instructions. The [Armv8-M Architecture Reference Manual](#) describes the intricacies of these instructions in relation to the processor's architectural state, but from the coprocessor's point of view:

- `mcr` writes a 32-bit value to the coprocessor from a single Arm integer register
- `mcr2` writes a 64-bit value to the coprocessor from a pair of Arm integer registers
- `mrc` reads a 32-bit value from the coprocessor, writing to either a single Arm integer register or to the processor status flags
- `cdp` performs some internal coprocessor operation without exchanging data with the processor

For each `mcr`, `mcr2`, `mrc` and `cdp` instruction, the RCP also accepts the matching `mcr2`, `mcr2`, `mrc2`, and `cdp2` opcode variant. These opcodes differ only in bit 12. The plain versions have a pseudorandom delay of up to 127 cycles on their execution, whereas the 2-suffixed versions have no such delay.

Most RCP instructions are in the form of hardware-checked assertions. The phrase "asserts that" in the following instruction listings means that, if some asserted condition is not true, the coprocessor raises an RCP fault.

3.6.3.7.1. Initialisation

`rcp_salt_core0`

Asserts that the core 0 salt register is currently invalid. Writes a 64-bit value, and mark it as valid.

Opcode:

```
mcrr p7, #8, Rt, Rt2, c0
```

`Rt` is the 32 LSBs of the salt, `Rt2` is the 32 MSBs.

`rcp_salt_core1`

Asserts that the core 1 salt register is currently invalid. Writes a 64-bit value, and mark it as valid.

Opcode:

```
mcrr p7, #8, Rt, Rt2, c1
```

`rcp_canary_status`

Returns a true or false bit pattern (`0xa500a500` or `0x00c300c3` respectively) that indicates whether the salt register for this core has been initialised.

Opcode:

```
mrc p7, #1, Rt, c0, c0, #0
```

Invoking with `Rt = 0xf` sets the Arm `N` and `C` flags if and only if the salt register is valid.

If the salt has not been initialised, any operation other than initialising the salt or checking the canary status triggers an RCP fault.

This opcode is used on core 0 to skip the RCP initialisation sequence if the bootrom has been re-entered without a reset under debugger control, and on core 1 to wait for its RCP salt to be initialised.

3.6.3.7.2. Canary

`rcp_canary_get`

Gets a 32-bit canary value as a function of the salt register and the 8-bit tag encoded by two 4-bit coprocessor register numbers `CRn` and `CRm`. `CRn` contains the four MSBs, `CRm` the four LSBs.

Opcode:

```
mrc p7, #0, Rt, CRn, CRm, #1
```

Section 3.6.3.5 specifies the 32-bit value returned by this instruction, but you should treat this as an opaque value to be consumed by `rcp_canary_check`.

rcp_canary_check

Asserts that a value matches the result of an `rcp_canary_get` with the same 8-bit tag. The tag is encoded by two 4-bit coprocessor register numbers, `CRn` and `CRm`. `CRn` contains the four MSBs, `CRm` the four LSBs.

Opcode:

```
mcr p7, #0, Rt, CRn, CRm, #1
```

3.6.3.7.3. Boolean Validation

The RCP defines `0xa500a500` as the `true` value for 32-bit booleans, and `0x00c300c3` as the `false` value. All other bit patterns are poison, and trigger an RCP fault when consumed by any RCP boolean instructions. These values are chosen as they are valid immediates in Armv8-M Main.

This provides limited runtime type checking to ensure that boolean values are used in boolean contexts. The RP2350 bootrom occasionally uses redundant operations to generate booleans in a way that results in an invalid bit pattern if the two redundant operations do not return the same value, such as when checking boot flags in OTP.

rcp_bvalid

Asserts that `Rt` is a valid boolean (`0xa500a500` or `0x00c300c3`).

Opcode:

```
mcr p7, #1, Rt, c0, c0, #0
```

rcp_btrue

Asserts that `Rt` is `true` (`0xa500a500`).

Opcode:

```
mcr p7, #2, Rt, c0, c0, #0
```

rcp_bfalse

Asserts that `Rt` is `false` (`0x00c300c3`).

Opcode:

```
mcr p7, #3, Rt, c0, c0, #1
```

rcp_b2valid

Asserts that `Rt` and `Rt2` are both valid booleans.

Opcode:

```
mcrr p7, #0, Rt, Rt2, c8
```

rcp_b2and

Asserts that **Rt** and **Rt2** are both **true**.

Opcode:

```
mcrr p7, #1, Rt, Rt2, c0
```

rcp_b2or

Asserts that both **Rt** and **Rt2** are valid, and at least one is true.

```
mcrr p7, #2, Rt, Rt2, c0
```

rcp_bxorvalid

Asserts that **Rt** XOR **Rt2** is a valid boolean. The XOR mask is generally a fixed bit pattern used to validate the origin of the boolean, such as a return value from a critical function.

Opcode:

```
mcrr p7, #3, Rt, Rt2, c8
```

rcp_bxortrue

Asserts that **Rt** XOR **Rt2** is **true**.

Opcode:

```
mcrr p7, #4, Rt, Rt2, c0
```

rcp_bxorfalse

Asserts that **Rt** XOR **Rt2** is **false**.

Opcode:

```
mcrr p7, #5, Rt, Rt2, c8
```

3.6.3.7.4. Integer Validation**rcp_ivalid**

Asserts that **Rt** XOR **Rt2** is equal to **0x96009600**. This is used to validate 32-bit integers stored redundantly in two memory words. The XOR difference provides assurance that two parallel chains of integer operations have not mixed.

Opcode:

```
mcrr p7, #6, Rt, Rt2, c8
```

rcp_iequal

Asserts that **Rt** is equal to **Rt2**. Useful for general software assertions that are worth checking in hardware.

Opcode:

```
mcrr p7, #7, Rt, Rt2, c0
```

3.6.3.7.5. Random

rcp_random_byte

Returns a random 8-bit value generated from the upper 24 bits of the 64-bit salt value. Bits **31:8** of the result are all-zero.

Opcode:

```
mrc p7, #2, Rt, c0, c0, #0
```

This is the same PRNG used for random delay values. It is mainly exposed for debugging purposes, and should not be used for general software RNG purposes because the 24-bit state space is inadequate for scenarios where the quality and predictability of the random numbers is important.

This instruction never has an execution delay. Once the Cortex-M33 issues the coprocessor access, it always completes in one cycle.

3.6.3.7.6. Sequence Count Checking

These instructions are used to assert that a sequence of operations happens in the correct order. The count is initialised to an 8-bit value at the beginning of such a sequence, then repeatedly checked, incrementing with each check. If the 8-bit check value does not match the current counter value, the coprocessor raises an RCP fault.

rcp_count_set

Writes an 8-bit count value to the RCP sequence counter. Encodes the 8-bit value using two 4-bit coprocessor numbers: **CRn** provides the MSBs, **CRm** the LSBs.

Opcode:

```
mcr p7, #4, r0, CRn, CRm, #0
```

rcp_count_check

Asserts that an 8-bit count value matches the current value of the RCP sequence counter. Increments the counter by one, wrapping back to **0x00** after reaching **0xff**. Encodes the 8-bit count value using two 4-bit coprocessor numbers: **CRn** provides the MSBs, **CRm** the LSBs.

Opcode:

```
mcr p7, #5, r0, CRn, CRm, #1
```

3.6.3.7.7. Panic

`rcp_panic`

Stalls the coprocessor port forever. If the processor abandons the coprocessor access, asserts NMI and continues stalling the coprocessor port. Also immediately raises an RCP fault on other cores.

Opcode:

```
cdp p7, #0, c0, c0, c0, #1
```

Software executes an `rcp_panic` instruction when it detects a condition that makes it unsafe to continue executing the current program. The RCP responds by stalling the processor's CDP access forever, which should cause the processor to stop fetching and executing instructions.

The processor is allowed to abandon a stalled coprocessor instruction when interrupted, which may cause it to continue executing in an unsafe state. The RCP responds to an abandoned transfer by asserting the non-maskable interrupt, pre-empting the interrupt handler that caused the coprocessor access to be abandoned. This should swiftly encounter another RCP instruction and once again stall the processor, this time without allowing interruption.

Panic is specified in this way, instead of gating the processor clock, so the debugger can still attach cleanly to the processor after a panic.

3.6.4. Floating Point Unit

The Cortex-M33 cores on RP2350 are configured with the standard Arm single-precision floating point unit (FPU). Coprocessor ports 10 and 11 access the FPU.

The Arm floating point extension is documented in the [Armv8-M Architecture Reference Manual](#).

Applications built with the SDK use the FPU automatically by default. For example, calculations with the `float` data type in C automatically use the standard FPU, while calculations with the `double` data type automatically use the RP2350 double-precision coprocessor ([Section 3.6.2](#)).

3.7. Cortex-M33 Processor

Arm Documentation

Excerpted from the [Cortex-M33 Technical Reference Manual](#). Used with permission.

The Arm Cortex-M33 processor is a low gate count, highly energy-efficient processor intended for microcontroller and embedded applications. The processor is based on the Armv8-M architecture and is primarily for use in environments where security is an important consideration.

NOTE

Full details of the Arm Cortex-M33 processor can be found in the [Technical Reference Manual](#).

3.7.1. Features

The Arm Cortex-M33 processor provides the following features and benefits:

- An in-order issue pipeline
- Thumb-2 technology; for more information, see the [Armv8-M Architecture Reference Manual](#)
- Little-endian data accesses
- A Nested Vectored Interrupt Controller (NVIC) closely integrated with the processor
- A Floating Point Unit (FPU) supporting single-precision arithmetic
- Support for exception-continuable instructions, such as LDM, LDMDB, STM, STMDB, PUSH, POP, VLDM, VSTM, VPUSH, and VPOP
- A low-cost debug solution that provides the ability to implement:
 - breakpoints
 - watchpoints
 - tracing
 - system profiling
 - Support for `printf()` style debugging through an Instrumentation Trace Macrocell (ITM)
- Support for the Embedded Trace Macrocell (ETM) instruction trace option. For more information, see the [Arm CoreSight ETM-M33 Technical Reference Manual](#)
- A coprocessor interface for external hardware accelerators
- Low-power features including architectural clock gating, sleep mode, and a power-aware system with Wake-up Interrupt Controller (WIC)
- A memory system that includes memory protection and security attribution

3.7.2. Configuration

Each Arm Cortex-M33 processor in RP2350 is configured with the following features:

- **FPU:** Single precision FPU
- **DSP:** DSP extension
- **SECEXT:** Security extensions
- **CPIF:** coprocessor interface
- **MPU_NS:** 8 non-secure MPU regions
- **MPU_S:** 8 secure MPU regions
- **SAU:** 8 SAU regions
- **IRQ:** 52 external interrupts
- **IRQLVL:** 4 exception priority bits
- **DBGLVL:** Full debug set: 4 watchpoint, 8 breakpoint comparators, debug monitor

- ITM: [DWT](#) and ITM trace
- [ETM](#): ETM trace
- [MTB](#): no MTB trace
- [WIC](#): Wake up interrupt controller
- [WICLINES](#): 55: All external interrupts and 3 internal events: NMI, RVEX, Debug
- [CTI](#): Cross trigger interface
- RAR: reset all registers on power up
- [UNCROSS_I_D](#): Modify internal address map
- [SBIST](#): no SBIST features
- CDE modules not used
- [CDERTLID](#): RTL ID for system with multi Cortex-M33: 16

Architectural clock gating allows the processor core to support SLEEP and DEEPSLEEP power states by disabling the clock to parts of the processor core. Power gating is not supported.

Each M33 core has its own interrupt controller which can individually mask out interrupt sources as required. The same interrupts route to both M33 cores.

The processor supports the following interfaces:

- Code AHB (C-AHB) interface
- System AHB (S-AHB) interface
- External PPB (EPPB) APB interface
- Debug AHB (D-AHB) interface

The processor implements the following optional interfaces:

- Arm TrustZone technology, using the Armv8-M Security Extension supporting Secure and Non-secure states
- Memory Protection Units (MPUs), which you can configure to protect regions of memory
- Floating-point arithmetic functionality with support for single precision arithmetic
- Support for ETM and MTB trace

3.7.2.1. Modifications by Raspberry Pi

3.7.2.1.1. [UNCROSS_I_D](#)

The original Cortex-M33 processor design routes the following operations to either the Code or System port:

- instruction fetch
- load/stores
- debugger accesses

Accesses below address [0x20000000](#) route to the Code port. All other accesses route to the System port.

This routing strategy makes contention possible on both the internal bus and the main system AHB5 crossbar. The *Cortex-M33 Technical Reference Manual* describes this strategy in detail.

In RP2350, Raspberry Pi modified the Cortex-M33 bus matrix to:

- route all instruction fetch operations to the Code port

- route all load/stores and debugger accesses to the System port

This eliminates most internal conflicts and improves performance in certain software use cases, e.g. when allocating both code and data from a single unified SRAM pool.

In [Section 3.7.2](#), we refer to this feature as [UNCROSS_I_D](#).

There are no other modifications to the Cortex-M33 processor.

3.7.2.2. Interfaces

The processor has various external interfaces:

Code and System AHB interfaces

Harvard AHB bus architecture supporting exclusive transactions and security state.

System AHB interface

The System AHB (S-AHB) interface is used for any instruction fetch and data access to the memory-mapped SRAM, Peripheral, External RAM and External device, or Vendor_SYS regions of the Armv8-M memory map.

Code AHB interface

The Code AHB (C-AHB) interface is used for any instruction fetch and data access to the Code region of the Armv8-M memory map.

External Private Peripheral Bus

The External PPB (EPPB) APB interface enables access to CoreSight-compatible debug and trace components in a system connected to the processor.

Secure attribution interface

The processor has an interface that connects to an external Implementation Defined Attribution Unit (IDAU), which enables your system to set security attributes based on address.

ATB interfaces

The ATB interfaces output trace data for debugging. The ATB interfaces are compatible with the CoreSight architecture. See the Arm CoreSight Architecture Specification v2.0 for more information. The instruction ATB interface is used by the ETM, and the instrumentation ATB interface is used by the Instrumentation Trace Macrocell (ITM).

Micro Trace Buffer interfaces

The Micro Trace Buffer (MTB) AHB slave interface and SRAM interface are for the CoreSight Micro Trace Buffer.

Coprocessor interface

The coprocessor interface is designed for closely coupled external accelerator hardware.

Debug AHB interface

The Debug AHB (D-AHB) slave interface allows a debugger access to registers, memory, and peripherals. The D-AHB interface provides debug access to the processor and the complete memory map.

Cross Trigger Interface

The processor includes a Cross Trigger Interface (CTI) Unit that has an interface that is suitable for connection to external CoreSight components using a Cross Trigger Matrix (CTM).

Power control interface

The processor supports a number of internal power domains which can be enabled and disabled using Q-channel interfaces connected to a Power Management Unit (PMU) in the system.

3.7.2.3. Security attribution and memory protection

The Cortex-M33 processor supports the Armv8-M Protected Memory System Architecture (PMSA) that provides programmable support for [memory protection](#) using a number of software controllable regions. RP2350 supports 8 programmable regions.

PMSA allows privileged software to assign access permissions to a memory region. When unprivileged software attempts to access the region, a fault exception is triggered. PMSA includes fault status registers that allow an exception handler to determine the source of the fault, apply corrective action, and notify the system. This reduces the potential impact of incorrectly-written application code.

The Cortex-M33 processor also includes support for defining memory regions as Secure or Non-secure, as defined in the Armv8-M Security Extension. This protects memory regions from accesses with an inappropriate level of security.

3.7.2.4. Floating-Point Unit (FPU)

The FPU provides:

- Instructions for single-precision (C programming language float type) data-processing operations
- Instructions for double-precision (C programming language double type) load and store operations
- Combined multiply-add instructions for increased precision (Fused MAC)
- Hardware support for conversion, addition, subtraction, multiplication, accumulate, division, and square-root
- Hardware support for denormals and all IEEE Standard 754-2008 rounding modes
- Thirty-two 32-bit single-precision registers or sixteen 64-bit double-precision registers
- Lazy floating-point context save

3.7.2.4.1. Lazy floating-point context save

This FPU function delays automated stacking of floating-point state until the ISR attempts to execute a floating-point instruction. This reduces the latency to enter the ISR and removes floating-point context save for ISRs that do not use floating-point.

3.7.2.5. NVIC

The Nested Vectored Interrupt Controller NVIC prioritizes external interrupt signals. Software can set the priority of each interrupt. The NVIC and the Cortex-M33 processor core are closely coupled, providing low latency interrupt processing and efficient processing of late arriving interrupts.

NOTE

"Nested" refers to the fact that interrupts can themselves be interrupted, by higher-priority interrupts. "Vectored" refers to the hardware dispatching each interrupt to a distinct handler routine specified by a vector table. For more details about nesting and vectoring behaviour, see the [Armv8-M Architecture Reference Manual](#).

All NVIC registers are only accessible using word transfers. Any attempt to read or write a halfword or byte individually is unpredictable.

NVIC registers are always little-endian.

The Nested Vectored Interrupt Controller (NVIC) is closely integrated with the core to achieve low-latency interrupt processing.

Functions of the NVIC include:

- External interrupts, configurable from 1 to 480 using a contiguous or non-contiguous mapping. This is configured at implementation.
- Configurable levels of interrupt priority from 8 to 256. This is configured at implementation.
- Dynamic reprioritisation of interrupts.
- Priority grouping. This enables selection of pre-empting interrupt levels and non-pre-empting interrupt levels.
- Support for tail-chaining and late arrival of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
- Support for the Armv8-M Security Extension. Secure interrupts can be prioritized above any Non-secure interrupt.

3.7.2.6. Cross Trigger Interface Unit (CTI)

The CTI enables the debug logic, MTB, and ETM to interact with each other and with other CoreSight™ components.

3.7.2.7. ETM

The ETM provides instruction-only capabilities.

3.7.2.8. MTB

The MTB provides a simple low-cost execution trace solution for the Cortex-M33 processor.

Trace is written to an SRAM interface, and can be extracted using a dedicated AHB slave interface (M-AHB) on the processor. The MTB can be controlled by memory-mapped registers in the PPB region or by events generated by the DWT or through the CTI.

See the Arm [CoreSight MTB-M33 Technical Reference Manual](#) for more information.

3.7.2.9. Debug and Trace

Debug and trace components include a configurable Breakpoint Unit (BPU) used to implement breakpoints and a configurable Data Watchpoint and Trace (DWT) unit used to implement watchpoints, data tracing, and system profiling. Other debug and trace components include:

- ITM for support of `printf()` style debugging, using instrumentation trace
- Interfaces suitable for:
 - Passing on-chip data through a Trace Port Interface Unit (TPIU) to a Trace Port Analyzer (TPA), including Serial Wire Output (SWO) mode
 - A ROM table to allow debuggers to determine which components are implemented in the Cortex-M33 processor
 - Debugger access to all memory and registers in the system, including access to memory-mapped devices, access to internal core registers when the core is halted, and access to debug control registers even when reset is asserted

3.7.3. Compliance

The processor complies with, or implements, the relevant Arm architectural standards and protocols, and relevant external standards.

3.7.3.1. Arm architecture

The processor is compliant with the following:

- Armv8-M Main Extension
- Armv8-M Security Extension
- Armv8-M Protected Memory System Architecture (PMSA)
- Armv8-M Floating-point Extension
- Armv8-M Digital Signal Processing (DSP) Extension
- Armv8-M Debug Extension
- Armv8-M Flash Patch Breakpoint (FPB) architecture version 2.0

3.7.3.2. Bus architecture

The processor provides external interfaces that comply with the AMBA 5 AHB5 protocol. The processor also implements interfaces for CoreSight and other debug components using the APB4 protocol and ATBv1.1 part of the AMBA 4 ATB protocol.

For more information, see the:

- [Arm AMBA 5 AHB Protocol Specification](#)
- [AMBA APB Protocol Version 2.0 Specification](#)
- [Arm AMBA 4 ATB Protocol Specification ATBv1.0 and ATBv1.1](#)

The processor also provides a Q-Channel interface. For more information, see the [AMBA Low Power Interface Specification](#).

3.7.3.3. Debug

The debug features of the processor implement the Arm Debug Interface Architecture. For more information, see the [Arm Debug Interface Architecture Specification, ADLv5.0 to ADLv5.2](#).

3.7.3.4. Embedded Trace Macrocell

The trace features of the processor implement the Arm Embedded Trace Macrocell (ETM) v4.2 architecture.

For more information, see the [Arm CoreSight ETM-M33 Technical Reference Manual](#).

3.7.3.5. Floating-Point Unit

The Cortex-M33 processor with FPU supports single-precision arithmetic as defined by the FPv5 architecture that is part of the Armv8-M architecture. The FPU provides floating-point computation functionality compliant with ANSI/IEEE Standard 754-2008, IEEE Standard for Binary Floating-Point Arithmetic.

The FPU supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions.

The FPU provides an extension register file containing 32 single-precision registers.

The registers can be viewed as:

- Thirty-two 32-bit single-word registers, **S0-S31**
- Sixteen 64-bit double-word registers, **D0-D15**
- A combination of registers from these views

3.7.3.5.1. FPU modes

The FPU provides full-compliance, flush-to-zero, and Default NaN modes of operation. In full-compliance mode, the FPU processes all operations according to the IEEE 754 standard in hardware.

Modes of operation are controlled using the Floating-Point Status and Control Register, **FPSCR**.

Setting the **FPSCR.FZ** bit enables Flush-to-Zero (FZ) mode. In FZ mode, the FPU treats all subnormal input operands of arithmetic operations as zeros. Exceptions that result from a zero operand are signalled appropriately. **VABS**, **VNEG**, and **VMOV** are not considered arithmetic operations and are not affected by FZ mode. When an operation yields a *tiny* result (as described in the IEEE 754 standard, where the destination precision is smaller in magnitude than the minimum normal value before rounding) FZ mode replaces the result with a zero.

The **FPSCR.IDC** bit indicates when an input flush occurs.

The **FPSCR.UFC** bit indicates when a result flush occurs.

Setting the **FPSCR.DN** bit enables Default NaN (DN) mode. In NaN mode, the result of any arithmetic data processing operation that involves an input NaN, or that generates a NaN result, returns the default NaN. All arithmetic operations except for **VABS**, **VNEG**, and **VMOV** ignore the fraction bits of an input NaN.

Setting neither the **FPSCR.DN** bit nor the **FPSCR.FZ** bit enables full-compliance mode. In full-compliance mode, FPv5 functionality is compliant with the IEEE 754 standard in hardware.

For more information about the FPU and **FPSCR**, see the [Armv8-M Architecture Reference Manual](#).

3.7.3.5.2. FPU Exceptions

The FPU sets the cumulative exception status flag in the **FPSCR** register as required for each instruction, in accordance with the FPv5 architecture. The FPU does not support exception traps.

The processor has six output pins. By default, they are disconnected. Each reflect the status of one of the cumulative exception flags:

FPIXC

Masked floating-point inexact exception.

FPUFC

Masked floating-point underflow exception.

FPOFC

Masked floating-point overflow exception.

FPDZC

Masked floating-point divide by zero exception.

FPIDC

Masked floating-point input denormal exception.

FPIOC

Invalid operation.

When a floating-point context is active, the stack frame extends to accommodate the floating-point registers. To reduce the additional interrupt latency associated with writing the larger stack frame on exception entry, the processor supports lazy stacking. This means that the processor reserves space on the stack for the FP state, but does not save that state information to the stack unless the processor executes an FPU instruction inside the exception handler.

The lazy save of the FP state is interruptible by a higher priority exception. The FP state saving operation starts over after that exception returns.

3.7.3.5.3. Low power FPU operation

If the FPU is in a separate power domain, the way the FPU domain powers down depends on whether the FPU domain includes state retention logic.

To power down the FPU:

- If FPU domain includes state retention logic, disable the FPU by clearing the `CPACR.CP10` and `CPACR.CP11` bitfields.
- If FPU domain does not include state retention logic, disable the FPU by clearing the `CPACR.CP10` and `CPACR.CP11` bitfields and set both the `CPPWR.SU10` and `CPPWR.SU11` bitfields to 1.

 **WARNING**

Setting the `CPPWR.SU10` and `CPPWR.SU11` bitfields indicates that FPU state can be lost.

3.7.4. Programmer's model

The Cortex-M33 programmer's model is an implementation of the Armv8-M Main Extension architecture.

For a complete description of the programmers model, refer to the [Armv8-M Architecture Reference Manual](#), which also contains the Armv8-M Thumb instructions. In addition, other options of the programmers model are described in the System Control, MPU, NVIC, FPU, Debug, DWT, ITM, and TPIU feature topics.

3.7.4.1. Modes of operation and execution

The Cortex-M33 processor supports Secure and Non-secure security states, Thread and Handler operating modes, and can run in either Thumb or Debug operating states. In addition, the processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode.

See the [Armv8-M Architecture Reference Manual](#) for more information about the modes of operation and execution.

3.7.4.1.1. Security states

With the Armv8-M Security Extension, the programmer's model includes two orthogonal security states: Secure state and Non-secure state. The processor always resets into Secure state. Each security state includes a set of independent operating modes and supports both privileged and unprivileged user access. Registers in the System Control Space are banked across Secure and Non-secure state, with a Non-secure register view available to Secure state at an aliased address.

3.7.4.1.2. Operating modes

For each security state, the processor can operate in *Thread* or *Handler* mode. The following conditions cause the processor to enter Thread or Handler mode:

- The processor enters Thread mode on reset, or as a result of an exception return to Thread mode. Privileged and Unprivileged code can run in Thread mode.
- The processor enters Handler mode as a result of an exception. In Handler mode, all code is privileged.

The processor can change security state on taking an exception, for example when a Secure exception is taken from Non-secure state, the Thread mode enters the Secure state Handler mode. The processor can also call Secure functions

from Non-secure state and Non-secure functions from Secure state. The Security Extension includes requirements for these calls to prevent Secure data from being accessed in Non-secure state.

3.7.4.1.3. Operating states

The processor can operate in Thumb or Debug state:

- Thumb state is the state of normal execution running 16-bit and 32-bit halfword-aligned Thumb instructions.
- Debug state is the state when the processor is in Halting debug.

3.7.4.1.4. Privileged access and unprivileged user access

Code can execute as privileged or unprivileged. Unprivileged execution limits resource access appropriate to the current security state. Privileged execution has access to all resources available to the security state. Handler mode is always privileged. Thread mode can be privileged or unprivileged.

3.7.4.2. Instruction set summary

The processor implements the following instruction from Armv8-M:

- All base instructions
- All instructions in the Main Extension
- All instructions in the Security Extension
- All instructions in the DSP Extension
- All single-precision instructions and double precision load/store instructions in the Floating-point Extension

For more information about Armv8-M instructions, see the [Armv8-M Architecture Reference Manual](#).

3.7.4.3. Memory model

The processor contains a bus matrix that arbitrates instruction fetches and memory accesses from the processor core between the external memory system and the internal System Control Space (SCS) and debug components.

Priority is usually given to the processor to keep debug accesses as non-intrusive as possible.

The system memory map is Armv8-M Main Extension compliant, and is common both to the debugger and processor accesses.

The default memory map provides user and privileged access to all regions except for the Private Peripheral Bus (PPB). The PPB space only allows privileged access.

The following table shows the default memory map. This is the memory map used when the included MPUs are disabled. The attributes and permissions of all regions, except that targeting the NVIC and debug components, can be modified using an implemented MPU.

Table 110. Default memory map

Address Range (inclusive)	Region	Interface
0x00000000 - 0x1FFFFFFF	Code	Instruction and data accesses.
0x20000000 - 0x3FFFFFFF	SRAM	Instruction and data accesses.
0x40000000 - 0x5FFFFFFF	Peripheral	Instruction and data accesses. Any attempt to execute instructions from the peripheral and external device region results in a MemManage fault.

Address Range (inclusive)	Region	Interface
0x60000000 - 0x9FFFFFFF	External RAM	Instruction and data accesses. Any attempt to execute instructions from the peripheral and external device region results in a MemManage fault.
0xA0000000 - 0xDFFFFFFF	External device	Instruction and data accesses. Any attempt to execute instructions from the peripheral and external device region results in a MemManage fault.
0xE0000000 - 0xE00FFFFF	PPB	Reserved for system control and debug. Cannot be used for exception vector tables. Data accesses are either performed internally or on EPPB. Accesses in the range 0xE0000000 - 0xE0043FFF are handled within the processor. Accesses in the range 0xE0044000 - 0xE00FFFFF appear as APB transactions on the EPPB interface of the processor. Any attempt to execute instructions from the region results in a MemManage fault.
0xE0100000 - 0xFFFFFFFF	Vendor_SYS	Partly reserved for future processor feature expansion. Any attempt to execute instructions from the region results in a MemManage fault.

The internal Secure Attribution Unit (SAU) determines the security level associated with an address. Some internal peripherals have memory-mapped registers in the PPB region which are banked between Secure and Non-secure state. When the processor is in Secure state, software can access both the Secure and Non-secure versions of these registers. The Non-secure versions are accessed using an aliased address.

For more information about the memory model, see the [Armv8-M Architecture Reference Manual](#).

3.7.4.3.1. Private Peripheral Bus (PPB)

The Private Peripheral Bus (PPB) memory region provides access to internal and external processor resources.

The internal PPB provides access to:

- The System Control Space (SCS), including the Memory Protection Unit (MPU), Secure Attribution Unit (SAU), and the Nested Vectored Interrupt Controller (NVIC).
- The Data Watchpoint and Trace (DWT) unit.
- The Breakpoint Unit (BPU).
- The Embedded Trace Macrocell (ETM).
- CoreSight Micro Trace Buffer (MTB).
- Cross Trigger Interface (CTI).
- The ROM table.

The external PPB (EPPB) provides access to implementation-specific external areas of the PPB memory map.

3.7.4.3.2. Unaligned accesses

The Cortex-M33 processor supports unaligned accesses. They are converted into two or more aligned AHB transactions on the C-AHB or S-AHB master ports on the processor.

Unaligned support is only available for load/store singles (LDR, LDRH, STR, STRH, TBH) to addresses in Normal memory. Load/store double and load/store multiple instructions already support word aligned accesses, but do not permit other unaligned accesses, and generate a fault if this is attempted. Unaligned accesses in Device memory are not permitted and fault. Unaligned accesses that cross memory map boundaries are architecturally **UNPREDICTABLE**.

NOTE

If `CCR.UNALIGN_TRP` for the current Security state is set, any unaligned accesses generate a fault.

3.7.4.4. Exclusive monitor

The Cortex-M33 processor implements a local exclusive monitor. The local monitor within the processor has been constructed so that it does not hold any physical address, but instead treats any store-exclusive access as matching the address of the previous load-exclusive. This means that the implemented exclusives reservation granule is the entire memory address range. For more information about semaphores and the local exclusive monitor, see the [Arm v8-M Architecture Reference Manual](#).

3.7.4.5. Processor core registers summary

The following table shows the processor core register set summary. Each of these registers is 32 bits wide. When the Arm v8-M Security Extension is included, some of the registers are banked. The Secure view of these registers is available when the Cortex-M33 processor is in Secure state and the Non-secure view when Cortex-M33 processor is in Non-secure state.

Table 111. Processor core register set summary

Name	Description
<code>R0-R12</code>	<code>R0-R12</code> are general-purpose registers for data operations.
<code>MSP (R13)</code>	The Stack Pointer (SP) is register <code>R13</code> . In Thread mode, the CONTROL register indicates the stack pointer to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP). There are two MSP registers in the Cortex-M33 processor: <code>MSP_NS</code> for the Non-secure state, and <code>MSP_S</code> for the Secure state. There are two PSP registers in the Cortex-M33 processor: <code>PSP_NS</code> for the Non-secure state, and <code>PSP_S</code> for the Secure state.
<code>PSP (R13)</code>	The Stack Pointer (SP) is register <code>R13</code> . In Thread mode, the CONTROL register indicates the stack pointer to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP). There are two MSP registers in the Cortex-M33 processor: <code>MSP_NS</code> for the Non-secure state, and <code>MSP_S</code> for the Secure state. There are two PSP registers in the Cortex-M33 processor: <code>PSP_NS</code> for the Non-secure state, and <code>PSP_S</code> for the Secure state.
<code>MSPLIM</code>	The stack limit registers limit the extent to which the MSP and PSP registers can descend respectively. There are two MSPLIM registers in the Cortex-M33 processor: <code>MSPLIM_NS</code> for the Non-secure state, and <code>MSPLIM_S</code> for the Secure state. There are two PSPLIM registers in the Cortex-M33 processor: <code>PSPLIM_NS</code> for the Non-secure state, and <code>PSPLIM_S</code> for the Secure state.
<code>PSPLIM</code>	The stack limit registers limit the extent to which the MSP and PSP registers can descend respectively. There are two MSPLIM registers in the Cortex-M33 processor: <code>MSPLIM_NS</code> for the Non-secure state, and <code>MSPLIM_S</code> for the Secure state. There are two PSPLIM registers in the Cortex-M33 processor: <code>PSPLIM_NS</code> for the Non-secure state, and <code>PSPLIM_S</code> for the Secure state.

Name	Description
LR (R14)	The Link Register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	The Program Counter (PC) is register R15. It contains the current program address.
PSR	The Program Status Register (PSR) combines the Application Program Status Register (APSR), Interrupt Program Status Register (IPSR), and Execution Program Status Register (EPSR). These registers provide different views of the PSR.
PRIMASK	The PRIMASK register prevents activation of exceptions with configurable priority. When the Armv8-M Security Extension is included, there are two PRIMASK registers in the Cortex-M33 processor: PRIMASK_NS for the Non-secure state and PRIMASK_S for the Secure state.
BASEPRI	The BASEPRI register defines the minimum priority for exception processing. There are two BASEPRI registers in the Cortex-M33 processor: BASEPRI_NS for the Non-secure state, and BASEPRI_S for the Secure state.
FAULTMASK	The FAULTMASK register prevents activation of all exceptions except for NON-MASKABLE INTERRUPT (NMI) and Secure HardFault. There are two FAULTMASK registers in the Cortex-M33 processor: FAULTMASK_NS for the Non-secure state, and FAULTMASK_S for the Secure state.
CONTROL	The CONTROL register controls the stack used, and optionally the privilege level, when the processor is in Thread mode. There are two CONTROL registers in the Cortex-M33 processor: CONTROL_NS for the Non-secure state and CONTROL_S for the Secure state.

3.7.4.6. Exceptions

Exceptions are handled and prioritized by the processor and the NVIC. In addition to architecturally defined behaviour, the processor implements advanced exception and interrupt handling that reduces interrupt latency and includes implementation defined behaviour.

The processor core and the Nested Vectored Interrupt Controller (NVIC) together prioritize and handle all exceptions. When handling exceptions:

- All exceptions are handled in Handler mode.
- Processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the Interrupt Service Routine (ISR).
- The vector is fetched in parallel to the state saving, enabling efficient interrupt entry.

The processor supports tail-chaining that enables back-to-back interrupts without the overhead of state saving and restoration.

Software can choose only to enable a subset of the configured number of interrupts, and can choose how many bits of the configured priorities to use.

Exceptions can be specified as either Secure or Non-secure. When an exception occurs the processor switches to the associated security state. The priority of Secure and Non-secure exceptions can be programmed independently. You

can deprioritise Non-secure configurable exceptions using the [AIRCR.PRI5](#) bit field to enable Secure interrupts to take priority.

When taking and returning from an exception, the register state is always stored using the stack pointer associated with the background security state. When taking a Non-secure exception from Secure state, all the register state is stacked and then registers are cleared to prevent Secure data being available to the Non-secure handler. The vector base address is banked between Secure and Non-secure state. [VTOR_S](#) contains the Secure vector base address, and [VTOR_NS](#) contains the Non-secure vector base address. These registers can be programmed by software, and also initialized at reset by the system.

NOTE

Vector table entries are compatible with interworking between Arm and Thumb instructions. This causes bit[0] of the vector value to load into the Execution Program Status Register (EPSR) T-bit on exception entry. All populated vectors in the vector table entries must have bit[0] set. Creating a table entry with bit[0] clear generates an [INVSTATE](#) fault on the first instruction of the handler corresponding to this vector.

3.7.4.7. Security Attribution and Memory Protection

Security attribution and memory protection in the processor is provided by the Security Attribution Unit (SAU) and the Memory Protection Units (MPUs).

The SAU is a programmable unit that determines the security of an address. RP2350 includes 8 memory regions.

For instructions and data, the SAU returns the security attribute that is associated with the address.

For instructions, the attribute determines the allowable Security state of the processor when the instruction is executed. It can also identify whether code at a Secure address can be called from Non-secure state.

For data, the attribute determines whether a memory address can be accessed from Non-secure state, and also whether the external memory request is marked as Secure or Non-secure.

If a data access is made from Non-secure state to an address marked as Secure, then a SecureFault exception is taken by the processor. If a data access is made from Secure state to an address marked as Non-secure, then the associated memory access is marked as Non-secure.

The security level returned by the SAU is a combination of the region type defined in the internal SAU, if configured, and the type that is returned on the associated Implementation Defined Attribution Unit (IDAU). If an address maps to regions defined by both internal and external attribution units, the region of the highest security level is selected.

The register fields [SAU_CTRL.EN](#) and [SAU_CTRL.ALLNS](#) control the enable state of the SAU and the default security level when the SAU is disabled. Both [SAU_CTRL.EN](#) and [SAU_CTRL.ALLNS](#) reset to zero disabling the SAU and setting all memory, apart from some specific regions in the PPB space to Secure level. If the SAU is not enabled, and [SAU_CTRL.ALLNS](#) is zero, then the IDAU cannot set any regions of memory to a security level lower than Secure, for example Secure NSC or NS. If the SAU is enabled, then [SAU_CTRL.ALLNS](#) does not affect the Security level of memory.

RP2350 supports the Armv8-M Protected Memory System Architecture (PMSA). The MPU provides full support for:

- protection regions
- access permissions
- exporting memory attributes to the system

MPU mismatches and permission violations invoke the MemManage handler. For more information, see the [Armv8-M Architecture Reference Manual](#).

You can use the MPU to:

- enforce privilege rules
- separate processes

- manage memory attributes

The MPU supports 16 memory regions: 8 secure and 8 non-secure. The MPU is banked between Secure and Non-secure states. The number of regions in the Secure and Non-secure MPU can be configured independently and each can be programmed to protect memory for the associated Security state.

3.7.4.8. External coprocessors

The external coprocessor interface:

- Supports low-latency data transfer from the processor to and from the accelerator components.
- Has a sustained bandwidth up to twice of the processor memory interface.

The following instruction types are supported:

- Register transfer from the Cortex-M33 processor to the coprocessor **MCR**, **MCRR**, **MCR2**, **MCRR2**.
- Register transfer from the coprocessor to the Cortex-M33 processor **MRC**, **MRRC**, **MRC2**, **MRRC2**.
- Data processing instructions **CDP**, **CDP2**.

NOTE

The regular and extension forms of the coprocessor instructions for example, **MCR** and **MCRR2**, have the same functionality but different encodings. The **MRC** and **MRC2** instructions support the transfer of **APSR.NZVC** flags when the processor register field is set to PC, for example **Rt == 0xF**.

3.7.4.8.1. Restrictions

The following restrictions apply when to coprocessor instructions:

- The **LDC(2)** or **STC(2)** instructions are not supported. If these are included in software with the **<coproc>** field set to a value between 0-7 and the coprocessor is present and enabled in the appropriate fields in the **CPACR/NSACR** registers, the Cortex-M33 processor always attempts to take an *Undefined instruction* (UNDEFINSTR) UsageFault exception.
- The processor register fields for data transfer instructions must not include the stack pointer (**Rt == 0xD**), this encoding is **UNPREDICTABLE** in the Armv8-M architecture and results in an *Undefined instruction* (UNDEFINSTR) UsageFault exception in the **CPACR/NSACR** registers.
- If any coprocessor instruction is executed when the corresponding coprocessor is disabled in the **CPACR/NSACR** register, the Cortex-M33 processor always attempts to take a *No coprocessor* (NOCP) UsageFault exception.

3.7.4.8.2. Data transfer rates

The following table shows the ideal data transfer rates for the coprocessor interface. This means that the coprocessor responds immediately to an instruction. The ideal data transfer rates are sustainable if the corresponding coprocessor instructions are executed consecutively.

The following instructions have the following data transfer rates:

MCR, MCR2 (Processor to coprocessor)

32 bits per cycle

MRC, MRC2 (Coprocessor to processor)

32 bits per cycle

MCRR, MCRR2 (Processor to coprocessor)

64 bits per cycle

MRRC, MRRC2 (Coprocessor to processor)

64 bits per cycle

3.7.4.9. Debug

Cortex-M33 debug functionality includes processor halt, single-step, processor core register access, Vector Catch, unlimited software breakpoints, and full system memory access.

The processor also includes support for hardware breakpoints and watchpoints configured during implementation:

- A breakpoint unit supporting eight instruction comparators
- A watchpoint unit supporting four data watchpoint comparators

The Cortex-M33 processor supports system level debug authentication to control access from a debugger to resources and memory. Authentication via the Armv8-M Security Extension can be used to allow a debugger full access to Non-secure code and data without exposing any Secure information.

The processor implementation can be partitioned to place the debug components in a separate power domain from the processor core and NVIC.

All debug registers are accessible by the D-AHB interface.

For more information, see the [Armv8-M Architecture Reference Manual](#).

3.7.4.10. Data Watchpoint and Trace unit (DWT)

The DWT is a full configuration, containing four comparators ([DWT_COMP0](#) to [DWT_COMP3](#)). These comparators support the following features:

- Hardware watchpoint support
- Hardware trace packet support
- CMPMATCH support for ETM/MTB/CTI triggers
- Cycle counter matching support ([DWT_COMP0](#) only)
- Instruction address matching support
- Data address matching support
- Data value matching support ([DWT_COMP1](#) only in a reduced DWT, [DWT_COMP3](#) only in a Full DWT)
- Linked/limit matching support ([DWT_COMP1](#) and [DWT_COMP3](#) only)

The DWT contains counters for:

- Cycles ([DWT_CYCCNT](#), [CYCCNT](#))
- Folded Instructions (FOLDCNT)
- Additional cycles required to execute all load/store instructions (LSUCNT)
- Processor sleep cycles (SLEEPCNT)
- Additional cycles required to execute multi-cycle instructions and instruction fetch stalls (CPICNT)
- Cycles spent in exception processing (EXCCNT)

Before using DWT, set the [DEMCR](#).[TRCENA](#) bit to 1.

The DWT provides periodic requests for protocol synchronization to the ITM and the TPIU.

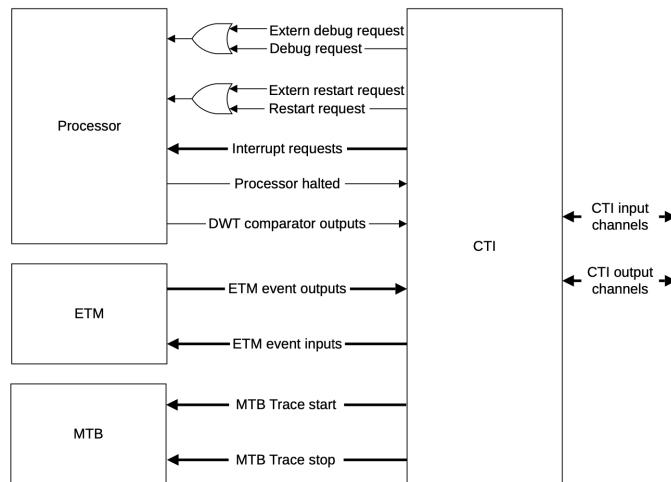
3.7.4.11. Cross Trigger Interface (CTI)

The CTI enables the debug logic, MTB, and ETM to interact with each other and with other CoreSight components. This is called cross triggering. For example, you can configure the CTI to generate an interrupt when the ETM trigger event occurs or to start tracing when a DWT comparator match is detected.

The following figure shows the debug system components and the available trigger inputs and trigger outputs:

[Figure 14](#) shows the components of the debug system.

Figure 14. Debug system components



The following table shows how the CTI trigger inputs are connected to the Cortex-M33 processor:

Table 112. Trigger signals to the CTI

Signal	Description	Connection	Acknowledge, handshake
CTITRIGIN[7]		ETM to CTI	Pulsed
CTITRIGIN[6]		ETM to CTI	Pulsed
CTITRIGIN[5]	ETM Event Output 1	ETM to CTI	Pulsed
CTITRIGIN[4]	ETM Event Output 0 or Comparator Output 3	ETM/Processor to CTI	Pulsed
CTITRIGIN[3]	DWT Comparator Output 2	Processor to CTI	Pulsed
CTITRIGIN[2]	DWT Comparator Output 1	Processor to CTI	Pulsed
CTITRIGIN[1]	DWT Comparator Output 0	Processor to CTI	Pulsed
CTITRIGIN[0]	Processor Halted	Processor to CTI	Pulsed

The following table shows how the CTI trigger outputs are connected to the processor and ETM:

Table 113. Trigger signals from the CTI

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[7]	ETM Event Input 3	CTI to ETM	Pulsed
CTITRIGOUT[6]	ETM Event Input 2	CTI to ETM	Pulsed
CTITRIGOUT[5]	ETM Event Input 1 or MTB Trace stop	CTI to ETM or MTB	Pulsed
CTITRIGOUT[4]	ETM Event Input 1 or MTB Trace start	CTI to ETM or MTB	Pulsed

Signal	Description	Connection	Acknowledge, handshake
CTITRIGOUT[3]	Interrupt request 1	CTI to system	Acknowledged by writing to the CTIINTACK register in ISR
CTITRIGOUT[2]	Interrupt request 0	CTI to system	Acknowledged by writing to the CTIINTACK register in ISR
CTITRIGOUT[1]	Processor Restart	CTI to Processor	Processor Restarted
CTITRIGOUT[0]	Processor debug request	CTI to Processor	Acknowledged by the debugger writing to the CTIINTACK register

After the processor is halted using CTI Trigger Output 0, the Processor Debug Request signal remains asserted. The debugger must write to [CTIINTACK](#) to clear the halting request before restarting the processor.

After asserting an interrupt using the CTI Trigger Output 1 or 2, the Interrupt Service Routine (ISR) must clear the interrupt request by writing to the CTI Interrupt Acknowledge, [CTIINTACK](#).

Interrupt requests from the CTI to the system are only asserted when invasive debug is enabled in the processor.

3.7.4.11.1. CTI programmers model

The following table shows the CTI programmable registers, with address offset, type, and reset value for each register. See the Arm [CoreSight™ SoC-400 Technical Reference Manual](#) for register descriptions.

Table 114. Cortex-M33
CTI register summary

Address offset	Name	Type	Reset value	Description
0xE0042000	CTICONTROL	RW	0x00000000	CTI Control Register
0xE0042010	CTIINTACK	WO	UNKNOWN	CTI Interrupt Acknowledge Register
0xE0042014	CTIAPPSET	RW	0x00000000	CTI Application Trigger Set Register
0xE0042018	CTIAPPCLEAR	RW	0x00000000	CTI Application Trigger Clear Register
0xE004201C	CTIAPPPULSE	WO	UNKNOWN	CTI Application Pulse Register
0xE0042020-0xE004203C	CTIINEN[7:0]	RW	0x00000000	CTI Trigger to Channel Enable Registers
0xE00420A0-0xE00420BC	CTIOUTEN[7:0]	RW	0x00000000	CTI Channel to Trigger Enable Registers
0xE0042130	CTITRIGINSTATUS	RO	0x00000000	CTI Trigger In Status Register
0xE0042134	CTITRIGOUTSTATUS	RO	0x00000000	CTI Trigger Out Status Register
0xE0042138	CTICHINSTATUS	RO	0x00000000	CTI Channel In Status Register
0xE0042140	CTIGATE	RW	0x0000000F	Enable CTI Channel Gate Register
0xE0042144	ASICCTL	RW	0x00000000	External Multiplexer Control Register
0xE0042EE4	ITCHOUT	WO	UNKNOWN	Integration Test Channel Output Register
0xE0042EE8	ITTRIGOUT	WO	UNKNOWN	Integration Test Trigger Output Register

Address offset	Name	Type	Reset value	Description
0xE0042EF4	ITCHIN	RO	0x00000000	Integration Test Channel Input Register
0xE0042F00	ITCTRL	RW	0x00000000	Integration Mode Control Register
0xE0042FC8	DEVID	RO	0x00040800	Device Configuration Register
0xE0042FBC	DEVARCH	RO	0x47701A14	Device Architecture Register
0xE0042FCC	DEVTYPE	RO	0x00000014	Device Type Identifier Register
0xE0042FD0	PIDR4	RO	0x00000004	Peripheral ID4 Register
0xE0042FD4	PIDR5	RO	0x00000000	Peripheral ID5 Register
0xE0042FD8	PIDR6	RO	0x00000000	Peripheral ID6 Register
0xE0042FDC	PIDR7	RO	0x00000000	Peripheral ID7 Register
0xE0042FE0	PIDR0	RO	0x00000021	Peripheral ID0 Register
0xE0042FE4	PIDR1	RO	0x000000BD	Peripheral ID1 Register
0xE0042FE8	PIDR2	RO	0x0000000B	Peripheral ID2 Register
0xE0042FEC	PIDR3	RO	0x00000001	Peripheral ID3 Register
0xE0042FF0	CIDR0	RO	0x0000000D	Component ID0 Register
0xE0042FF4	CIDR1	RO	0x00000090	Component ID1 Register
0xE0042FF8	CIDR2	RO	0x00000005	Component ID2 Register
0xE0042FFC	CIDR3	RO	0x000000B1	Component ID3 Register

3.7.5. List of Registers

The Arm Cortex-M33 registers start at a base address of `0xe0000000`, defined as `PPB_BASE` in the SDK.

Table 115. List of M33 registers

Offset	Name	Info
0x00000	ITM_STIM0	ITM Stimulus Port Register 0
0x00004	ITM_STIM1	ITM Stimulus Port Register 1
0x00008	ITM_STIM2	ITM Stimulus Port Register 2
0x0000c	ITM_STIM3	ITM Stimulus Port Register 3
0x00010	ITM_STIM4	ITM Stimulus Port Register 4
0x00014	ITM_STIM5	ITM Stimulus Port Register 5
0x00018	ITM_STIM6	ITM Stimulus Port Register 6
0x0001c	ITM_STIM7	ITM Stimulus Port Register 7
0x00020	ITM_STIM8	ITM Stimulus Port Register 8
0x00024	ITM_STIM9	ITM Stimulus Port Register 9
0x00028	ITM_STIM10	ITM Stimulus Port Register 10
0x0002c	ITM_STIM11	ITM Stimulus Port Register 11
0x00030	ITM_STIM12	ITM Stimulus Port Register 12

Offset	Name	Info
0x00034	ITM_STIM13	ITM Stimulus Port Register 13
0x00038	ITM_STIM14	ITM Stimulus Port Register 14
0x0003c	ITM_STIM15	ITM Stimulus Port Register 15
0x00040	ITM_STIM16	ITM Stimulus Port Register 16
0x00044	ITM_STIM17	ITM Stimulus Port Register 17
0x00048	ITM_STIM18	ITM Stimulus Port Register 18
0x0004c	ITM_STIM19	ITM Stimulus Port Register 19
0x00050	ITM_STIM20	ITM Stimulus Port Register 20
0x00054	ITM_STIM21	ITM Stimulus Port Register 21
0x00058	ITM_STIM22	ITM Stimulus Port Register 22
0x0005c	ITM_STIM23	ITM Stimulus Port Register 23
0x00060	ITM_STIM24	ITM Stimulus Port Register 24
0x00064	ITM_STIM25	ITM Stimulus Port Register 25
0x00068	ITM_STIM26	ITM Stimulus Port Register 26
0x0006c	ITM_STIM27	ITM Stimulus Port Register 27
0x00070	ITM_STIM28	ITM Stimulus Port Register 28
0x00074	ITM_STIM29	ITM Stimulus Port Register 29
0x00078	ITM_STIM30	ITM Stimulus Port Register 30
0x0007c	ITM_STIM31	ITM Stimulus Port Register 31
0x00e00	ITM_TER0	Provide an individual enable bit for each ITM_STIM register
0x00e40	ITM_TPR	Controls which stimulus ports can be accessed by unprivileged code
0x00e80	ITM_TCR	Configures and controls transfers through the ITM interface
0x00ef0	INT_ATREADY	Integration Mode: Read ATB Ready
0x00ef8	INT_ATVALID	Integration Mode: Write ATB Valid
0x00f00	ITM_ITCTRL	Integration Mode Control Register
0x00fb0	ITM_DEVARCH	Provides CoreSight discovery information for the ITM
0x00fcc	ITM_DEVTYPE	Provides CoreSight discovery information for the ITM
0x00fd0	ITM_PIDR4	Provides CoreSight discovery information for the ITM
0x00fd4	ITM_PIDR5	Provides CoreSight discovery information for the ITM
0x00fd8	ITM_PIDR6	Provides CoreSight discovery information for the ITM
0x00fdc	ITM_PIDR7	Provides CoreSight discovery information for the ITM
0x00fe0	ITM_PIDR0	Provides CoreSight discovery information for the ITM
0x00fe4	ITM_PIDR1	Provides CoreSight discovery information for the ITM
0x00fe8	ITM_PIDR2	Provides CoreSight discovery information for the ITM
0x00fec	ITM_PIDR3	Provides CoreSight discovery information for the ITM

Offset	Name	Info
0x00ff0	ITM_CIDR0	Provides CoreSight discovery information for the ITM
0x00ff4	ITM_CIDR1	Provides CoreSight discovery information for the ITM
0x00ff8	ITM_CIDR2	Provides CoreSight discovery information for the ITM
0x00ffc	ITM_CIDR3	Provides CoreSight discovery information for the ITM
0x01000	DWT_CTRL	Provides configuration and status information for the DWT unit, and used to control features of the unit
0x01004	DWT_CYCCNT	Shows or sets the value of the processor cycle counter, CYCCNT
0x0100c	DWT_EXCCNT	Counts the total cycles spent in exception processing
0x01014	DWT_LSUCNT	Increments on the additional cycles required to execute all load or store instructions
0x01018	DWT_FOLDCNT	Increments on the additional cycles required to execute all load or store instructions
0x01020	DWT_COMP0	Provides a reference value for use by watchpoint comparator 0
0x01028	DWT_FUNCTION0	Controls the operation of watchpoint comparator 0
0x01030	DWT_COMP1	Provides a reference value for use by watchpoint comparator 1
0x01038	DWT_FUNCTION1	Controls the operation of watchpoint comparator 1
0x01040	DWT_COMP2	Provides a reference value for use by watchpoint comparator 2
0x01048	DWT_FUNCTION2	Controls the operation of watchpoint comparator 2
0x01050	DWT_COMP3	Provides a reference value for use by watchpoint comparator 3
0x01058	DWT_FUNCTION3	Controls the operation of watchpoint comparator 3
0x01fbc	DWT_DEVARCH	Provides CoreSight discovery information for the DWT
0x01fcc	DWT_DEVTYPE	Provides CoreSight discovery information for the DWT
0x01fd0	DWT_PIDR4	Provides CoreSight discovery information for the DWT
0x01fd4	DWT_PIDR5	Provides CoreSight discovery information for the DWT
0x01fd8	DWT_PIDR6	Provides CoreSight discovery information for the DWT
0x01fdc	DWT_PIDR7	Provides CoreSight discovery information for the DWT
0x01fe0	DWT_PIDR0	Provides CoreSight discovery information for the DWT
0x01fe4	DWT_PIDR1	Provides CoreSight discovery information for the DWT
0x01fe8	DWT_PIDR2	Provides CoreSight discovery information for the DWT
0x01fec	DWT_PIDR3	Provides CoreSight discovery information for the DWT
0x01ff0	DWT_CIDR0	Provides CoreSight discovery information for the DWT
0x01ff4	DWT_CIDR1	Provides CoreSight discovery information for the DWT
0x01ff8	DWT_CIDR2	Provides CoreSight discovery information for the DWT
0x01ffc	DWT_CIDR3	Provides CoreSight discovery information for the DWT
0x02000	FP_CTRL	Provides FPB implementation information, and the global enable for the FPB unit

Offset	Name	Info
0x02004	FP_REMAP	Indicates whether the implementation supports Flash Patch remap and, if it does, holds the target address for remap
0x02008	FP_COMP0	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x0200c	FP_COMP1	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x02010	FP_COMP2	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x02014	FP_COMP3	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x02018	FP_COMP4	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x0201c	FP_COMP5	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x02020	FP_COMP6	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x02024	FP_COMP7	Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator
0x02fbc	FP_DEVARCH	Provides CoreSight discovery information for the FPB
0x02fcc	FP_DEVTYPE	Provides CoreSight discovery information for the FPB
0x02fd0	FP_PIDR4	Provides CoreSight discovery information for the FP
0x02fd4	FP_PIDR5	Provides CoreSight discovery information for the FP
0x02fd8	FP_PIDR6	Provides CoreSight discovery information for the FP
0x02fdc	FP_PIDR7	Provides CoreSight discovery information for the FP
0x02fe0	FP_PIDR0	Provides CoreSight discovery information for the FP
0x02fe4	FP_PIDR1	Provides CoreSight discovery information for the FP
0x02fe8	FP_PIDR2	Provides CoreSight discovery information for the FP
0x02fec	FP_PIDR3	Provides CoreSight discovery information for the FP

Offset	Name	Info
0x02ff0	FP_CIDR0	Provides CoreSight discovery information for the FP
0x02ff4	FP_CIDR1	Provides CoreSight discovery information for the FP
0x02ff8	FP_CIDR2	Provides CoreSight discovery information for the FP
0x02ffc	FP_CIDR3	Provides CoreSight discovery information for the FP
0x0e004	ICTR	Provides information about the interrupt controller
0x0e008	ACTLR	Provides IMPLEMENTATION DEFINED configuration and control options
0x0e010	SYST_CSR	SysTick Control and Status Register
0x0e014	SYST_RVR	SysTick Reload Value Register
0x0e018	SYST_CVR	SysTick Current Value Register
0x0e01c	SYST_CALIB	SysTick Calibration Value Register
0x0e100	NVIC_ISERO	Enables or reads the enabled state of each group of 32 interrupts
0x0e104	NVIC_ISER1	Enables or reads the enabled state of each group of 32 interrupts
0x0e180	NVIC_ICERO	Clears or reads the enabled state of each group of 32 interrupts
0x0e184	NVIC_ICER1	Clears or reads the enabled state of each group of 32 interrupts
0x0e200	NVIC_ISPR0	Enables or reads the pending state of each group of 32 interrupts
0x0e204	NVIC_ISPR1	Enables or reads the pending state of each group of 32 interrupts
0x0e280	NVIC_ICPRO	Clears or reads the pending state of each group of 32 interrupts
0x0e284	NVIC_ICPR1	Clears or reads the pending state of each group of 32 interrupts
0x0e300	NVIC_IABR0	For each group of 32 interrupts, shows the active state of each interrupt
0x0e304	NVIC_IABR1	For each group of 32 interrupts, shows the active state of each interrupt
0x0e380	NVIC_ITNS0	For each group of 32 interrupts, determines whether each interrupt targets Non-secure or Secure state
0x0e384	NVIC_ITNS1	For each group of 32 interrupts, determines whether each interrupt targets Non-secure or Secure state
0x0e400	NVIC_IPR0	Sets or reads interrupt priorities
0x0e404	NVIC_IPR1	Sets or reads interrupt priorities
0x0e408	NVIC_IPR2	Sets or reads interrupt priorities
0x0e40c	NVIC_IPR3	Sets or reads interrupt priorities
0x0e410	NVIC_IPR4	Sets or reads interrupt priorities
0x0e414	NVIC_IPR5	Sets or reads interrupt priorities
0x0e418	NVIC_IPR6	Sets or reads interrupt priorities
0x0e41c	NVIC_IPR7	Sets or reads interrupt priorities
0x0e420	NVIC_IPR8	Sets or reads interrupt priorities
0x0e424	NVIC_IPR9	Sets or reads interrupt priorities

Offset	Name	Info
0x0e428	NVIC_IPR10	Sets or reads interrupt priorities
0x0e42c	NVIC_IPR11	Sets or reads interrupt priorities
0x0e430	NVIC_IPR12	Sets or reads interrupt priorities
0x0e434	NVIC_IPR13	Sets or reads interrupt priorities
0x0e438	NVIC_IPR14	Sets or reads interrupt priorities
0x0e43c	NVIC_IPR15	Sets or reads interrupt priorities
0x0ed00	CPUID	Provides identification information for the PE, including an implementer code for the device and a device ID number
0x0ed04	ICSR	Controls and provides status information for NMI, PendSV, SysTick and interrupts
0x0ed08	VTOR	Vector Table Offset Register
0x0ed0c	AIRCR	Application Interrupt and Reset Control Register
0x0ed10	SCR	System Control Register
0x0ed14	CCR	Sets or returns configuration and control data
0x0ed18	SHPR1	Sets or returns priority for system handlers 4 - 7
0x0ed1c	SHPR2	Sets or returns priority for system handlers 8 - 11
0x0ed20	SHPR3	Sets or returns priority for system handlers 12 - 15
0x0ed24	SHCSR	Provides access to the active and pending status of system exceptions
0x0ed28	CFSR	Contains the three Configurable Fault Status Registers. 31:16 UFSR: Provides information on UsageFault exceptions 15:8 BCSR: Provides information on BusFault exceptions 7:0 MMFSR: Provides information on MemManage exceptions
0x0ed2c	HFSR	Shows the cause of any HardFaults
0x0ed30	DFSR	Shows which debug event occurred
0x0ed34	MMFAR	Shows the address of the memory location that caused an MPU fault
0x0ed38	BFAR	Shows the address associated with a precise data access BusFault
0x0ed40	ID_PFR0	Gives top-level information about the instruction set supported by the PE
0x0ed44	ID_PFR1	Gives information about the programmers' model and Extensions support
0x0ed48	ID_DFR0	Provides top level information about the debug system
0x0ed4c	ID_AFRO	Provides information about the IMPLEMENTATION DEFINED features of the PE
0x0ed50	ID_MMFR0	Provides information about the implemented memory model and memory management support

Offset	Name	Info
0x0ed54	ID_MMFR1	Provides information about the implemented memory model and memory management support
0x0ed58	ID_MMFR2	Provides information about the implemented memory model and memory management support
0x0ed5c	ID_MMFR3	Provides information about the implemented memory model and memory management support
0x0ed60	ID_ISAR0	Provides information about the instruction set implemented by the PE
0x0ed64	ID_ISAR1	Provides information about the instruction set implemented by the PE
0x0ed68	ID_ISAR2	Provides information about the instruction set implemented by the PE
0x0ed6c	ID_ISAR3	Provides information about the instruction set implemented by the PE
0x0ed70	ID_ISAR4	Provides information about the instruction set implemented by the PE
0x0ed74	ID_ISAR5	Provides information about the instruction set implemented by the PE
0x0ed7c	CTR	Provides information about the architecture of the caches. CTR is RES0 if CLIDR is zero.
0x0ed88	CPACR	Specifies the access privileges for coprocessors and the FP Extension
0x0ed8c	NSACR	Defines the Non-secure access permissions for both the FP Extension and coprocessors CP0 to CP7
0x0ed90	MPU_TYPE	The MPU Type Register indicates how many regions the MPU `FTSSS supports
0x0ed94	MPU_CTRL	Enables the MPU and, when the MPU is enabled, controls whether the default memory map is enabled as a background region for privileged accesses, and whether the MPU is enabled for HardFaults, NMIs, and exception handlers when FAULTMASK is set to 1
0x0ed98	MPU_RNR	Selects the region currently accessed by MPU_RBAR and MPU_RLAR
0x0ed9c	MPU_RBAR	Provides indirect read and write access to the base address of the currently selected MPU region `FTSSS
0x0eda0	MPU_RLAR	Provides indirect read and write access to the limit address of the currently selected MPU region `FTSSS
0x0eda4	MPU_RBAR_A1	Provides indirect read and write access to the base address of the MPU region selected by MPU_RNR[7:2]:(1[1:0]) `FTSSS
0x0eda8	MPU_RLAR_A1	Provides indirect read and write access to the limit address of the currently selected MPU region selected by MPU_RNR[7:2]:(1[1:0]) `FTSSS
0x0edac	MPU_RBAR_A2	Provides indirect read and write access to the base address of the MPU region selected by MPU_RNR[7:2]:(2[1:0]) `FTSSS

Offset	Name	Info
0x0edb0	MPU_RLAR_A2	Provides indirect read and write access to the limit address of the currently selected MPU region selected by MPU_RNR[7:2]:(2[1:0]) `FTSSS
0x0edb4	MPU_RBAR_A3	Provides indirect read and write access to the base address of the MPU region selected by MPU_RNR[7:2]:(3[1:0]) `FTSSS
0x0edb8	MPU_RLAR_A3	Provides indirect read and write access to the limit address of the currently selected MPU region selected by MPU_RNR[7:2]:(3[1:0]) `FTSSS
0x0edc0	MPU_MAIR0	Along with MPU_MAIR1, provides the memory attribute encodings corresponding to the AttrIndex values
0x0edc4	MPU_MAIR1	Along with MPU_MAIR0, provides the memory attribute encodings corresponding to the AttrIndex values
0x0edd0	SAU_CTRL	Allows enabling of the Security Attribution Unit
0x0edd4	SAU_TYPE	Indicates the number of regions implemented by the Security Attribution Unit
0x0edd8	SAU_RNR	Selects the region currently accessed by SAU_RBAR and SAU_RLAR
0x0eddc	SAU_RBAR	Provides indirect read and write access to the base address of the currently selected SAU region
0x0ede0	SAU_RLAR	Provides indirect read and write access to the limit address of the currently selected SAU region
0x0ede4	SFSR	Provides information about any security related faults
0x0ede8	SFAR	Shows the address of the memory location that caused a Security violation
0x0edf0	DHCSR	Controls halting debug
0x0edf4	DCRSR	With the DCRDR, provides debug access to the general-purpose registers, special-purpose registers, and the FP extension registers. A write to the DCRSR specifies the register to transfer, whether the transfer is a read or write, and starts the transfer
0x0edf8	DCRDR	With the DCRSR, provides debug access to the general-purpose registers, special-purpose registers, and the FP Extension registers. If the Main Extension is implemented, it can also be used for message passing between an external debugger and a debug agent running on the PE
0x0edfc	DEMCR	Manages vector catch behavior and DebugMonitor handling when debugging
0x0ee08	DCSR	Provides control and status information for Secure debug
0x0ef00	STIR	Provides a mechanism for software to generate an interrupt
0x0ef34	FPCCR	Holds control data for the Floating-point extension
0x0ef38	FPCAR	Holds the location of the unpopulated floating-point register space allocated on an exception stack frame
0x0ef3c	FPDSCR	Holds the default values for the floating-point status control data that the PE assigns to the FPSCR when it creates a new floating-point context

Offset	Name	Info
0x0ef40	MVFR0	Describes the features provided by the Floating-point Extension
0x0ef44	MVFR1	Describes the features provided by the Floating-point Extension
0x0ef48	MVFR2	Describes the features provided by the Floating-point Extension
0x0efbc	DDEVARCH	Provides CoreSight discovery information for the SCS
0x0efcc	DDEVTYPE	Provides CoreSight discovery information for the SCS
0x0efd0	DPIDR4	Provides CoreSight discovery information for the SCS
0x0efd4	DPIDR5	Provides CoreSight discovery information for the SCS
0x0efd8	DPIDR6	Provides CoreSight discovery information for the SCS
0x0efdc	DPIDR7	Provides CoreSight discovery information for the SCS
0x0efe0	DPIDR0	Provides CoreSight discovery information for the SCS
0x0efe4	DPIDR1	Provides CoreSight discovery information for the SCS
0x0efe8	DPIDR2	Provides CoreSight discovery information for the SCS
0x0efec	DPIDR3	Provides CoreSight discovery information for the SCS
0x0eff0	DCIDR0	Provides CoreSight discovery information for the SCS
0x0eff4	DCIDR1	Provides CoreSight discovery information for the SCS
0x0eff8	DCIDR2	Provides CoreSight discovery information for the SCS
0x0effc	DCIDR3	Provides CoreSight discovery information for the SCS
0x41004	TRCPRGCTRLR	Programming Control Register
0x4100c	TRCSTATR	The TRCSTATR indicates the ETM-Teal status
0x41010	TRCCONFIGR	The TRCCONFIGR sets the basic tracing options for the trace unit
0x41020	TRCEVENTCTL0R	The TRCEVENTCTL0R controls the tracing of events in the trace stream. The events also drive the ETM-Teal external outputs.
0x41024	TRCEVENTCTL1R	The TRCEVENTCTL1R controls how the events selected by TRCEVENTCTL0R behave
0x4102c	TRCSTALLCTRLR	The TRCSTALLCTRLR enables ETM-Teal to stall the processor if the ETM-Teal FIFO goes over the programmed level to minimize risk of overflow
0x41030	TRCTSCTRLR	The TRCTSCTRLR controls the insertion of global timestamps into the trace stream. A timestamp is always inserted into the instruction trace stream
0x41034	TRCSYNCPR	The TRCSYNCPR specifies the period of trace synchronization of the trace streams. TRCSYNCPR defines a number of bytes of trace between requests for trace synchronization. This value is always a power of two
0x41038	TRCCCCTRLR	The TRCCCCTRLR sets the threshold value for instruction trace cycle counting. The threshold represents the minimum interval between cycle count trace packets
0x41080	TRCVICTLR	The TRCVICTLR controls instruction trace filtering

Offset	Name	Info
0x41140	TRCCNTRLDVR0	The TRCCNTRLDVR defines the reload value for the reduced function counter
0x41180	TRCIDR8	TRCIDR8
0x41184	TRCIDR9	TRCIDR9
0x41188	TRCIDR10	TRCIDR10
0x4118c	TRCIDR11	TRCIDR11
0x41190	TRCIDR12	TRCIDR12
0x41194	TRCIDR13	TRCIDR13
0x411c0	TRCIMSPEC	The TRCIMSPEC shows the presence of any IMPLEMENTATION SPECIFIC features, and enables any features that are provided
0x411e0	TRCIDR0	TRCIDR0
0x411e4	TRCIDR1	TRCIDR1
0x411e8	TRCIDR2	TRCIDR2
0x411ec	TRCIDR3	TRCIDR3
0x411f0	TRCIDR4	TRCIDR4
0x411f4	TRCIDR5	TRCIDR5
0x411f8	TRCIDR6	TRCIDR6
0x411fc	TRCIDR7	TRCIDR7
0x41208	TRCRSCTRL2	The TRCRSCTRL controls the trace resources
0x4120c	TRCRSCTRL3	The TRCRSCTRL controls the trace resources
0x412a0	TRCSSCSR	Controls the corresponding single-shot comparator resource
0x412c0	TRCSSPCICR	Selects the PE comparator inputs for Single-shot control
0x41310	TRCPDCR	Requests the system to provide power to the trace unit
0x41314	TRCPDSR	Returns the following information about the trace unit: - OS Lock status. - Core power domain status. - Power interruption status
0x41ee4	TRCITATBIDR	Trace Intergration ATB Identification Register
0x41ef4	TRCITIATBINR	Trace Integration Instruction ATB In Register
0x41efc	TRCITIATBOUR	Trace Integration Instruction ATB Out Register
0x41fa0	TRCCLAIMSET	Claim Tag Set Register
0x41fa4	TRCCLAIMCLR	Claim Tag Clear Register
0x41fb8	TRCAUTHSTATUS	Returns the level of tracing that the trace unit can support
0x41fbc	TRCDEVARCH	TRCDEVARCH
0x41fc8	TRCDEVID	TRCDEVID
0x41fcc	TRCDEVTYPE	TRCDEVTYPE
0x41fd0	TRCPIDR4	TRCPIDR4
0x41fd4	TRCPIDR5	TRCPIDR5
0x41fd8	TRCPIDR6	TRCPIDR6

Offset	Name	Info
0x41fdc	TRCPIDR7	TRCPIDR7
0x41fe0	TRCPIDR0	TRCPIDR0
0x41fe4	TRCPIDR1	TRCPIDR1
0x41fe8	TRCPIDR2	TRCPIDR2
0x41fec	TRCPIDR3	TRCPIDR3
0x41ff0	TRCCIDR0	TRCCIDR0
0x41ff4	TRCCIDR1	TRCCIDR1
0x41ff8	TRCCIDR2	TRCCIDR2
0x41ffc	TRCCIDR3	TRCCIDR3
0x42000	CTICONTROL	CTI Control Register
0x42010	CTIINTACK	CTI Interrupt Acknowledge Register
0x42014	CTIAPPSET	CTI Application Trigger Set Register
0x42018	CTIAPPCLEAR	CTI Application Trigger Clear Register
0x4201c	CTIAPPPULSE	CTI Application Pulse Register
0x42020	CTIINEN0	CTI Trigger to Channel Enable Registers
0x42024	CTIINEN1	CTI Trigger to Channel Enable Registers
0x42028	CTIINEN2	CTI Trigger to Channel Enable Registers
0x4202c	CTIINEN3	CTI Trigger to Channel Enable Registers
0x42030	CTIINEN4	CTI Trigger to Channel Enable Registers
0x42034	CTIINEN5	CTI Trigger to Channel Enable Registers
0x42038	CTIINEN6	CTI Trigger to Channel Enable Registers
0x4203c	CTIINEN7	CTI Trigger to Channel Enable Registers
0x420a0	CTIOUTEN0	CTI Trigger to Channel Enable Registers
0x420a4	CTIOUTEN1	CTI Trigger to Channel Enable Registers
0x420a8	CTIOUTEN2	CTI Trigger to Channel Enable Registers
0x420ac	CTIOUTEN3	CTI Trigger to Channel Enable Registers
0x420b0	CTIOUTEN4	CTI Trigger to Channel Enable Registers
0x420b4	CTIOUTEN5	CTI Trigger to Channel Enable Registers
0x420b8	CTIOUTEN6	CTI Trigger to Channel Enable Registers
0x420bc	CTIOUTEN7	CTI Trigger to Channel Enable Registers
0x42130	CTITRIGINSTATUS	CTI Trigger to Channel Enable Registers
0x42134	CTITRIGOUTSTATUS	CTI Trigger In Status Register
0x42138	CTICHINSTATUS	CTI Channel In Status Register
0x42140	CTIGATE	Enable CTI Channel Gate register
0x42144	ASICCTL	External Multiplexer Control register
0x42ee4	ITCHOUT	Integration Test Channel Output register

Offset	Name	Info
0x42ee8	ITTRIGOUT	Integration Test Trigger Output register
0x42ef4	ITCHIN	Integration Test Channel Input register
0x42f00	ITCTRL	Integration Mode Control register
0x42fbc	DEVARCH	Device Architecture register
0x42fc8	DEVID	Device Configuration register
0x42fcc	DEVTYPE	Device Type Identifier register
0x42fd0	PIDR4	CoreSight Periperal ID4
0x42fd4	PIDR5	CoreSight Periperal ID5
0x42fd8	PIDR6	CoreSight Periperal ID6
0x42fdc	PIDR7	CoreSight Periperal ID7
0x42fe0	PIDR0	CoreSight Periperal ID0
0x42fe4	PIDR1	CoreSight Periperal ID1
0x42fe8	PIDR2	CoreSight Periperal ID2
0x42fec	PIDR3	CoreSight Periperal ID3
0x42ff0	CIDR0	CoreSight Component ID0
0x42ff4	CIDR1	CoreSight Component ID1
0x42ff8	CIDR2	CoreSight Component ID2
0x42ffc	CIDR3	CoreSight Component ID3

M33: ITM_STIM0, ITM_STIM1, ..., ITM_STIM30, ITM_STIM31 Registers

Offsets: 0x00000, 0x00004, ..., 0x00078, 0x0007c

Description

Provides the interface for generating Instrumentation packets

Table 116.
ITM_STIM0,
ITM_STIM1, ...,
ITM_STIM30,
ITM_STIM31 Registers

Bits	Name	Description	Type	Reset
31:0	STIMULUS	Data to write to the Stimulus Port FIFO, for forwarding as an Instrumentation packet. The size of write access determines the type of Instrumentation packet generated.	RW	0x00000000

M33: ITM_TER0 Register

Offset: 0x00e00

Description

Provide an individual enable bit for each ITM_STIM register

Table 117. ITM_TER0 Register

Bits	Name	Description	Type	Reset
31:0	STIMENA	For STIMENA[m] in ITM_TER*n, controls whether ITM_STIM(32*n + m) is enabled	RW	0x00000000

M33: ITM_TPR Register

Offset: 0x00e40

Description

Controls which stimulus ports can be accessed by unprivileged code

Table 118. ITM_TPR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	PRIVMASK	Bit mask to enable tracing on ITM stimulus ports	RW	0x0

M33: ITM_TCR Register

Offset: 0x00e80

Description

Configures and controls transfers through the ITM interface

Table 119. ITM_TCR Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23	BUSY	Indicates whether the ITM is currently processing events	RO	0x0
22:16	TRACEBUSID	Identifier for multi-source trace stream formatting. If multi-source trace is in use, the debugger must write a unique non-zero trace ID value to this field	RW	0x00
15:12	Reserved.	-	-	-
11:10	GTSFREQ	Defines how often the ITM generates a global timestamp, based on the global timestamp clock frequency, or disables generation of global timestamps	RW	0x0
9:8	TSPRESCALE	Local timestamp prescaler, used with the trace packet reference clock	RW	0x0
7:6	Reserved.	-	-	-
5	STALLENA	Stall the PE to guarantee delivery of Data Trace packets.	RW	0x0
4	SWOENA	Enables asynchronous clocking of the timestamp counter	RW	0x0
3	TXENA	Enables forwarding of hardware event packet from the DWT unit to the ITM for output to the TPIU	RW	0x0
2	SYNCENA	Enables Synchronization packet transmission for a synchronous TPIU	RW	0x0
1	TSENA	Enables Local timestamp generation	RW	0x0
0	ITMENA	Enables the ITM	RW	0x0

M33: INT_ATREADY Register

Offset: 0x00ef0

Description

Integration Mode: Read ATB Ready

Table 120. INT_ATREADY Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	AFVALID	A read of this bit returns the value of AFVALID	RO	0x0
0	ATREADY	A read of this bit returns the value of ATREADY	RO	0x0

M33: INT_ATVALID Register

Offset: 0x00ef8

Description

Integration Mode: Write ATB Valid

Table 121.
INT_ATVALID Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	AFREADY	A write to this bit gives the value of AFREADY	RW	0x0
0	ATREADY	A write to this bit gives the value of ATVALID	RW	0x0

M33: ITM_ITCTRL Register

Offset: 0x00f00

Description

Integration Mode Control Register

Table 122.
ITM_ITCTRL Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	IME	Integration mode enable bit - The possible values are: 0 - The trace unit is not in integration mode. 1 - The trace unit is in integration mode. This mode enables: A debug agent to perform topology detection. SoC test software to perform integration testing.	RW	0x0

M33: ITM_DEVARCH Register

Offset: 0x00fbc

Description

Provides CoreSight discovery information for the ITM

Table 123.
ITM_DEVARCH Register

Bits	Name	Description	Type	Reset
31:21	ARCHITECT	Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.	RO	0x23b
20	PRESENT	Defines that the DEVARCH register is present	RO	0x1
19:16	REVISION	Defines the architecture revision of the component	RO	0x0
15:12	ARCHVER	Defines the architecture version of the component	RO	0x1
11:0	ARCHPART	Defines the architecture of the component	RO	0xa01

M33: ITM_DEVTYPE Register

Offset: 0x00fcc

Description

Provides CoreSight discovery information for the ITM

Table 124.
ITM_DEVTYPE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7:4	SUB	Component sub-type	RO	0x4
3:0	MAJOR	Component major type	RO	0x3

M33: ITM_PIDR4 Register

Offset: 0x00fd0

Description

Provides CoreSight discovery information for the ITM

Table 125. ITM_PIDR4 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SIZE	See CoreSight Architecture Specification	RO	0x0
3:0	DES_2	See CoreSight Architecture Specification	RO	0x4

M33: ITM_PIDR5 Register

Offset: 0x00fd4

Description

Provides CoreSight discovery information for the ITM

Table 126. ITM_PIDR5 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: ITM_PIDR6 Register

Offset: 0x00fd8

Description

Provides CoreSight discovery information for the ITM

Table 127. ITM_PIDR6 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: ITM_PIDR7 Register

Offset: 0x00fdc

Description

Provides CoreSight discovery information for the ITM

Table 128. ITM_PIDR7 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: ITM_PIDR0 Register

Offset: 0x00fe0

Description

Provides CoreSight discovery information for the ITM

Table 129. ITM_PIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PART_0	See CoreSight Architecture Specification	RO	0x21

M33: ITM_PIDR1 Register

Offset: 0x00fe4

Description

Provides CoreSight discovery information for the ITM

Table 130. ITM_PIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DES_0	See CoreSight Architecture Specification	RO	0xb
3:0	PART_1	See CoreSight Architecture Specification	RO	0xd

M33: ITM_PIDR2 Register

Offset: 0x00fe8

Description

Provides CoreSight discovery information for the ITM

Table 131. ITM_PIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	See CoreSight Architecture Specification	RO	0x0
3	JEDEC	See CoreSight Architecture Specification	RO	0x1
2:0	DES_1	See CoreSight Architecture Specification	RO	0x3

M33: ITM_PIDR3 Register

Offset: 0x00fec

Description

Provides CoreSight discovery information for the ITM

Table 132. ITM_PIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVAND	See CoreSight Architecture Specification	RO	0x0
3:0	CMOD	See CoreSight Architecture Specification	RO	0x0

M33: ITM_CIDR0 Register

Offset: 0x00ff0

Description

Provides CoreSight discovery information for the ITM

Table 133. ITM_CIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_0	See CoreSight Architecture Specification	RO	0x0d

M33: ITM_CIDR1 Register

Offset: 0x00ff4

Description

Provides CoreSight discovery information for the ITM

Table 134. ITM_CIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	CLASS	See CoreSight Architecture Specification	RO	0x9
3:0	PRMBL_1	See CoreSight Architecture Specification	RO	0x0

M33: ITM_CIDR2 Register

Offset: 0x00ff8

Description

Provides CoreSight discovery information for the ITM

Table 135. ITM_CIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_2	See CoreSight Architecture Specification	RO	0x05

M33: ITM_CIDR3 Register

Offset: 0x00ffc

Description

Provides CoreSight discovery information for the ITM

Table 136. ITM_CIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_3	See CoreSight Architecture Specification	RO	0xb1

M33: DWT_CTRL Register

Offset: 0x01000

Description

Provides configuration and status information for the DWT unit, and used to control features of the unit

Table 137. DWT_CTRL Register

Bits	Name	Description	Type	Reset
31:28	NUMCOMP	Number of DWT comparators implemented	RO	0x7
27	NOTRCPKT	Indicates whether the implementation does not support trace	RO	0x0
26	NOEXTTRIG	Reserved, RAZ	RO	0x0

Bits	Name	Description	Type	Reset
25	NOYCFCNT	Indicates whether the implementation does not include a cycle counter	RO	0x1
24	NOPRFCNT	Indicates whether the implementation does not include the profiling counters	RO	0x1
23	CYCDISS	Controls whether the cycle counter is disabled in Secure state	RW	0x0
22	CYCEVTENA	Enables Event Counter packet generation on POSTCNT underflow	RW	0x1
21	FOLDEVTENA	Enables DWT_FOLDCNT counter	RW	0x1
20	LSUEVTENA	Enables DWT_LSUCNT counter	RW	0x1
19	SLEEPEVTENA	Enable DWT_SLEEPNT counter	RW	0x0
18	EXCEVTENA	Enables DWT_EXCCNT counter	RW	0x1
17	CPIEVTEA	Enables DWT_CPICNT counter	RW	0x0
16	EXTTRCENA	Enables generation of Exception Trace packets	RW	0x0
15:13	Reserved.	-	-	-
12	PCSAMPLENA	Enables use of POSTCNT counter as a timer for Periodic PC Sample packet generation	RW	0x1
11:10	SYNCTAP	Selects the position of the synchronization packet counter tap on the CYCCNT counter. This determines the Synchronization packet rate	RW	0x2
9	CYCTAP	Selects the position of the POSTCNT tap on the CYCCNT counter	RW	0x0
8:5	POSTINIT	Initial value for the POSTCNT counter	RW	0x1
4:1	POSTPRESET	Reload value for the POSTCNT counter	RW	0x2
0	CYCCNTENA	Enables CYCCNT	RW	0x0

M33: DWT_CYCCNT Register

Offset: 0x01004

Description

Shows or sets the value of the processor cycle counter, CYCCNT

Table 138.
DWT_CYCCNT Register

Bits	Name	Description	Type	Reset
31:0	CYCCNT	Increments one on each processor clock cycle when DWT_CTRL.CYCCNTENA == 1 and DEMCR.TRCENA == 1. On overflow, CYCCNT wraps to zero	RW	0x00000000

M33: DWT_EXCCNT Register

Offset: 0x0100c

Description

Counts the total cycles spent in exception processing

Table 139.
DWT_EXCCNT
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	EXCCNT	Counts one on each cycle when all of the following are true: - DWT_CTRL.EXCEVTENA == 1 and DEMCR.TRCENA == 1. - No instruction is executed, see DWT_CPICT. - An exception-entry or exception-exit related operation is in progress. - Either SecureNoninvasiveDebugAllowed() == TRUE, or NS-Req for the operation is set to Non-secure and NoninvasiveDebugAllowed() == TRUE.	RW	0x00

M33: DWT_LSUCNT Register

Offset: 0x01014

Description

Increments on the additional cycles required to execute all load or store instructions

Table 140.
DWT_LSUCNT Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	LSUCNT	Counts one on each cycle when all of the following are true: - DWT_CTRL.LSUEVTENA == 1 and DEMCR.TRCENA == 1. - No instruction is executed, see DWT_CPICT. - No exception-entry or exception-exit operation is in progress, see DWT_EXCCNT. - A load-store operation is in progress. - Either SecureNoninvasiveDebugAllowed() == TRUE, or NS-Req for the operation is set to Non-secure and NoninvasiveDebugAllowed() == TRUE.	RW	0x00

M33: DWT_FOLDCNT Register

Offset: 0x01018

Description

Increments on the additional cycles required to execute all load or store instructions

Table 141.
DWT_FOLDCNT
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	FOLDCNT	Counts on each cycle when all of the following are true: - DWT_CTRL.FOLDEVTENA == 1 and DEMCR.TRCENA == 1. - At least two instructions are executed, see DWT_CPICT. - Either SecureNoninvasiveDebugAllowed() == TRUE, or the PE is in Non-secure state and NoninvasiveDebugAllowed() == TRUE. The counter is incremented by the number of instructions executed, minus one	RW	0x00

M33: DWT_COMPO Register

Offset: 0x01020

Table 142.
DWT_COMP0 Register

Bits	Description	Type	Reset
31:0	Provides a reference value for use by watchpoint comparator 0	RW	0x00000000

M33: DWT_FUNCTION0 Register

Offset: 0x01028

Description

Controls the operation of watchpoint comparator 0

Table 143.
DWT_FUNCTION0 Register

Bits	Name	Description	Type	Reset
31:27	ID	Identifies the capabilities for MATCH for comparator *n	RO	0x0b
26:25	Reserved.	-	-	-
24	MATCHED	Set to 1 when the comparator matches	RO	0x0
23:12	Reserved.	-	-	-
11:10	DATAVSIZE	Defines the size of the object being watched for by Data Value and Data Address comparators	RW	0x0
9:6	Reserved.	-	-	-
5:4	ACTION	Defines the action on a match. This field is ignored and the comparator generates no actions if it is disabled by MATCH	RW	0x0
3:0	MATCH	Controls the type of match generated by this comparator	RW	0x0

M33: DWT_COMP1 Register

Offset: 0x01030

Table 144.
DWT_COMP1 Register

Bits	Description	Type	Reset
31:0	Provides a reference value for use by watchpoint comparator 1	RW	0x00000000

M33: DWT_FUNCTION1 Register

Offset: 0x01038

Description

Controls the operation of watchpoint comparator 1

Table 145.
DWT_FUNCTION1 Register

Bits	Name	Description	Type	Reset
31:27	ID	Identifies the capabilities for MATCH for comparator *n	RO	0x11
26:25	Reserved.	-	-	-
24	MATCHED	Set to 1 when the comparator matches	RO	0x1
23:12	Reserved.	-	-	-
11:10	DATAVSIZE	Defines the size of the object being watched for by Data Value and Data Address comparators	RW	0x2
9:6	Reserved.	-	-	-
5:4	ACTION	Defines the action on a match. This field is ignored and the comparator generates no actions if it is disabled by MATCH	RW	0x2

Bits	Name	Description	Type	Reset
3:0	MATCH	Controls the type of match generated by this comparator	RW	0x8

M33: DWT_COMP2 Register

Offset: 0x01040

Table 146.
DWT_COMP2 Register

Bits	Description	Type	Reset
31:0	Provides a reference value for use by watchpoint comparator 2	RW	0x00000000

M33: DWT_FUNCTION2 Register

Offset: 0x01048

Description

Controls the operation of watchpoint comparator 2

Table 147.
DWT_FUNCTION2 Register

Bits	Name	Description	Type	Reset
31:27	ID	Identifies the capabilities for MATCH for comparator *n	RO	0x0a
26:25	Reserved.	-	-	-
24	MATCHED	Set to 1 when the comparator matches	RO	0x0
23:12	Reserved.	-	-	-
11:10	DATAVSIZE	Defines the size of the object being watched for by Data Value and Data Address comparators	RW	0x0
9:6	Reserved.	-	-	-
5:4	ACTION	Defines the action on a match. This field is ignored and the comparator generates no actions if it is disabled by MATCH	RW	0x0
3:0	MATCH	Controls the type of match generated by this comparator	RW	0x0

M33: DWT_COMP3 Register

Offset: 0x01050

Table 148.
DWT_COMP3 Register

Bits	Description	Type	Reset
31:0	Provides a reference value for use by watchpoint comparator 3	RW	0x00000000

M33: DWT_FUNCTION3 Register

Offset: 0x01058

Description

Controls the operation of watchpoint comparator 3

Table 149.
DWT_FUNCTION3 Register

Bits	Name	Description	Type	Reset
31:27	ID	Identifies the capabilities for MATCH for comparator *n	RO	0x04
26:25	Reserved.	-	-	-
24	MATCHED	Set to 1 when the comparator matches	RO	0x0
23:12	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
11:10	DATAVSIZE	Defines the size of the object being watched for by Data Value and Data Address comparators	RW	0x2
9:6	Reserved.	-	-	-
5:4	ACTION	Defines the action on a match. This field is ignored and the comparator generates no actions if it is disabled by MATCH	RW	0x0
3:0	MATCH	Controls the type of match generated by this comparator	RW	0x0

M33: DWT_DEVARCH Register

Offset: 0x01fbc

Description

Provides CoreSight discovery information for the DWT

Table 150.
DWT_DEVARCH Register

Bits	Name	Description	Type	Reset
31:21	ARCHITECT	Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.	RO	0x23b
20	PRESENT	Defines that the DEVARCH register is present	RO	0x1
19:16	REVISION	Defines the architecture revision of the component	RO	0x0
15:12	ARCHVER	Defines the architecture version of the component	RO	0x1
11:0	ARCHPART	Defines the architecture of the component	RO	0xa02

M33: DWT_DEVTYPE Register

Offset: 0x01fcc

Description

Provides CoreSight discovery information for the DWT

Table 151.
DWT_DEVTYPE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SUB	Component sub-type	RO	0x0
3:0	MAJOR	Component major type	RO	0x0

M33: DWT_PIDR4 Register

Offset: 0x01fd0

Description

Provides CoreSight discovery information for the DWT

Table 152.
DWT_PIDR4 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SIZE	See CoreSight Architecture Specification	RO	0x0
3:0	DES_2	See CoreSight Architecture Specification	RO	0x4

M33: DWT_PIDR5 Register

Offset: 0x01fd4

Description

Provides CoreSight discovery information for the DWT

Table 153.
DWT_PIDR5 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: DWT_PIDR6 Register

Offset: 0x01fd8

Description

Provides CoreSight discovery information for the DWT

Table 154.
DWT_PIDR6 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: DWT_PIDR7 Register

Offset: 0x01fdc

Description

Provides CoreSight discovery information for the DWT

Table 155.
DWT_PIDR7 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: DWT_PIDR0 Register

Offset: 0x01fe0

Description

Provides CoreSight discovery information for the DWT

Table 156.
DWT_PIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PART_0	See CoreSight Architecture Specification	RO	0x21

M33: DWT_PIDR1 Register

Offset: 0x01fe4

Description

Provides CoreSight discovery information for the DWT

Table 157.
DWT_PIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DES_0	See CoreSight Architecture Specification	RO	0xb
3:0	PART_1	See CoreSight Architecture Specification	RO	0xd

M33: DWT_PIDR2 Register

Offset: 0x01fe8

Description

Provides CoreSight discovery information for the DWT

Table 158.
DWT_PIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	See CoreSight Architecture Specification	RO	0x0
3	JEDEC	See CoreSight Architecture Specification	RO	0x1
2:0	DES_1	See CoreSight Architecture Specification	RO	0x3

M33: DWT_PIDR3 Register

Offset: 0x01fec

Description

Provides CoreSight discovery information for the DWT

Table 159.
DWT_PIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVAND	See CoreSight Architecture Specification	RO	0x0
3:0	CMOD	See CoreSight Architecture Specification	RO	0x0

M33: DWT_CIDR0 Register

Offset: 0x01ff0

Description

Provides CoreSight discovery information for the DWT

Table 160.
DWT_CIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_0	See CoreSight Architecture Specification	RO	0x0d

M33: DWT_CIDR1 Register

Offset: 0x01ff4

Description

Provides CoreSight discovery information for the DWT

Table 161.
DWT_CIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	CLASS	See CoreSight Architecture Specification	RO	0x9
3:0	PRMBL_1	See CoreSight Architecture Specification	RO	0x0

M33: DWT_CIDR2 Register

Offset: 0x01ff8

Description

Provides CoreSight discovery information for the DWT

Table 162.
DWT_CIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_2	See CoreSight Architecture Specification	RO	0x05

M33: DWT_CIDR3 Register

Offset: 0x01ffc

Description

Provides CoreSight discovery information for the DWT

Table 163.
DWT_CIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_3	See CoreSight Architecture Specification	RO	0xb1

M33: FP_CTRL Register

Offset: 0x02000

Description

Provides FPB implementation information, and the global enable for the FPB unit

Table 164. FP_CTRL Register

Bits	Name	Description	Type	Reset
31:28	REV	Flash Patch and Breakpoint Unit architecture revision	RO	0x6
27:15	Reserved.	-	-	-
14:12	NUM_CODE_14_1_2	Indicates the number of implemented instruction address comparators. Zero indicates no Instruction Address comparators are implemented. The Instruction Address comparators are numbered from 0 to NUM_CODE - 1	RO	0x5
11:8	NUM_LIT	Indicates the number of implemented literal address comparators. The Literal Address comparators are numbered from NUM_CODE to NUM_CODE + NUM_LIT - 1	RO	0x5
7:4	NUM_CODE_7_4_	Indicates the number of implemented instruction address comparators. Zero indicates no Instruction Address comparators are implemented. The Instruction Address comparators are numbered from 0 to NUM_CODE - 1	RO	0x8
3:2	Reserved.	-	-	-
1	KEY	Writes to the FP_CTRL are ignored unless KEY is concurrently written to one	RW	0x0
0	ENABLE	Enables the FPB	RW	0x0

M33: FP_REMAP Register

Offset: 0x02004

Description

Indicates whether the implementation supports Flash Patch remap and, if it does, holds the target address for remap

Table 165. FP_REMAP Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29	RMPSP	Indicates whether the FPB unit supports the Flash Patch remap function	RO	0x0
28:5	REMAP	Holds the bits[28:5] of the Flash Patch remap address	RO	0x000000
4:0	Reserved.	-	-	-

M33: FP_COMP0, FP_COMP1, ..., FP_COMP6, FP_COMP7 Registers

Offsets: 0x02008, 0x0200c, ..., 0x02020, 0x02024

Description

Holds an address for comparison. The effect of the match depends on the configuration of the FPB and whether the comparator is an instruction address comparator or a literal address comparator

Table 166. FP_COMP0, FP_COMP1, ..., FP_COMP6, FP_COMP7 Registers

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	BE	Selects between flashpatch and breakpoint functionality	RW	0x0

M33: FP_DEVARCH Register

Offset: 0x02fbc

Description

Provides CoreSight discovery information for the FPB

Table 167. FP_DEVARCH Register

Bits	Name	Description	Type	Reset
31:21	ARCHITECT	Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.	RO	0x23b
20	PRESENT	Defines that the DEVARCH register is present	RO	0x1
19:16	REVISION	Defines the architecture revision of the component	RO	0x0
15:12	ARCHVER	Defines the architecture version of the component	RO	0x1
11:0	ARCHPART	Defines the architecture of the component	RO	0xa03

M33: FP_DEVTYP Register

Offset: 0x02fcc

Description

Provides CoreSight discovery information for the FPB

Table 168. FP_DEVTYP Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SUB	Component sub-type	RO	0x0
3:0	MAJOR	Component major type	RO	0x0

M33: FP_PIDR4 Register

Offset: 0x02fd0

Description

Provides CoreSight discovery information for the FP

Table 169. FP_PIDR4
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SIZE	See CoreSight Architecture Specification	RO	0x0
3:0	DES_2	See CoreSight Architecture Specification	RO	0x4

M33: FP_PIDR5 Register

Offset: 0x02fd4

Description

Provides CoreSight discovery information for the FP

Table 170. FP_PIDR5
Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: FP_PIDR6 Register

Offset: 0x02fd8

Description

Provides CoreSight discovery information for the FP

Table 171. FP_PIDR6
Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: FP_PIDR7 Register

Offset: 0x02fdc

Description

Provides CoreSight discovery information for the FP

Table 172. FP_PIDR7
Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: FP_PIDR0 Register

Offset: 0x02fe0

Description

Provides CoreSight discovery information for the FP

Table 173. FP_PIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PART_0	See CoreSight Architecture Specification	RO	0x21

M33: FP_PIDR1 Register

Offset: 0x02fe4

Description

Provides CoreSight discovery information for the FP

Table 174. FP_PIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DES_0	See CoreSight Architecture Specification	RO	0xb
3:0	PART_1	See CoreSight Architecture Specification	RO	0xd

M33: FP_PIDR2 Register

Offset: 0x02fe8

Description

Provides CoreSight discovery information for the FP

Table 175. FP_PIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	See CoreSight Architecture Specification	RO	0x0
3	JEDEC	See CoreSight Architecture Specification	RO	0x1
2:0	DES_1	See CoreSight Architecture Specification	RO	0x3

M33: FP_PIDR3 Register

Offset: 0x02fec

Description

Provides CoreSight discovery information for the FP

Table 176. FP_PIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVAND	See CoreSight Architecture Specification	RO	0x0
3:0	CMOD	See CoreSight Architecture Specification	RO	0x0

M33: FP_CIDR0 Register

Offset: 0x02ff0

Description

Provides CoreSight discovery information for the FP

Table 177. FP_CIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_0	See CoreSight Architecture Specification	RO	0x0d

M33: FP_CIDR1 Register

Offset: 0x02ff4

Description

Provides CoreSight discovery information for the FP

Table 178. FP_CIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	CLASS	See CoreSight Architecture Specification	RO	0x9
3:0	PRMBL_1	See CoreSight Architecture Specification	RO	0x0

M33: FP_CIDR2 Register

Offset: 0x02ff8

Description

Provides CoreSight discovery information for the FP

Table 179. FP_CIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_2	See CoreSight Architecture Specification	RO	0x05

M33: FP_CIDR3 Register

Offset: 0x02ffc

Description

Provides CoreSight discovery information for the FP

Table 180. FP_CIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_3	See CoreSight Architecture Specification	RO	0xb1

M33: ICTR Register

Offset: 0x0e004

Description

Provides information about the interrupt controller

Table 181. ICTR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	INTLINESNUM	Indicates the number of the highest implemented register in each of the NVIC control register sets, or in the case of NVIC_IPR*n, 4×INTLINESNUM	RO	0x1

M33: ACTLR Register

Offset: 0x0e008

Description

Provides IMPLEMENTATION DEFINED configuration and control options

Table 182. ACTLR Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29	EXTEXCLALL	External Exclusives Allowed with no MPU	RW	0x0
28:13	Reserved.	-	-	-
12	DISITMATBFLUSH	Disable ATB Flush	RW	0x0
11	Reserved.	-	-	-
10	FPEXCODIS	Disable FPU exception outputs	RW	0x0
9	DISOOFP	Disable out-of-order FP instruction completion	RW	0x0
8:3	Reserved.	-	-	-
2	DISFOLD	Disable dual-issue.	RW	0x0
1	Reserved.	-	-	-
0	DISMCYCINT	Disable dual-issue.	RW	0x0

M33: SYST_CSR Register

Offset: 0x0e010

Description

Use the SysTick Control and Status Register to enable the SysTick features.

Table 183. SYST_CSR Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read. Clears on read by application or debugger.	RO	0x0
15:3	Reserved.	-	-	-
2	CLKSOURCE	SysTick clock source. Always reads as one if SYST_CALIB reports NOREF. Selects the SysTick timer clock source: 0 = External reference clock. 1 = Processor clock.	RW	0x0

Bits	Name	Description	Type	Reset
1	TICKINT	Enables SysTick exception request: 0 = Counting down to zero does not assert the SysTick exception request. 1 = Counting down to zero asserts the SysTick exception request.	RW	0x0
0	ENABLE	Enable SysTick counter: 0 = Counter disabled. 1 = Counter enabled.	RW	0x0

M33: SYST_RVR Register

Offset: 0x0e014

Description

Use the SysTick Reload Value Register to specify the start value to load into the current value register when the counter reaches 0. It can be any value between 0 and 0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick interrupt and COUNTFLAG are activated when counting from 1 to 0. The reset value of this register is UNKNOWN.

To generate a multi-shot timer with a period of N processor clock cycles, use a RELOAD value of N-1. For example, if the SysTick interrupt is required every 100 clock pulses, set RELOAD to 99.

Table 184. SYST_RVR Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:0	RELOAD	Value to load into the SysTick Current Value Register when the counter reaches 0.	RW	0x000000

M33: SYST_CVR Register

Offset: 0x0e018

Description

Use the SysTick Current Value Register to find the current value in the register. The reset value of this register is UNKNOWN.

Table 185. SYST_CVR Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:0	CURRENT	Reads return the current value of the SysTick counter. This register is write-clear. Writing to it with any value clears the register to 0. Clearing this register also clears the COUNTFLAG bit of the SysTick Control and Status Register.	RW	0x000000

M33: SYST_CALIB Register

Offset: 0x0e01c

Description

Use the SysTick Calibration Value Register to enable software to scale to any required speed using divide and multiply.

Table 186.
SYST_CALIB Register

Bits	Name	Description	Type	Reset
31	NOREF	If reads as 1, the Reference clock is not provided - the CLKSOURCE bit of the SysTick Control and Status register will be forced to 1 and cannot be cleared to 0.	RO	0x0
30	SKEW	If reads as 1, the calibration value for 10ms is inexact (due to clock frequency).	RO	0x0
29:24	Reserved.	-	-	-
23:0	TENMS	An optional Reload value to be used for 10ms (100Hz) timing, subject to system clock skew errors. If the value reads as 0, the calibration value is not known.	RO	0x000000

M33: NVIC_ISER0, NVIC_ISER1 Registers

Offsets: 0x0e100, 0x0e104

Description

Enables or reads the enabled state of each group of 32 interrupts

Table 187.
NVIC_ISER0,
NVIC_ISER1 Registers

Bits	Name	Description	Type	Reset
31:0	SETENA	For SETENA[m] in NVIC_ISER*n, indicates whether interrupt 32*n + m is enabled	RW	0x00000000

M33: NVIC_ICER0, NVIC_ICER1 Registers

Offsets: 0x0e180, 0x0e184

Description

Clears or reads the enabled state of each group of 32 interrupts

Table 188.
NVIC_ICER0,
NVIC_ICER1 Registers

Bits	Name	Description	Type	Reset
31:0	CLRENA	For CLRENA[m] in NVIC_ICER*n, indicates whether interrupt 32*n + m is enabled	RW	0x00000000

M33: NVIC_ISPR0, NVIC_ISPR1 Registers

Offsets: 0x0e200, 0x0e204

Description

Enables or reads the pending state of each group of 32 interrupts

Table 189.
NVIC_ISPR0,
NVIC_ISPR1 Registers

Bits	Name	Description	Type	Reset
31:0	SETPEND	For SETPEND[m] in NVIC_ISPR*n, indicates whether interrupt 32*n + m is pending	RW	0x00000000

M33: NVIC_ICPR0, NVIC_ICPR1 Registers

Offsets: 0x0e280, 0x0e284

Description

Clears or reads the pending state of each group of 32 interrupts

Table 190.
NVIC_ICP0,
NVIC_ICP1 Registers

Bits	Name	Description	Type	Reset
31:0	CLRPEND	For CLRPEND[m] in NVIC_ICPR*n, indicates whether interrupt $32*n + m$ is pending	RW	0x00000000

M33: NVIC_IABR0, NVIC_IABR1 Registers

Offsets: 0x0e300, 0x0e304

Description

For each group of 32 interrupts, shows the active state of each interrupt

Table 191.
NVIC_IABR0,
NVIC_IABR1 Registers

Bits	Name	Description	Type	Reset
31:0	ACTIVE	For ACTIVE[m] in NVIC_IABR*n, indicates the active state for interrupt $32*n+m$	RW	0x00000000

M33: NVIC_ITNS0, NVIC_ITNS1 Registers

Offsets: 0x0e380, 0x0e384

Description

For each group of 32 interrupts, determines whether each interrupt targets Non-secure or Secure state

Table 192.
NVIC_ITNS0,
NVIC_ITNS1 Registers

Bits	Name	Description	Type	Reset
31:0	ITNS	For ITNS[m] in NVIC_ITNS*n, `IAAMO the target Security state for interrupt $32*n+m$	RW	0x00000000

M33: NVIC_IPR0, NVIC_IPR1, ..., NVIC_IPR14, NVIC_IPR15 Registers

Offsets: 0x0e400, 0x0e404, ..., 0x0e438, 0x0e43c

Description

Sets or reads interrupt priorities

Table 193. NVIC_IPR0,
NVIC_IPR1, ...,
NVIC_IPR14
NVIC_IPR15 Registers

Bits	Name	Description	Type	Reset
31:28	PRI_N3	For register NVIC_IPRn, the priority of interrupt number $4*n+3$, or RES0 if the PE does not implement this interrupt	RW	0x0
27:24	Reserved.	-	-	-
23:20	PRI_N2	For register NVIC_IPRn, the priority of interrupt number $4*n+2$, or RES0 if the PE does not implement this interrupt	RW	0x0
19:16	Reserved.	-	-	-
15:12	PRI_N1	For register NVIC_IPRn, the priority of interrupt number $4*n+1$, or RES0 if the PE does not implement this interrupt	RW	0x0
11:8	Reserved.	-	-	-
7:4	PRI_N0	For register NVIC_IPRn, the priority of interrupt number $4*n+0$, or RES0 if the PE does not implement this interrupt	RW	0x0
3:0	Reserved.	-	-	-

M33: CPUD Register

Offset: 0x0ed00

Description

Provides identification information for the PE, including an implementer code for the device and a device ID number

Table 194. CPUID Register

Bits	Name	Description	Type	Reset
31:24	IMPLEMENTER	This field must hold an implementer code that has been assigned by ARM	RO	0x41
23:20	VARIANT	IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product	RO	0x1
19:16	ARCHITECTURE	Defines the Architecture implemented by the PE	RO	0xf
15:4	PARTNO	IMPLEMENTATION DEFINED primary part number for the device	RO	0xd21
3:0	REVISION	IMPLEMENTATION DEFINED revision number for the device	RO	0x0

M33: ICSR Register

Offset: 0x0ed04

Description

Controls and provides status information for NMI, PendSV, SysTick and interrupts

Table 195. ICSR Register

Bits	Name	Description	Type	Reset
31	PENDNMISET	Indicates whether the NMI exception is pending	RO	0x0
30	PENDNMICLR	Allows the NMI exception pend state to be cleared	RW	0x0
29	Reserved.	-	-	-
28	PENDSVSET	Indicates whether the PendSV `FTSSS exception is pending	RO	0x0
27	PENDSVCLR	Allows the PendSV exception pend state to be cleared `FTSSS	RW	0x0
26	PENDSTSET	Indicates whether the SysTick `FTSSS exception is pending	RO	0x0
25	PENDSTCLR	Allows the SysTick exception pend state to be cleared `FTSSS	RW	0x0
24	STTNS	Controls whether in a single SysTick implementation, the SysTick is Secure or Non-secure	RW	0x0
23	ISRPREEMPT	Indicates whether a pending exception will be serviced on exit from debug halt state	RO	0x0
22	ISRPENDING	Indicates whether an external interrupt, generated by the NVIC, is pending	RO	0x0
21	Reserved.	-	-	-
20:12	VECTPENDING	The exception number of the highest priority pending and enabled interrupt	RO	0x000
11	RETTOBASE	In Handler mode, indicates whether there is more than one active exception	RO	0x0
10:9	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
8:0	VECTACTIVE	The exception number of the current executing exception	RO	0x000

M33: VTOR Register

Offset: 0x0ed08

Description

The VTOR indicates the offset of the vector table base address from memory address 0x00000000.

Table 196. VTOR Register

Bits	Name	Description	Type	Reset
31:7	TBLOFF	Vector table base offset field. It contains bits[31:7] of the offset of the table base from the bottom of the memory map.	RW	0x0000000
6:0	Reserved.	-	-	-

M33: AIRCR Register

Offset: 0x0ed0c

Description

Use the Application Interrupt and Reset Control Register to: determine data endianness, clear all active state information from debug halt mode, request a system reset.

Table 197. AIRCR Register

Bits	Name	Description	Type	Reset
31:16	VECTKEY	Register key: Reads as Unknown On writes, write 0x05FA to VECTKEY, otherwise the write is ignored.	RW	0x0000
15	_ENDIANESS	Data endianness implemented: 0 = Little-endian.	RO	0x0
14	PRIS	Prioritize Secure exceptions. The value of this bit defines whether Secure exception priority boosting is enabled. 0 Priority ranges of Secure and Non-secure exceptions are identical. 1 Non-secure exceptions are de-prioritized.	RW	0x0
13	BFHFNMINS	BusFault, HardFault, and NMI Non-secure enable. 0 BusFault, HardFault, and NMI are Secure. 1 BusFault and NMI are Non-secure and exceptions can target Non-secure HardFault.	RW	0x0
12:11	Reserved.	-	-	-
10:8	PRIGROUP	Interrupt priority grouping field. This field determines the split of group priority from subpriority. See https://developer.arm.com/documentation/100235/0004/the-cortex-m33-peripherals/system-control-block/application-interrupt-and-reset-control-register?lang=en	RW	0x0
7:4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3	SYSRESETREQS	System reset request, Secure state only. 0 SYSRESETREQ functionality is available to both Security states. 1 SYSRESETREQ functionality is only available to Secure state.	RW	0x0
2	SYSRESETREQ	Writing 1 to this bit causes the SYSRESETREQ signal to the outer system to be asserted to request a reset. The intention is to force a large system reset of all major components except for debug. The C_HALT bit in the DHCSR is cleared as a result of the system reset requested. The debugger does not lose contact with the device.	RW	0x0
1	VECTCLRACTIVE	Clears all active state information for fixed and configurable exceptions. This bit: is self-clearing, can only be set by the DAP when the core is halted. When set: clears all active exception status of the processor, forces a return to Thread mode, forces an IPSR of 0. A debugger must re-initialize the stack.	RW	0x0
0	Reserved.	-	-	-

M33: SCR Register

Offset: 0x0ed10

Description

System Control Register. Use the System Control Register for power-management functions: signal to the system when the processor can enter a low power state, control how the processor enters and exits low power states.

Table 198. SCR Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	SEVONPEND	Send Event on Pending bit: 0 = Only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded. 1 = Enabled events and all interrupts, including disabled interrupts, can wakeup the processor. When an event or interrupt becomes pending, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an SEV instruction or an external event.	RW	0x0
3	SLEEPDEEPS	0 SLEEPDEEP is available to both security states 1 SLEEPDEEP is only available to Secure state	RW	0x0
2	SLEEPDEEP	Controls whether the processor uses sleep or deep sleep as its low power mode: 0 = Sleep. 1 = Deep sleep.	RW	0x0

Bits	Name	Description	Type	Reset
1	SLEEPONEXIT	Indicates sleep-on-exit when returning from Handler mode to Thread mode: 0 = Do not sleep when returning to Thread mode. 1 = Enter sleep, or deep sleep, on return from an ISR to Thread mode. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.	RW	0x0
0	Reserved.	-	-	-

M33: CCR Register

Offset: 0x0ed14

Description

Sets or returns configuration and control data

Table 199. CCR Register

Bits	Name	Description	Type	Reset
31:19	Reserved.	-	-	-
18	BP	Enables program flow prediction `FTSSS	RO	0x0
17	IC	This is a global enable bit for instruction caches in the selected Security state	RO	0x0
16	DC	Enables data caching of all data accesses to Normal memory `FTSSS	RO	0x0
15:11	Reserved.	-	-	-
10	STKOFHFNIGN	Controls the effect of a stack limit violation while executing at a requested priority less than 0	RW	0x0
9	RES1	Reserved, RES1	RO	0x1
8	BFHFNIGN	Determines the effect of precise BusFaults on handlers running at a requested priority less than 0	RW	0x0
7:5	Reserved.	-	-	-
4	DIV_0_TRP	Controls the generation of a DIVBYZERO UsageFault when attempting to perform integer division by zero	RW	0x0
3	UNALIGN_TRP	Controls the trapping of unaligned word or halfword accesses	RW	0x0
2	Reserved.	-	-	-
1	USERSETMPEND	Determines whether unprivileged accesses are permitted to pend interrupts via the STIR	RW	0x0
0	RES1_1	Reserved, RES1	RO	0x1

M33: SHPR1 Register

Offset: 0x0ed18

Description

Sets or returns priority for system handlers 4 - 7

Table 200. SHPR1 Register

Bits	Name	Description	Type	Reset
31:29	PRI_7_3	Priority of system handler 7, SecureFault	RW	0x0
28:24	Reserved.	-	-	-
23:21	PRI_6_3	Priority of system handler 6, SecureFault	RW	0x0
20:16	Reserved.	-	-	-
15:13	PRI_5_3	Priority of system handler 5, SecureFault	RW	0x0
12:8	Reserved.	-	-	-
7:5	PRI_4_3	Priority of system handler 4, SecureFault	RW	0x0
4:0	Reserved.	-	-	-

M33: SHPR2 Register

Offset: 0x0ed1c

Description

Sets or returns priority for system handlers 8 - 11

Table 201. SHPR2 Register

Bits	Name	Description	Type	Reset
31:29	PRI_11_3	Priority of system handler 11, SecureFault	RW	0x0
28:24	Reserved.	-	-	-
23:16	PRI_10	Reserved, RES0	RO	0x00
15:8	PRI_9	Reserved, RES0	RO	0x00
7:0	PRI_8	Reserved, RES0	RO	0x00

M33: SHPR3 Register

Offset: 0x0ed20

Description

Sets or returns priority for system handlers 12 - 15

Table 202. SHPR3 Register

Bits	Name	Description	Type	Reset
31:29	PRI_15_3	Priority of system handler 15, SecureFault	RW	0x0
28:24	Reserved.	-	-	-
23:21	PRI_14_3	Priority of system handler 14, SecureFault	RW	0x0
20:16	Reserved.	-	-	-
15:8	PRI_13	Reserved, RES0	RO	0x00
7:5	PRI_12_3	Priority of system handler 12, SecureFault	RW	0x0
4:0	Reserved.	-	-	-

M33: SHCSR Register

Offset: 0x0ed24

Description

Provides access to the active and pending status of system exceptions

Table 203. SHCSR Register

Bits	Name	Description	Type	Reset
31:22	Reserved.	-	-	-
21	HARDFAULTPEND	`IAAMO the pending state of the HardFault exception `CTTSSS	RW	0x0
20	SECUREFAULTPEN DED	`IAAMO the pending state of the SecureFault exception	RW	0x0
19	SECUREFAULTEN A	`DW the SecureFault exception is enabled	RW	0x0
18	USGFAULTENA	`DW the UsageFault exception is enabled `FTSSS	RW	0x0
17	BUSFAULTENA	`DW the BusFault exception is enabled	RW	0x0
16	MEMFAULTENA	`DW the MemManage exception is enabled `FTSSS	RW	0x0
15	SVCALLPENDED	`IAAMO the pending state of the SVCall exception `FTSSS	RW	0x0
14	BUSFAULTPENDE D	`IAAMO the pending state of the BusFault exception	RW	0x0
13	MEMFAULTPEN ED	`IAAMO the pending state of the MemManage exception `FTSSS	RW	0x0
12	USGFAULTPENDE D	The UsageFault exception is banked between Security states, `IAAMO the pending state of the UsageFault exception `FTSSS	RW	0x0
11	SYSTICKACT	`IAAMO the active state of the SysTick exception `FTSSS	RW	0x0
10	PENDSVACT	`IAAMO the active state of the PendSV exception `FTSSS	RW	0x0
9	Reserved.	-	-	-
8	MONITORACT	`IAAMO the active state of the DebugMonitor exception	RW	0x0
7	SVCALLACT	`IAAMO the active state of the SVCall exception `FTSSS	RW	0x0
6	Reserved.	-	-	-
5	NMIACT	`IAAMO the active state of the NMI exception	RW	0x0
4	SECUREFAULTAC T	`IAAMO the active state of the SecureFault exception	RW	0x0
3	USGFAULTACT	`IAAMO the active state of the UsageFault exception `FTSSS	RW	0x0
2	HARDFAULTACT	Indicates and allows limited modification of the active state of the HardFault exception `FTSSS	RW	0x0
1	BUSFAULTACT	`IAAMO the active state of the BusFault exception	RW	0x0
0	MEMFAULTACT	`IAAMO the active state of the MemManage exception `FTSSS	RW	0x0

M33: CFSR Register

Offset: 0x0ed28

Description

Contains the three Configurable Fault Status Registers.

31:16 UFSR: Provides information on UsageFault exceptions

15:8 BCSR: Provides information on BusFault exceptions

7:0 MMFSR: Provides information on MemManage exceptions

Table 204. CFSR Register

Bits	Name	Description	Type	Reset
31:26	Reserved.	-	-	-
25	UFSR_DIVBYZERO	Sticky flag indicating whether an integer division by zero error has occurred	RW	0x0
24	UFSR_UNALIGNED	Sticky flag indicating whether an unaligned access error has occurred	RW	0x0
23:21	Reserved.	-	-	-
20	UFSR_STKOF	Sticky flag indicating whether a stack overflow error has occurred	RW	0x0
19	UFSR_NOCP	Sticky flag indicating whether a coprocessor disabled or not present error has occurred	RW	0x0
18	UFSR_INVPC	Sticky flag indicating whether an integrity check error has occurred	RW	0x0
17	UFSR_INVSTATE	Sticky flag indicating whether an EPSR.T or EPSR.IT validity error has occurred	RW	0x0
16	UFSR_UNDEFINST	Sticky flag indicating whether an undefined instruction error has occurred	RW	0x0
15	BFSR_BFARVALID	Indicates validity of the contents of the BFAR register	RW	0x0
14	Reserved.	-	-	-
13	BFSR_LSPERR	Records whether a BusFault occurred during FP lazy state preservation	RW	0x0
12	BFSR_STKERR	Records whether a derived BusFault occurred during exception entry stacking	RW	0x0
11	BFSR_UNSTKERR	Records whether a derived BusFault occurred during exception return unstacking	RW	0x0
10	BFSR_IMPRECISERR	Records whether an imprecise data access error has occurred	RW	0x0
9	BFSR_PRECISERR	Records whether a precise data access error has occurred	RW	0x0
8	BFSR_IBUSERR	Records whether a BusFault on an instruction prefetch has occurred	RW	0x0
7:0	MMFSR	Provides information on MemManage exceptions	RW	0x00

M33: HFSR Register

Offset: 0x0ed2c

Description

Shows the cause of any HardFaults

Table 205. HFSR Register

Bits	Name	Description	Type	Reset
31	DEBUGEVT	Indicates when a Debug event has occurred	RW	0x0
30	FORCED	Indicates that a fault with configurable priority has been escalated to a HardFault exception, because it could not be made active, because of priority, or because it was disabled	RW	0x0
29:2	Reserved.	-	-	-
1	VECTTBL	Indicates when a fault has occurred because of a vector table read error on exception processing	RW	0x0
0	Reserved.	-	-	-

M33: DFSR Register

Offset: 0x0ed30

Description

Shows which debug event occurred

Table 206. DFSR Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	EXTERNAL	Sticky flag indicating whether an External debug request debug event has occurred	RW	0x0
3	VCATCH	Sticky flag indicating whether a Vector catch debug event has occurred	RW	0x0
2	DWTTRAP	Sticky flag indicating whether a Watchpoint debug event has occurred	RW	0x0
1	BKPT	Sticky flag indicating whether a Breakpoint debug event has occurred	RW	0x0
0	HALTED	Sticky flag indicating that a Halt request debug event or Step debug event has occurred	RW	0x0

M33: MMFAR Register

Offset: 0x0ed34

Description

Shows the address of the memory location that caused an MPU fault

Table 207. MMFAR Register

Bits	Name	Description	Type	Reset
31:0	ADDRESS	This register is updated with the address of a location that produced a MemManage fault. The MMFSR shows the cause of the fault, and whether this field is valid. This field is valid only when MMFSR.MMARVALID is set, otherwise it is UNKNOWN	RW	0x00000000

M33: BFAR Register

Offset: 0x0ed38

Description

Shows the address associated with a precise data access BusFault

Table 208. BFAR Register

Bits	Name	Description	Type	Reset
31:0	ADDRESS	This register is updated with the address of a location that produced a BusFault. The BFSR shows the reason for the fault. This field is valid only when BFSR.BFARVALID is set, otherwise it is UNKNOWN	RW	0x00000000

M33: ID_PFR0 Register

Offset: 0x0ed40

Description

Gives top-level information about the instruction set supported by the PE

Table 209. ID_PFR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	STATE1	T32 instruction set support	RO	0x3
3:0	STATE0	A32 instruction set support	RO	0x0

M33: ID_PFR1 Register

Offset: 0x0ed44

Description

Gives information about the programmers' model and Extensions support

Table 210. ID_PFR1 Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:8	MPROGMOD	Identifies support for the M-Profile programmers' model support	RO	0x5
7:4	SECURITY	Identifies whether the Security Extension is implemented	RO	0x2
3:0	Reserved.	-	-	-

M33: ID_DFR0 Register

Offset: 0x0ed48

Description

Provides top level information about the debug system

Table 211. ID_DFR0 Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:20	MPROFDBG	Indicates the supported M-profile debug architecture	RO	0x2
19:0	Reserved.	-	-	-

M33: ID_AFR0 Register

Offset: 0x0ed4c

Description

Provides information about the IMPLEMENTATION DEFINED features of the PE

Table 212. ID_AFR0 Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:12	IMPDEF3	IMPLEMENTATION DEFINED meaning	RO	0x0
11:8	IMPDEF2	IMPLEMENTATION DEFINED meaning	RO	0x0
7:4	IMPDEF1	IMPLEMENTATION DEFINED meaning	RO	0x0
3:0	IMPDEF0	IMPLEMENTATION DEFINED meaning	RO	0x0

M33: ID_MMFR0 Register

Offset: 0x0ed50

Description

Provides information about the implemented memory model and memory management support

Table 213. ID_MMFR0 Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:20	AUXREG	Indicates support for Auxiliary Control Registers	RO	0x1
19:16	TCM	Indicates support for tightly coupled memories (TCMs)	RO	0x0
15:12	SHARELVL	Indicates the number of shareability levels implemented	RO	0x1
11:8	OUTERSHR	Indicates the outermost shareability domain implemented	RO	0xf
7:4	PMSA	Indicates support for the protected memory system architecture (PMSA)	RO	0x4
3:0	Reserved.	-	-	-

M33: ID_MMFR1 Register

Offset: 0x0ed54

Description

Provides information about the implemented memory model and memory management support

Table 214. ID_MMFR1 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: ID_MMFR2 Register

Offset: 0x0ed58

Description

Provides information about the implemented memory model and memory management support

Table 215. ID_MMFR2 Register

Bits	Name	Description	Type	Reset
31:28	Reserved.	-	-	-
27:24	WFISTALL	Indicates the support for Wait For Interrupt (WFI) stalling	RO	0x1
23:0	Reserved.	-	-	-

M33: ID_MMFR3 Register

Offset: 0x0ed5c

Description

Provides information about the implemented memory model and memory management support

Table 216. ID_MMFR3 Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:8	BPMAINT	Indicates the supported branch predictor maintenance	RO	0x0
7:4	CMAINTSW	Indicates the supported cache maintenance operations by set/way	RO	0x0
3:0	CMAINTVA	Indicates the supported cache maintenance operations by address	RO	0x0

M33: ID_ISAR0 Register

Offset: 0x0ed60

Description

Provides information about the instruction set implemented by the PE

Table 217. ID_ISAR0 Register

Bits	Name	Description	Type	Reset
31:28	Reserved.	-	-	-
27:24	DIVIDE	Indicates the supported Divide instructions	RO	0x8
23:20	DEBUG	Indicates the implemented Debug instructions	RO	0x0
19:16	COPROC	Indicates the supported Coprocessor instructions	RO	0x9
15:12	CMPBRANCH	Indicates the supported combined Compare and Branch instructions	RO	0x2
11:8	BITFIELD	Indicates the supported bit field instructions	RO	0x3
7:4	BITCOUNT	Indicates the supported bit count instructions	RO	0x0
3:0	Reserved.	-	-	-

M33: ID_ISAR1 Register

Offset: 0x0ed64

Description

Provides information about the instruction set implemented by the PE

Table 218. ID_ISAR1 Register

Bits	Name	Description	Type	Reset
31:28	Reserved.	-	-	-
27:24	INTERWORK	Indicates the implemented Interworking instructions	RO	0x5
23:20	IMMEDIATE	Indicates the implemented for data-processing instructions with long immediates	RO	0x7
19:16	IFTHEN	Indicates the implemented If-Then instructions	RO	0x2
15:12	EXTEND	Indicates the implemented Extend instructions	RO	0x5

Bits	Name	Description	Type	Reset
11:0	Reserved.	-	-	-

M33: ID_ISAR2 Register

Offset: 0x0ed68

Description

Provides information about the instruction set implemented by the PE

Table 219. ID_ISAR2 Register

Bits	Name	Description	Type	Reset
31:28	REVERSAL	Indicates the implemented Reversal instructions	RO	0x3
27:24	Reserved.	-	-	-
23:20	MULTU	Indicates the implemented advanced unsigned Multiply instructions	RO	0x1
19:16	MULTS	Indicates the implemented advanced signed Multiply instructions	RO	0x7
15:12	MULT	Indicates the implemented additional Multiply instructions	RO	0x3
11:8	MULTIACCESSINT	Indicates the support for interruptible multi-access instructions	RO	0x4
7:4	MEMHINT	Indicates the implemented Memory Hint instructions	RO	0x2
3:0	LOADSTORE	Indicates the implemented additional load/store instructions	RO	0x6

M33: ID_ISAR3 Register

Offset: 0x0ed6c

Description

Provides information about the instruction set implemented by the PE

Table 220. ID_ISAR3 Register

Bits	Name	Description	Type	Reset
31:28	Reserved.	-	-	-
27:24	TRUE NOP	Indicates the implemented true NOP instructions	RO	0x7
23:20	T32COPY	Indicates the support for T32 non flag-setting MOV instructions	RO	0x8
19:16	TABBRANCH	Indicates the implemented Table Branch instructions	RO	0x9
15:12	SYNCHPRIM	Used in conjunction with ID_ISAR4.SynchPrim_frc to indicate the implemented Synchronization Primitive instructions	RO	0x5
11:8	SVC	Indicates the implemented SVC instructions	RO	0x7
7:4	SIMD	Indicates the implemented SIMD instructions	RO	0x2
3:0	SATURATE	Indicates the implemented saturating instructions	RO	0x9

M33: ID_ISAR4 Register

Offset: 0x0ed70

Description

Provides information about the instruction set implemented by the PE

Table 221. ID_ISAR4 Register

Bits	Name	Description	Type	Reset
31:28	Reserved.	-	-	-
27:24	PSR_M	Indicates the implemented M profile instructions to modify the PSRs	RO	0x1
23:20	SYNCPRIM_FRAC	Used in conjunction with ID_ISAR3.SynchPrim to indicate the implemented Synchronization Primitive instructions	RO	0x3
19:16	BARRIER	Indicates the implemented Barrier instructions	RO	0x1
15:12	Reserved.	-	-	-
11:8	WRITEBACK	Indicates the support for writeback addressing modes	RO	0x1
7:4	WITHSHIFTS	Indicates the support for writeback addressing modes	RO	0x3
3:0	UNPRIV	Indicates the implemented unprivileged instructions	RO	0x2

M33: ID_ISAR5 Register

Offset: 0x0ed74

Description

Provides information about the instruction set implemented by the PE

Table 222. ID_ISAR5 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: CTR Register

Offset: 0x0ed7c

Description

Provides information about the architecture of the caches. CTR is RES0 if CLIDR is zero.

Table 223. CTR Register

Bits	Name	Description	Type	Reset
31	RES1	Reserved, RES1	RO	0x1
30:28	Reserved.	-	-	-
27:24	CWG	Log2 of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified	RO	0x0
23:20	ERG	Log2 of the number of words of the maximum size of the reservation granule that has been implemented for the Load-Exclusive and Store-Exclusive instructions	RO	0x0
19:16	DINLINE	Log2 of the number of words in the smallest cache line of all the data caches and unified caches that are controlled by the PE	RO	0x0
15:14	RES1_1	Reserved, RES1	RO	0x3
13:4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3:0	IMINLINE	Log2 of the number of words in the smallest cache line of all the instruction caches that are controlled by the PE	RO	0x0

M33: CPACR Register

Offset: 0x0ed88

Description

Specifies the access privileges for coprocessors and the FP Extension

Table 224. CPACR Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:22	CP11	The value in this field is ignored. If the implementation does not include the FP Extension, this field is RAZ/WI. If the value of this bit is not programmed to the same value as the CP10 field, then the value is UNKNOWN	RW	0x0
21:20	CP10	Defines the access rights for the floating-point functionality	RW	0x0
19:16	Reserved.	-	-	-
15:14	CP7	Controls access privileges for coprocessor 7	RW	0x0
13:12	CP6	Controls access privileges for coprocessor 6	RW	0x0
11:10	CP5	Controls access privileges for coprocessor 5	RW	0x0
9:8	CP4	Controls access privileges for coprocessor 4	RW	0x0
7:6	CP3	Controls access privileges for coprocessor 3	RW	0x0
5:4	CP2	Controls access privileges for coprocessor 2	RW	0x0
3:2	CP1	Controls access privileges for coprocessor 1	RW	0x0
1:0	CP0	Controls access privileges for coprocessor 0	RW	0x0

M33: NSACR Register

Offset: 0x0ed8c

Description

Defines the Non-secure access permissions for both the FP Extension and coprocessors CP0 to CP7

Table 225. NSACR Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CP11	Enables Non-secure access to the Floating-point Extension	RW	0x0
10	CP10	Enables Non-secure access to the Floating-point Extension	RW	0x0
9:8	Reserved.	-	-	-
7	CP7	Enables Non-secure access to coprocessor CP7	RW	0x0
6	CP6	Enables Non-secure access to coprocessor CP6	RW	0x0
5	CP5	Enables Non-secure access to coprocessor CP5	RW	0x0

Bits	Name	Description	Type	Reset
4	CP4	Enables Non-secure access to coprocessor CP4	RW	0x0
3	CP3	Enables Non-secure access to coprocessor CP3	RW	0x0
2	CP2	Enables Non-secure access to coprocessor CP2	RW	0x0
1	CP1	Enables Non-secure access to coprocessor CP1	RW	0x0
0	CP0	Enables Non-secure access to coprocessor CP0	RW	0x0

M33: MPU_TYPE Register

Offset: 0x0ed90

Description

The MPU Type Register indicates how many regions the MPU `FTSSS supports

Table 226. MPU_TYPE Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:8	DREGION	Number of regions supported by the MPU	RO	0x08
7:1	Reserved.	-	-	-
0	SEPARATE	Indicates support for separate instructions and data address regions	RO	0x0

M33: MPU_CTRL Register

Offset: 0x0ed94

Description

Enables the MPU and, when the MPU is enabled, controls whether the default memory map is enabled as a background region for privileged accesses, and whether the MPU is enabled for HardFaults, NMIs, and exception handlers when FAULTMASK is set to 1

Table 227. MPU_CTRL Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	PRIVDEFENA	Controls whether the default memory map is enabled for privileged software	RW	0x0
1	HFNMIENA	Controls whether handlers executing with priority less than 0 access memory with the MPU enabled or disabled. This applies to HardFaults, NMIs, and exception handlers when FAULTMASK is set to 1	RW	0x0
0	ENABLE	Enables the MPU	RW	0x0

M33: MPU_RNR Register

Offset: 0x0ed98

Description

Selects the region currently accessed by MPU_RBAR and MPU_RLAR

Table 228. MPU_RNR Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
2:0	REGION	Indicates the memory region accessed by MPU_RBAR and MPU_RLAR	RW	0x0

M33: MPU_RBAR Register

Offset: 0x0ed9c

Description

Provides indirect read and write access to the base address of the currently selected MPU region `FTSSS

Table 229. MPU_RBAR Register

Bits	Name	Description	Type	Reset
31:5	BASE	Contains bits [31:5] of the lower inclusive limit of the selected MPU memory region. This value is zero extended to provide the base address to be checked against	RW	0x0000000
4:3	SH	Defines the Shareability domain of this region for Normal memory	RW	0x0
2:1	AP	Defines the access permissions for this region	RW	0x0
0	XN	Defines whether code can be executed from this region	RW	0x0

M33: MPU_RLAR Register

Offset: 0x0eda0

Description

Provides indirect read and write access to the limit address of the currently selected MPU region `FTSSS

Table 230. MPU_RLAR Register

Bits	Name	Description	Type	Reset
31:5	LIMIT	Contains bits [31:5] of the upper inclusive limit of the selected MPU memory region. This value is postfixed with 0x1F to provide the limit address to be checked against	RW	0x0000000
4	Reserved.	-	-	-
3:1	ATTRINDX	Associates a set of attributes in the MPU_MAIR0 and MPU_MAIR1 fields	RW	0x0
0	EN	Region enable	RW	0x0

M33: MPU_RBAR_A1 Register

Offset: 0x0eda4

Description

Provides indirect read and write access to the base address of the MPU region selected by MPU_RNR[7:2]:1[1:0] `FTSSS

Table 231. MPU_RBAR_A1 Register

Bits	Name	Description	Type	Reset
31:5	BASE	Contains bits [31:5] of the lower inclusive limit of the selected MPU memory region. This value is zero extended to provide the base address to be checked against	RW	0x0000000
4:3	SH	Defines the Shareability domain of this region for Normal memory	RW	0x0
2:1	AP	Defines the access permissions for this region	RW	0x0

Bits	Name	Description	Type	Reset
0	XN	Defines whether code can be executed from this region	RW	0x0

M33: MPU_RLAR_A1 Register

Offset: 0x0eda8

Description

Provides indirect read and write access to the limit address of the currently selected MPU region selected by MPU_RNR[7:2]:(1[1:0]) `FTSSS

Table 232.
MPU_RLAR_A1
Register

Bits	Name	Description	Type	Reset
31:5	LIMIT	Contains bits [31:5] of the upper inclusive limit of the selected MPU memory region. This value is postfixed with 0x1F to provide the limit address to be checked against	RW	0x0000000
4	Reserved.	-	-	-
3:1	ATTRINDX	Associates a set of attributes in the MPU_MAIR0 and MPU_MAIR1 fields	RW	0x0
0	EN	Region enable	RW	0x0

M33: MPU_RBAR_A2 Register

Offset: 0x0edac

Description

Provides indirect read and write access to the base address of the MPU region selected by MPU_RNR[7:2]:(2[1:0]) `FTSSS

Table 233.
MPU_RBAR_A2
Register

Bits	Name	Description	Type	Reset
31:5	BASE	Contains bits [31:5] of the lower inclusive limit of the selected MPU memory region. This value is zero extended to provide the base address to be checked against	RW	0x0000000
4:3	SH	Defines the Shareability domain of this region for Normal memory	RW	0x0
2:1	AP	Defines the access permissions for this region	RW	0x0
0	XN	Defines whether code can be executed from this region	RW	0x0

M33: MPU_RLAR_A2 Register

Offset: 0x0edb0

Description

Provides indirect read and write access to the limit address of the currently selected MPU region selected by MPU_RNR[7:2]:(2[1:0]) `FTSSS

Table 234.
MPU_RLAR_A2
Register

Bits	Name	Description	Type	Reset
31:5	LIMIT	Contains bits [31:5] of the upper inclusive limit of the selected MPU memory region. This value is postfixed with 0x1F to provide the limit address to be checked against	RW	0x0000000
4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3:1	ATTRINDX	Associates a set of attributes in the MPU_MAIR0 and MPU_MAIR1 fields	RW	0x0
0	EN	Region enable	RW	0x0

M33: MPU_RBAR_A3 Register

Offset: 0x0edb4

Description

Provides indirect read and write access to the base address of the MPU region selected by MPU_RNR[7:2]:(3[1:0]) `FTSSS

Table 235.
MPU_RBAR_A3
Register

Bits	Name	Description	Type	Reset
31:5	BASE	Contains bits [31:5] of the lower inclusive limit of the selected MPU memory region. This value is zero extended to provide the base address to be checked against	RW	0x0000000
4:3	SH	Defines the Shareability domain of this region for Normal memory	RW	0x0
2:1	AP	Defines the access permissions for this region	RW	0x0
0	XN	Defines whether code can be executed from this region	RW	0x0

M33: MPU_RLAR_A3 Register

Offset: 0x0edb8

Description

Provides indirect read and write access to the limit address of the currently selected MPU region selected by MPU_RNR[7:2]:(3[1:0]) `FTSSS

Table 236.
MPU_RLAR_A3
Register

Bits	Name	Description	Type	Reset
31:5	LIMIT	Contains bits [31:5] of the upper inclusive limit of the selected MPU memory region. This value is postfixed with 0x1F to provide the limit address to be checked against	RW	0x0000000
4	Reserved.	-	-	-
3:1	ATTRINDX	Associates a set of attributes in the MPU_MAIR0 and MPU_MAIR1 fields	RW	0x0
0	EN	Region enable	RW	0x0

M33: MPU_MAIR0 Register

Offset: 0x0edc0

Description

Along with MPU_MAIR1, provides the memory attribute encodings corresponding to the AttrIndex values

Table 237.
MPU_MAIR0 Register

Bits	Name	Description	Type	Reset
31:24	ATTR3	Memory attribute encoding for MPU regions with an AttrIndex of 3	RW	0x00
23:16	ATTR2	Memory attribute encoding for MPU regions with an AttrIndex of 2	RW	0x00

Bits	Name	Description	Type	Reset
15:8	ATTR1	Memory attribute encoding for MPU regions with an AttrIndex of 1	RW	0x00
7:0	ATTR0	Memory attribute encoding for MPU regions with an AttrIndex of 0	RW	0x00

M33: MPU_MAIR1 Register

Offset: 0x0edc4

Description

Along with MPU_MAIR0, provides the memory attribute encodings corresponding to the AttrIndex values

Table 238. MPU_MAIR1 Register

Bits	Name	Description	Type	Reset
31:24	ATTR7	Memory attribute encoding for MPU regions with an AttrIndex of 7	RW	0x00
23:16	ATTR6	Memory attribute encoding for MPU regions with an AttrIndex of 6	RW	0x00
15:8	ATTR5	Memory attribute encoding for MPU regions with an AttrIndex of 5	RW	0x00
7:0	ATTR4	Memory attribute encoding for MPU regions with an AttrIndex of 4	RW	0x00

M33: SAU_CTRL Register

Offset: 0x0eddo

Description

Allows enabling of the Security Attribution Unit

Table 239. SAU_CTRL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	ALLNS	When SAU_CTRL.ENABLE is 0 this bit controls if the memory is marked as Non-secure or Secure	RW	0x0
0	ENABLE	Enables the SAU	RW	0x0

M33: SAU_TYPE Register

Offset: 0x0eddd4

Description

Indicates the number of regions implemented by the Security Attribution Unit

Table 240. SAU_TYPE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	SREGION	The number of implemented SAU regions	RO	0x08

M33: SAU_RNR Register

Offset: 0x0eddd8

Description

Selects the region currently accessed by SAU_RBAR and SAU_RLAR

Table 241. SAU_RNR Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	REGION	Indicates the SAU region accessed by SAU_RBAR and SAU_RLAR	RW	0x00

M33: SAU_RBAR Register

Offset: 0x0eddc

Description

Provides indirect read and write access to the base address of the currently selected SAU region

Table 242. SAU_RBAR Register

Bits	Name	Description	Type	Reset
31:5	BADDR	Holds bits [31:5] of the base address for the selected SAU region	RW	0x0000000
4:0	Reserved.	-	-	-

M33: SAU_RLAR Register

Offset: 0x0ede0

Description

Provides indirect read and write access to the limit address of the currently selected SAU region

Table 243. SAU_RLAR Register

Bits	Name	Description	Type	Reset
31:5	LADDR	Holds bits [31:5] of the limit address for the selected SAU region	RW	0x0000000
4:2	Reserved.	-	-	-
1	NSC	Controls whether Non-secure state is permitted to execute an SG instruction from this region	RW	0x0
0	ENABLE	SAU region enable	RW	0x0

M33: SFSR Register

Offset: 0x0ede4

Description

Provides information about any security related faults

Table 244. SFSR Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	LSERR	Sticky flag indicating that an error occurred during lazy state activation or deactivation	RW	0x0
6	SFARVALID	This bit is set when the SFAR register contains a valid value. As with similar fields, such as BFSR.BFARVALID and MMFSR.MMARVALID, this bit can be cleared by other exceptions, such as BusFault	RW	0x0

Bits	Name	Description	Type	Reset
5	LSPERR	Stick flag indicating that an SAU or IDAU violation occurred during the lazy preservation of floating-point state	RW	0x0
4	INVTRAN	Sticky flag indicating that an exception was raised due to a branch that was not flagged as being domain crossing causing a transition from Secure to Non-secure memory	RW	0x0
3	AUVIOL	Sticky flag indicating that an attempt was made to access parts of the address space that are marked as Secure with NS-Req for the transaction set to Non-secure. This bit is not set if the violation occurred during lazy state preservation. See LSPERR	RW	0x0
2	INVER	This can be caused by EXC_RETURN.DCRS being set to 0 when returning from an exception in the Non-secure state, or by EXC_RETURN.ES being set to 1 when returning from an exception in the Non-secure state	RW	0x0
1	INVIS	This bit is set if the integrity signature in an exception stack frame is found to be invalid during the unstacking operation	RW	0x0
0	INVEP	This bit is set if a function call from the Non-secure state or exception targets a non-SG instruction in the Secure state. This bit is also set if the target address is a SG instruction, but there is no matching SAU/IDAU region with the NSC flag set	RW	0x0

M33: SFAR Register

Offset: 0x0ede8

Description

Shows the address of the memory location that caused a Security violation

Table 245. SFAR Register

Bits	Name	Description	Type	Reset
31:0	ADDRESS	The address of an access that caused a attribution unit violation. This field is only valid when SFSR.SFARVALID is set. This allows the actual flip flops associated with this register to be shared with other fault address registers. If an implementation chooses to share the storage in this way, care must be taken to not leak Secure address information to the Non-secure state. One way of achieving this is to share the SFAR register with the MMFAR_S register, which is not accessible to the Non-secure state	RW	0x00000000

M33: DHCSR Register

Offset: 0x0edf0

Description

Controls halting debug

Table 246. DHCSR Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
26	S_RESTART_ST	Indicates the PE has processed a request to clear DHCSR.C_HALT to 0. That is, either a write to DHCSR that clears DHCSR.C_HALT from 1 to 0, or an External Restart Request	RO	0x0
25	S_RESET_ST	Indicates whether the PE has been reset since the last read of the DHCSR	RO	0x0
24	S_RETIRE_ST	Set to 1 every time the PE retires one of more instructions	RO	0x0
23:21	Reserved.	-	-	-
20	S_SDE	Indicates whether Secure invasive debug is allowed	RO	0x0
19	S_LOCKUP	Indicates whether the PE is in Lockup state	RO	0x0
18	S_SLEEP	Indicates whether the PE is sleeping	RO	0x0
17	S_HALT	Indicates whether the PE is in Debug state	RO	0x0
16	S_REGRDY	Handshake flag to transfers through the DCRDR	RO	0x0
15:6	Reserved.	-	-	-
5	C_SNAPSTALL	Allow imprecise entry to Debug state	RW	0x0
4	Reserved.	-	-	-
3	C_MASKINTS	When debug is enabled, the debugger can write to this bit to mask PendSV, SysTick and external configurable interrupts	RW	0x0
2	C_STEP	Enable single instruction step	RW	0x0
1	C_HALT	PE enter Debug state halt request	RW	0x0
0	C_DEBUGEN	Enable Halting debug	RW	0x0

M33: DCRSR Register

Offset: 0x0edf4

Description

With the DCRDR, provides debug access to the general-purpose registers, special-purpose registers, and the FP extension registers. A write to the DCRSR specifies the register to transfer, whether the transfer is a read or write, and starts the transfer

Table 247. DCRSR Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	REGWR	Specifies the access type for the transfer	RW	0x0
15:7	Reserved.	-	-	-
6:0	REGSEL	Specifies the general-purpose register, special-purpose register, or FP register to transfer	RW	0x00

M33: DCRDR Register

Offset: 0x0edf8

Description

With the DCRSR, provides debug access to the general-purpose registers, special-purpose registers, and the FP

Extension registers. If the Main Extension is implemented, it can also be used for message passing between an external debugger and a debug agent running on the PE

Table 248. DCRDR Register

Bits	Name	Description	Type	Reset
31:0	DBGTMP	Provides debug access for reading and writing the general-purpose registers, special-purpose registers, and Floating-point Extension registers	RW	0x00000000

M33: DEMCR Register

Offset: 0x0edfc

Description

Manages vector catch behavior and DebugMonitor handling when debugging

Table 249. DEMCR Register

Bits	Name	Description	Type	Reset
31:25	Reserved.	-	-	-
24	TRCENA	Global enable for all DWT and ITM features	RW	0x0
23:21	Reserved.	-	-	-
20	SDME	Indicates whether the DebugMonitor targets the Secure or the Non-secure state and whether debug events are allowed in Secure state	RO	0x0
19	MON_REQ	DebugMonitor semaphore bit	RW	0x0
18	MON_STEP	Enable DebugMonitor stepping	RW	0x0
17	MON_PEND	Sets or clears the pending state of the DebugMonitor exception	RW	0x0
16	MON_EN	Enable the DebugMonitor exception	RW	0x0
15:12	Reserved.	-	-	-
11	VC_SFERR	SecureFault exception halting debug vector catch enable	RW	0x0
10	VC_HARDERR	HardFault exception halting debug vector catch enable	RW	0x0
9	VC_INTERR	Enable halting debug vector catch for faults during exception entry and return	RW	0x0
8	VC_BUSERR	BusFault exception halting debug vector catch enable	RW	0x0
7	VC_STATERR	Enable halting debug trap on a UsageFault exception caused by a state information error, for example an Undefined Instruction exception	RW	0x0
6	VC_CHKERR	Enable halting debug trap on a UsageFault exception caused by a checking error, for example an alignment check error	RW	0x0
5	VC_NOCPERR	Enable halting debug trap on a UsageFault caused by an access to a coprocessor	RW	0x0
4	VC_MMERR	Enable halting debug trap on a MemManage exception	RW	0x0
3:1	Reserved.	-	-	-
0	VC_CORERESET	Enable Reset Vector Catch. This causes a warm reset to halt a running system	RW	0x0

M33: DCSR Register

Offset: 0x0ee08

Description

Provides control and status information for Secure debug

Table 250. DCSR Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CDSKEY	Writes to the CDS bit are ignored unless CDSKEY is concurrently written to zero	RW	0x0
16	CDS	This field indicates the current Security state of the processor	RW	0x0
15:2	Reserved.	-	-	-
1	SBRSEL	If SBRSELEN is 1 this bit selects whether the Non-secure or the Secure version of the memory-mapped Banked registers are accessible to the debugger	RW	0x0
0	SBRSELEN	Controls whether the SBRSEL field or the current Security state of the processor selects which version of the memory-mapped Banked registers are accessed to the debugger	RW	0x0

M33: STIR Register

Offset: 0x0ef00

Description

Provides a mechanism for software to generate an interrupt

Table 251. STIR Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8:0	INTID	Indicates the interrupt to be pended. The value written is (ExceptionNumber - 16)	RW	0x000

M33: FPCCR Register

Offset: 0x0ef34

Description

Holds control data for the Floating-point extension

Table 252. FPCCR Register

Bits	Name	Description	Type	Reset
31	ASPEN	When this bit is set to 1, execution of a floating-point instruction sets the CONTROL.FPCA bit to 1	RW	0x0
30	LSPEN	Enables lazy context save of floating-point state	RW	0x0
29	LSPENS	This bit controls whether the LSPEN bit is writeable from the Non-secure state	RW	0x1
28	CLRONRET	Clear floating-point caller saved registers on exception return	RW	0x0
27	CLRONRETS	This bit controls whether the CLRONRET bit is writeable from the Non-secure state	RW	0x0

Bits	Name	Description	Type	Reset
26	TS	Treat floating-point registers as Secure enable	RW	0x0
25:11	Reserved.	-	-	-
10	UFRDY	Indicates whether the software executing when the PE allocated the floating-point stack frame was able to set the UsageFault exception to pending	RW	0x1
9	SPLIMVIOL	This bit is banked between the Security states and indicates whether the floating-point context violates the stack pointer limit that was active when lazy state preservation was activated. SPLIMVIOL modifies the lazy floating-point state preservation behavior	RW	0x0
8	MONRDY	Indicates whether the software executing when the PE allocated the floating-point stack frame was able to set the DebugMonitor exception to pending	RW	0x0
7	SFRDY	Indicates whether the software executing when the PE allocated the floating-point stack frame was able to set the SecureFault exception to pending. This bit is only present in the Secure version of the register, and behaves as RAZ/WI when accessed from the Non-secure state	RW	0x0
6	BFRDY	Indicates whether the software executing when the PE allocated the floating-point stack frame was able to set the BusFault exception to pending	RW	0x1
5	MMRDY	Indicates whether the software executing when the PE allocated the floating-point stack frame was able to set the MemManage exception to pending	RW	0x1
4	HFRDY	Indicates whether the software executing when the PE allocated the floating-point stack frame was able to set the HardFault exception to pending	RW	0x1
3	THREAD	Indicates the PE mode when it allocated the floating-point stack frame	RW	0x0
2	S	Security status of the floating-point context. This bit is only present in the Secure version of the register, and behaves as RAZ/WI when accessed from the Non-secure state. This bit is updated whenever lazy state preservation is activated, or when a floating-point instruction is executed	RW	0x0
1	USER	Indicates the privilege level of the software executing when the PE allocated the floating-point stack frame	RW	0x1
0	LSPACT	Indicates whether lazy preservation of the floating-point state is active	RW	0x0

M33: FPCAR Register

Offset: 0x0ef38

Description

Holds the location of the unpopulated floating-point register space allocated on an exception stack frame

Table 253. FPCAR Register

Bits	Name	Description	Type	Reset
31:3	ADDRESS	The location of the unpopulated floating-point register space allocated on an exception stack frame	RW	0x00000000
2:0	Reserved.	-	-	-

M33: FPDSCR Register

Offset: 0x0ef3c

Description

Holds the default values for the floating-point status control data that the PE assigns to the FPSCR when it creates a new floating-point context

Table 254. FPDSCR Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	AHP	Default value for FPSCR.AHP	RW	0x0
25	DN	Default value for FPSCR.DN	RW	0x0
24	FZ	Default value for FPSCR.FZ	RW	0x0
23:22	RMODE	Default value for FPSCR.RMode	RW	0x0
21:0	Reserved.	-	-	-

M33: MVFR0 Register

Offset: 0x0ef40

Description

Describes the features provided by the Floating-point Extension

Table 255. MVFR0 Register

Bits	Name	Description	Type	Reset
31:28	FPROUND	Indicates the rounding modes supported by the FP Extension	RO	0x6
27:24	Reserved.	-	-	-
23:20	FPSQRT	Indicates the support for FP square root operations	RO	0x5
19:16	FPDIVIDE	Indicates the support for FP divide operations	RO	0x4
15:12	Reserved.	-	-	-
11:8	FPDP	Indicates support for FP double-precision operations	RO	0x6
7:4	FPSP	Indicates support for FP single-precision operations	RO	0x0
3:0	SIMDREG	Indicates size of FP register file	RO	0x1

M33: MVFR1 Register

Offset: 0x0ef44

Description

Describes the features provided by the Floating-point Extension

Table 256. MVFR1 Register

Bits	Name	Description	Type	Reset
31:28	FMAC	Indicates whether the FP Extension implements the fused multiply accumulate instructions	RO	0x8
27:24	FPHP	Indicates whether the FP Extension implements half-precision FP conversion instructions	RO	0x5
23:8	Reserved.	-	-	-
7:4	FPDNAN	Indicates whether the FP hardware implementation supports NaN propagation	RO	0x8
3:0	FPFTZ	Indicates whether subnormals are always flushed-to-zero	RO	0x9

M33: MVFR2 Register

Offset: 0x0ef48

Description

Describes the features provided by the Floating-point Extension

Table 257. MVFR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	FPMISC	Indicates support for miscellaneous FP features	RO	0x6
3:0	Reserved.	-	-	-

M33: DDEVARC Register

Offset: 0x0efbc

Description

Provides CoreSight discovery information for the SCS

Table 258. DDEVARC Register

Bits	Name	Description	Type	Reset
31:21	ARCHITECT	Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.	RO	0x23b
20	PRESENT	Defines that the DEVARC register is present	RO	0x1
19:16	REVISION	Defines the architecture revision of the component	RO	0x0
15:12	ARCHVER	Defines the architecture version of the component	RO	0x2
11:0	ARCHPART	Defines the architecture of the component	RO	0xa04

M33: DDEVTYPE Register

Offset: 0x0efcc

Description

Provides CoreSight discovery information for the SCS

Table 259. DDEVTYPE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SUB	Component sub-type	RO	0x0

Bits	Name	Description	Type	Reset
3:0	MAJOR	CoreSight major type	RO	0x0

M33: DPIDR4 Register

Offset: 0x0efd0

Description

Provides CoreSight discovery information for the SCS

Table 260. DPIDR4 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SIZE	See CoreSight Architecture Specification	RO	0x0
3:0	DES_2	See CoreSight Architecture Specification	RO	0x4

M33: DPIDR5 Register

Offset: 0x0efd4

Description

Provides CoreSight discovery information for the SCS

Table 261. DPIDR5 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: DPIDR6 Register

Offset: 0x0efd8

Description

Provides CoreSight discovery information for the SCS

Table 262. DPIDR6 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: DPIDR7 Register

Offset: 0x0efdc

Description

Provides CoreSight discovery information for the SCS

Table 263. DPIDR7 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: DPIDR0 Register

Offset: 0x0efe0

Description

Provides CoreSight discovery information for the SCS

Table 264. DPIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PART_0	See CoreSight Architecture Specification	RO	0x21

M33: DPIDR1 Register

Offset: 0x0efe4

Description

Provides CoreSight discovery information for the SCS

Table 265. DPIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DES_0	See CoreSight Architecture Specification	RO	0xb
3:0	PART_1	See CoreSight Architecture Specification	RO	0xd

M33: DPIDR2 Register

Offset: 0x0efe8

Description

Provides CoreSight discovery information for the SCS

Table 266. DPIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	See CoreSight Architecture Specification	RO	0x0
3	JEDEC	See CoreSight Architecture Specification	RO	0x1
2:0	DES_1	See CoreSight Architecture Specification	RO	0x3

M33: DPIDR3 Register

Offset: 0x0efec

Description

Provides CoreSight discovery information for the SCS

Table 267. DPIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVAND	See CoreSight Architecture Specification	RO	0x0
3:0	CMOD	See CoreSight Architecture Specification	RO	0x0

M33: DCIDR0 Register

Offset: 0x0eff0

Description

Provides CoreSight discovery information for the SCS

Table 268. DCIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_0	See CoreSight Architecture Specification	RO	0x0d

M33: DCIDR1 Register

Offset: 0x0eff4

Description

Provides CoreSight discovery information for the SCS

Table 269. DCIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	CLASS	See CoreSight Architecture Specification	RO	0x9
3:0	PRMBL_1	See CoreSight Architecture Specification	RO	0x0

M33: DCIDR2 Register

Offset: 0x0eff8

Description

Provides CoreSight discovery information for the SCS

Table 270. DCIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_2	See CoreSight Architecture Specification	RO	0x05

M33: DCIDR3 Register

Offset: 0x0effc

Description

Provides CoreSight discovery information for the SCS

Table 271. DCIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_3	See CoreSight Architecture Specification	RO	0xb1

M33: TRCPRGCTRL Register

Offset: 0x41004

Description

Programming Control Register

Table 272. TRCPRGCTRL Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	EN	Trace Unit Enable	RW	0x0

M33: TRCSTATR Register

Offset: 0x4100c

Description

The TRCSTATR indicates the ETM-Teal status

Table 273. TRCSTATR Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	PMSTABLE	Indicates whether the ETM-Teal registers are stable and can be read	RO	0x0
0	IDLE	Indicates that the trace unit is inactive	RO	0x0

M33: TRCCONFIGR Register

Offset: 0x41010

Description

The TRCCONFIGR sets the basic tracing options for the trace unit

Table 274. TRCCONFIGR Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	RS	Resturn stack enable	RW	0x0
11	TS	Global timestamp tracing	RW	0x0
10:5	COND	Conditional instruction tracing	RW	0x00
4	CCI	Cycle counting in instruction trace	RW	0x0
3	BB	Branch broadcast mode	RW	0x0
2:0	Reserved.	-	-	-

M33: TRCEVENTCTL0R Register

Offset: 0x41020

Description

The TRCEVENTCTL0R controls the tracing of events in the trace stream. The events also drive the ETM-Teal external outputs.

Table 275. TRCEVENTCTL0R Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	TYPE1	Selects the resource type for event 1	RW	0x0
14:11	Reserved.	-	-	-
10:8	SEL1	Selects the resource number, based on the value of TYPE1: When TYPE1 is 0, selects a single selected resource from 0-15 defined by SEL1[2:0]. When TYPE1 is 1, selects a Boolean combined resource pair from 0-7 defined by SEL1[2:0]	RW	0x0
7	TYPE0	Selects the resource type for event 0	RW	0x0
6:3	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
2:0	SEL0	Selects the resource number, based on the value of TYPE0: When TYPE1 is 0, selects a single selected resource from 0-15 defined by SEL0[2:0]. When TYPE1 is 1, selects a Boolean combined resource pair from 0-7 defined by SEL0[2:0]	RW	0x0

M33: TRCEVENTCTL1R Register

Offset: 0x41024

Description

The TRCEVENTCTL1R controls how the events selected by TRCEVENTCTL0R behave

Table 276.
TRCEVENTCTL1R
Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	LPOVERRIDE	Low power state behavior override	RW	0x0
11	ATB	ATB enabled	RW	0x0
10:2	Reserved.	-	-	-
1	INSTEN1	One bit per event, to enable generation of an event element in the instruction trace stream when the selected event occurs	RW	0x0
0	INSTEN0	One bit per event, to enable generation of an event element in the instruction trace stream when the selected event occurs	RW	0x0

M33: TRCSTALLCTLR Register

Offset: 0x4102c

Description

The TRCSTALLCTLR enables ETM-Teal to stall the processor if the ETM-Teal FIFO goes over the programmed level to minimize risk of overflow

Table 277.
TRCSTALLCTLR
Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	INSTPRIORITY	Reserved, RES0	RO	0x0
9	Reserved.	-	-	-
8	ISTALL	Stall processor based on instruction trace buffer space	RW	0x0
7:4	Reserved.	-	-	-
3:2	LEVEL	Threshold at which stalling becomes active. This provides four levels. This level can be varied to optimize the level of invasion caused by stalling, balanced against the risk of a FIFO overflow	RW	0x0
1:0	Reserved.	-	-	-

M33: TRCTSCTLR Register

Offset: 0x41030

Description

The TRCTSCTRLR controls the insertion of global timestamps into the trace stream. A timestamp is always inserted into the instruction trace stream

Table 278.
TRCTSCTRLR Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	TYPE0	Selects the resource type for event 0	RW	0x0
6:2	Reserved.	-	-	-
1:0	SEL0	Selects the resource number, based on the value of TYPE0: When TYPE1 is 0, selects a single selected resource from 0-15 defined by SEL0[2:0]. When TYPE1 is 1, selects a Boolean combined resource pair from 0-7 defined by SEL0[2:0]	RW	0x0

M33: TRCSYNCPR Register

Offset: 0x41034

Description

The TRCSYNCPR specifies the period of trace synchronization of the trace streams. TRCSYNCPR defines a number of bytes of trace between requests for trace synchronization. This value is always a power of two

Table 279.
TRCSYNCPR Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4:0	PERIOD	Defines the number of bytes of trace between trace synchronization requests as a total of the number of bytes generated by the instruction stream. The number of bytes is $2N$ where N is the value of this field: - A value of zero disables these periodic trace synchronization requests, but does not disable other trace synchronization requests. - The minimum value that can be programmed, other than zero, is 8, providing a minimum trace synchronization period of 256 bytes. - The maximum value is 20, providing a maximum trace synchronization period of 2^{20} bytes	RO	0x0a

M33: TRCCCCTRLR Register

Offset: 0x41038

Description

The TRCCCCTRLR sets the threshold value for instruction trace cycle counting. The threshold represents the minimum interval between cycle count trace packets

Table 280.
TRCCCCTRLR Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:0	THRESHOLD	Instruction trace cycle count threshold	RW	0x000

M33: TRCVICTLR Register

Offset: 0x41080

Description

The TRCVICTLR controls instruction trace filtering

Table 281. TRCVICTLR Register

Bits	Name	Description	Type	Reset
31:20	Reserved.	-	-	-
19	EXLEVEL_S3	In Secure state, each bit controls whether instruction tracing is enabled for the corresponding exception level	RW	0x0
18:17	Reserved.	-	-	-
16	EXLEVEL_S0	In Secure state, each bit controls whether instruction tracing is enabled for the corresponding exception level	RW	0x0
15:12	Reserved.	-	-	-
11	TRCERR	Selects whether a system error exception must always be traced	RW	0x0
10	TRCRESET	Selects whether a reset exception must always be traced	RW	0x0
9	SSSTATUS	Indicates the current status of the start/stop logic	RW	0x0
8	Reserved.	-	-	-
7	TYPE0	Selects the resource type for event 0	RW	0x0
6:2	Reserved.	-	-	-
1:0	SEL0	Selects the resource number, based on the value of TYPE0: When TYPE1 is 0, selects a single selected resource from 0-15 defined by SEL0[2:0]. When TYPE1 is 1, selects a Boolean combined resource pair from 0-7 defined by SEL0[2:0]	RW	0x0

M33: TRCCNTRLDVR0 Register

Offset: 0x41140

Description

The TRCCNTRLDVR defines the reload value for the reduced function counter

Table 282. TRCCNTRLDVR0 Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:0	VALUE	Defines the reload value for the counter. This value is loaded into the counter each time the reload event occurs	RW	0x0000

M33: TRC IDR8 Register

Offset: 0x41180

Description

TRC IDR8

Table 283. TRC IDR8 Register

Bits	Name	Description	Type	Reset
31:0	MAXSPEC	reads as `ImpDef	RO	0x00000000

M33: TRC IDR9 Register

Offset: 0x41184

Description

TRC IDR9

Table 284. TRC IDR9 Register

Bits	Name	Description	Type	Reset
31:0	NUMP0KEY	reads as `ImpDef	RO	0x00000000

M33: TRC IDR10 Register

Offset: 0x41188

Description

TRC IDR10

Table 285. TRC IDR10 Register

Bits	Name	Description	Type	Reset
31:0	NUMP1KEY	reads as `ImpDef	RO	0x00000000

M33: TRC IDR11 Register

Offset: 0x4118c

Description

TRC IDR11

Table 286. TRC IDR11 Register

Bits	Name	Description	Type	Reset
31:0	NUMP1SPC	reads as `ImpDef	RO	0x00000000

M33: TRC IDR12 Register

Offset: 0x41190

Description

TRC IDR12

Table 287. TRC IDR12 Register

Bits	Name	Description	Type	Reset
31:0	NUMCONDKEY	reads as `ImpDef	RO	0x00000001

M33: TRC IDR13 Register

Offset: 0x41194

Description

TRC IDR13

Table 288. TRC IDR13 Register

Bits	Name	Description	Type	Reset
31:0	NUMCONDSPC	reads as `ImpDef	RO	0x00000000

M33: TRCIMSPEC Register

Offset: 0x411c0

Description

The TRCIMSPEC shows the presence of any IMPLEMENTATION SPECIFIC features, and enables any features that are provided

Table 289.
TRCIMSPEC Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	SUPPORT	Reserved, RES0	RO	0x0

M33: TRCIDR0 Register

Offset: 0x411e0

Description

TRCIDR0

Table 290. TRCIDR0 Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29	COMMOP	reads as `ImpDef	RO	0x1
28:24	TSSIZE	reads as `ImpDef	RO	0x08
23:18	Reserved.	-	-	-
17	TRCEXDATA	reads as `ImpDef	RO	0x0
16:15	QSUPP	reads as `ImpDef	RO	0x0
14	QFILT	reads as `ImpDef	RO	0x0
13:12	CONDTYPE	reads as `ImpDef	RO	0x0
11:10	NUMEVENT	reads as `ImpDef	RO	0x1
9	RETSTACK	reads as `ImpDef	RO	0x1
8	Reserved.	-	-	-
7	TRCCCI	reads as `ImpDef	RO	0x1
6	TRCCOND	reads as `ImpDef	RO	0x1
5	TRCBB	reads as `ImpDef	RO	0x1
4:3	TRCDATA	reads as `ImpDef	RO	0x0
2:1	INSTPO	reads as `ImpDef	RO	0x0
0	RES1	Reserved, RES1	RO	0x1

M33: TRCIDR1 Register

Offset: 0x411e4

Description

TRCIDR1

Table 291. TRCIDR1 Register

Bits	Name	Description	Type	Reset
31:24	DESIGNER	reads as `ImpDef	RO	0x41
23:16	Reserved.	-	-	-
15:12	RES1	Reserved, RES1	RO	0xf
11:8	TRCARCHMAJ	reads as 0b0100	RO	0x4
7:4	TRCARCHMIN	reads as 0b0000	RO	0x2
3:0	REVISION	reads as `ImpDef	RO	0x1

M33: TRC IDR2 Register

Offset: 0x411e8

Description

TRC IDR2

Table 292. TRC IDR2 Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28:25	CCSIZE	reads as `ImpDef	RO	0x0
24:20	DVSIZE	reads as `ImpDef	RO	0x00
19:15	DASIZE	reads as `ImpDef	RO	0x00
14:10	VMIDSIZE	reads as `ImpDef	RO	0x00
9:5	CIDSIZE	reads as `ImpDef	RO	0x00
4:0	IASIZE	reads as `ImpDef	RO	0x04

M33: TRC IDR3 Register

Offset: 0x411ec

Description

TRC IDR3

Table 293. TRC IDR3 Register

Bits	Name	Description	Type	Reset
31	NOOVERFLOW	reads as `ImpDef	RO	0x0
30:28	NUMPROC	reads as `ImpDef	RO	0x0
27	SYSSTALL	reads as `ImpDef	RO	0x1
26	STALLCTL	reads as `ImpDef	RO	0x1
25	SYNCPR	reads as `ImpDef	RO	0x1
24	TRCERR	reads as `ImpDef	RO	0x1
23:20	EXLEVEL_NS	reads as `ImpDef	RO	0x0
19:16	EXLEVEL_S	reads as `ImpDef	RO	0x9
15:12	Reserved.	-	-	-
11:0	CCITMIN	reads as `ImpDef	RO	0x004

M33: TRC IDR4 Register

Offset: 0x411f0

Description

TRC IDR4

Table 294. TRC IDR4 Register

Bits	Name	Description	Type	Reset
31:28	NUMVMIDC	reads as `ImpDef	RO	0x0
27:24	NUMCIDC	reads as `ImpDef	RO	0x0
23:20	NUMSSCC	reads as `ImpDef	RO	0x1
19:16	NUMRSPAIR	reads as `ImpDef	RO	0x1

Bits	Name	Description	Type	Reset
15:12	NUMPC	reads as `ImpDef	RO	0x4
11:9	Reserved.	-	-	-
8	SUPPDAC	reads as `ImpDef	RO	0x0
7:4	NUMDVC	reads as `ImpDef	RO	0x0
3:0	NUMACPAIRS	reads as `ImpDef	RO	0x0

M33: TRC IDR5 Register

Offset: 0x411f4

Description

TRC IDR5

Table 295. TRC IDR5 Register

Bits	Name	Description	Type	Reset
31	REDFUNCNTR	reads as `ImpDef	RO	0x1
30:28	NUMCNTR	reads as `ImpDef	RO	0x1
27:25	NUMSEQSTATE	reads as `ImpDef	RO	0x0
24	Reserved.	-	-	-
23	LPOVERRIDE	reads as `ImpDef	RO	0x1
22	ATBTRIG	reads as `ImpDef	RO	0x1
21:16	TRACEIDSIZE	reads as 0x07	RO	0x07
15:12	Reserved.	-	-	-
11:9	NUMEXTINSEL	reads as `ImpDef	RO	0x0
8:0	NUMEXTIN	reads as `ImpDef	RO	0x004

M33: TRC IDR6 Register

Offset: 0x411f8

Description

TRC IDR6

Table 296. TRC IDR6 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: TRC IDR7 Register

Offset: 0x411fc

Description

TRC IDR7

Table 297. TRCIDR7 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: TRCRSCTRLR2 Register

Offset: 0x41208

Description

The TRCRSCTRLR controls the trace resources

Table 298. TRCRSCTRLR2 Register

Bits	Name	Description	Type	Reset
31:22	Reserved.	-	-	-
21	PAIRINV	Inverts the result of a combined pair of resources. This bit is only implemented on the lower register for a pair of resource selectors	RW	0x0
20	INV	Inverts the selected resources	RW	0x0
19	Reserved.	-	-	-
18:16	GROUP	Selects a group of resource	RW	0x0
15:8	Reserved.	-	-	-
7:0	SELECT	Selects one or more resources from the wanted group. One bit is provided per resource from the group	RW	0x00

M33: TRCRSCTRLR3 Register

Offset: 0x4120c

Description

The TRCRSCTRLR controls the trace resources

Table 299. TRCRSCTRLR3 Register

Bits	Name	Description	Type	Reset
31:22	Reserved.	-	-	-
21	PAIRINV	Inverts the result of a combined pair of resources. This bit is only implemented on the lower register for a pair of resource selectors	RW	0x0
20	INV	Inverts the selected resources	RW	0x0
19	Reserved.	-	-	-
18:16	GROUP	Selects a group of resource	RW	0x0
15:8	Reserved.	-	-	-
7:0	SELECT	Selects one or more resources from the wanted group. One bit is provided per resource from the group	RW	0x00

M33: TRCSSCSR Register

Offset: 0x412a0

Description

Controls the corresponding single-shot comparator resource

Table 300. TRCSSCSR Register

Bits	Name	Description	Type	Reset
31	STATUS	Single-shot status bit. Indicates if any of the comparators, that TRCSSCCRn.SAC or TRCSSCCRn.ARC selects, have matched	RW	0x0
30:4	Reserved.	-	-	-
3	PC	Reserved, RES1	RO	0x0
2	DV	Reserved, RES0	RO	0x0
1	DA	Reserved, RES0	RO	0x0
0	INST	Reserved, RES0	RO	0x0

M33: TRCSSPCICR Register

Offset: 0x412c0

Description

Selects the PE comparator inputs for Single-shot control

Table 301. TRCSSPCICR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	PC	Selects one or more PE comparator inputs for Single-shot control. TRCIDR4.NUMPC defines the size of the PC field. 1 bit is provided for each implemented PE comparator input. For example, if bit[1] == 1 this selects PE comparator input 1 for Single-shot control	RW	0x0

M33: TRCPDCR Register

Offset: 0x41310

Description

Requests the system to provide power to the trace unit

Table 302. TRCPDCR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	PU	Powerup request bit:	RW	0x0
2:0	Reserved.	-	-	-

M33: TRCPDSR Register

Offset: 0x41314

Description

Returns the following information about the trace unit: - OS Lock status. - Core power domain status. - Power interruption status

Table 303. TRCPDSR Register

Bits	Name	Description	Type	Reset
31:6	Reserved.	-	-	-
5	OSLK	OS Lock status bit:	RO	0x0
4:2	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
1	STICKYPD	Sticky powerdown status bit. Indicates whether the trace register state is valid:	RO	0x1
0	POWER	Power status bit:	RO	0x1

M33: TRCITATBIDR Register

Offset: 0x41ee4

Description

Trace Intergration ATB Identification Register

Table 304.
TRCITATBIDR Register

Bits	Name	Description	Type	Reset
31:7	Reserved.	-	-	-
6:0	ID	Trace ID	RW	0x00

M33: TRCITIATBINR Register

Offset: 0x41ef4

Description

Trace Integration Instruction ATB In Register

Table 305.
TRCITIATBINR Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	AFVALIDM	Integration Mode instruction AFVALIDM in	RW	0x0
0	ATREADYM	Integration Mode instruction ATREADYM in	RW	0x0

M33: TRCITIATBOUTR Register

Offset: 0x41efc

Description

Trace Integration Instruction ATB Out Register

Table 306.
TRCITIATBOUTR Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	AFREADY	Integration Mode instruction AFREADY out	RW	0x0
0	ATVALID	Integration Mode instruction ATVALID out	RW	0x0

M33: TRCCLAIMSET Register

Offset: 0x41fa0

Description

Claim Tag Set Register

Table 307.
TRCCLAIMSET Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	SET3	When a write to one of these bits occurs, with the value:	RW	0x1
2	SET2	When a write to one of these bits occurs, with the value:	RW	0x1

Bits	Name	Description	Type	Reset
1	SET1	When a write to one of these bits occurs, with the value:	RW	0x1
0	SET0	When a write to one of these bits occurs, with the value:	RW	0x1

M33: TRCCLAIMCLR Register

Offset: 0x41fa4

Description

Claim Tag Clear Register

Table 308.
TRCCLAIMCLR
Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	CLR3	When a write to one of these bits occurs, with the value:	RW	0x0
2	CLR2	When a write to one of these bits occurs, with the value:	RW	0x0
1	CLR1	When a write to one of these bits occurs, with the value:	RW	0x0
0	CLR0	When a write to one of these bits occurs, with the value:	RW	0x0

M33: TRCAUTHSTATUS Register

Offset: 0x41fb8

Description

Returns the level of tracing that the trace unit can support

Table 309.
TRCAUTHSTATUS
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:6	SNID	Indicates whether the system enables the trace unit to support Secure non-invasive debug:	RO	0x0
5:4	SID	Indicates whether the trace unit supports Secure invasive debug:	RO	0x0
3:2	NSNID	Indicates whether the system enables the trace unit to support Non-secure non-invasive debug:	RO	0x0
1:0	NSID	Indicates whether the trace unit supports Non-secure invasive debug:	RO	0x0

M33: TRCDEVARCH Register

Offset: 0x41fbc

Description

TRCDEVARCH

Table 310.
TRCDEVARCH Register

Bits	Name	Description	Type	Reset
31:21	ARCHITECT	reads as 0b01000111011	RO	0x23b
20	PRESENT	reads as 0b1	RO	0x1
19:16	REVISION	reads as 0b0000	RO	0x2
15:0	ARCHID	reads as 0b0100101000010011	RO	0x4a13

M33: TRCDEVID Register

Offset: 0x41fc8

Description

TRCDEVID

Table 311. TRCDEVID Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: TRCDEVTYPE Register

Offset: 0x41fcc

Description

TRCDEVTYPE

Table 312. TRCDEVTYPE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SUB	reads as 0b0001	RO	0x1
3:0	MAJOR	reads as 0b0011	RO	0x3

M33: TRCPIDR4 Register

Offset: 0x41fd0

Description

TRCPIDR4

Table 313. TRCPIDR4 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SIZE	reads as `ImpDef	RO	0x0
3:0	DES_2	reads as `ImpDef	RO	0x4

M33: TRCPIDR5 Register

Offset: 0x41fd4

Description

TRCPIDR5

Table 314. TRCPIDR5 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: TRCPIDR6 Register

Offset: 0x41fd8

Description

TRCPIDR6

Table 315. TRCPIDR6 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: TRCPIDR7 Register

Offset: 0x41fdc

Description

TRCPIDR7

Table 316. TRCPIDR7 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: TRCPIDR0 Register

Offset: 0x41fe0

Description

TRCPIDR0

Table 317. TRCPIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PART_0	reads as `ImpDef	RO	0x21

M33: TRCPIDR1 Register

Offset: 0x41fe4

Description

TRCPIDR1

Table 318. TRCPIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DES_0	reads as `ImpDef	RO	0xb
3:0	PART_0	reads as `ImpDef	RO	0xd

M33: TRCPIDR2 Register

Offset: 0x41fe8

Description

TRCPIDR2

Table 319. TRCPIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	reads as `ImpDef	RO	0x2
3	JEDEC	reads as 0b1	RO	0x1
2:0	DES_0	reads as `ImpDef	RO	0x3

M33: TRCPIDR3 Register

Offset: 0x41fec**Description**

TRCPIDR3

Table 320. TRCPIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVAND	reads as `ImpDef	RO	0x0
3:0	CMOD	reads as `ImpDef	RO	0x0

M33: TRCCIDR0 Register**Offset:** 0x41ff0**Description**

TRCCIDR0

Table 321. TRCCIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_0	reads as 0b00001101	RO	0x0d

M33: TRCCIDR1 Register**Offset:** 0x41ff4**Description**

TRCCIDR1

Table 322. TRCCIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	CLASS	reads as 0b1001	RO	0x9
3:0	PRMBL_1	reads as 0b0000	RO	0x0

M33: TRCCIDR2 Register**Offset:** 0x41ff8**Description**

TRCCIDR2

Table 323. TRCCIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_2	reads as 0b00000101	RO	0x05

M33: TRCCIDR3 Register**Offset:** 0x41ffc**Description**

TRCCIDR3

Table 324. TRCCIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7:0	PRMBL_3	reads as 0b10110001	RO	0xb1

M33: CTICONTROL Register

Offset: 0x42000

Description

CTI Control Register

Table 325.
CTICONTROL Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	GLBEN	Enables or disables the CTI	RW	0x0

M33: CTIINTACK Register

Offset: 0x42010

Description

CTI Interrupt Acknowledge Register

Table 326. CTIINTACK Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	INTACK	Acknowledges the corresponding ctirigout output. There is one bit of the register for each ctirigout output. When a 1 is written to a bit in this register, the corresponding ctirigout is acknowledged, causing it to be cleared.	RW	0x00

M33: CTIAPPSET Register

Offset: 0x42014

Description

CTI Application Trigger Set Register

Table 327. CTIAPPSET Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	APPSET	Setting a bit HIGH generates a channel event for the selected channel. There is one bit of the register for each channel	RW	0x0

M33: CTIAPPCLEAR Register

Offset: 0x42018

Description

CTI Application Trigger Clear Register

Table 328.
CTIAPPCLEAR
Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	APPCLEAR	Sets the corresponding bits in the CTIAPPSET to 0. There is one bit of the register for each channel.	RW	0x0

M33: CTIAPPULSE Register

Offset: 0x4201c

Description

CTI Application Pulse Register

Table 329.
CTIAPPULSE
Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	APPULSE	Setting a bit HIGH generates a channel event pulse for the selected channel. There is one bit of the register for each channel.	RW	0x0

M33: CTIINENO, CTIINEN1, ..., CTIINEN6, CTIINEN7 Registers

Offsets: 0x42020, 0x42024, ..., 0x42038, 0x4203c

Description

CTI Trigger to Channel Enable Registers

Table 330. CTIINENO,
CTIINEN1, ...,
CTIINEN6, CTIINEN7
Registers

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	TRIGINEN	Enables a cross trigger event to the corresponding channel when a ctirigin input is activated. There is one bit of the field for each of the four channels	RW	0x0

M33: CTIOUTENO, CTIOUTEN1, ..., CTIOUTEN6, CTIOUTEN7 Registers

Offsets: 0x420a0, 0x420a4, ..., 0x420b8, 0x420bc

Description

CTI Trigger to Channel Enable Registers

Table 331.
CTIOUTENO,
CTIOUTEN1, ...,
CTIOUTEN6,
CTIOUTEN7 Registers

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	TRIGOUTEN	Enables a cross trigger event to ctirigout when the corresponding channel is activated. There is one bit of the field for each of the four channels.	RW	0x0

M33: CTITRIGINSTATUS Register

Offset: 0x42130

Description

CTI Trigger to Channel Enable Registers

Table 332.
CTITRIGINSTATUS
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7:0	TRIGINSTATUS	Shows the status of the ctirigin inputs. There is one bit of the field for each trigger input. Because the register provides a view of the raw ctirigin inputs, the reset value is UNKNOWN.	RO	0x00

M33: CTITRIGOUTSTATUS Register

Offset: 0x42134

Description

CTI Trigger In Status Register

Table 333.
CTITRIGOUTSTATUS
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	TRIGOUTSTATUS	Shows the status of the ctirigout outputs. There is one bit of the field for each trigger output.	RO	0x00

M33: CTICHINSTATUS Register

Offset: 0x42138

Description

CTI Channel In Status Register

Table 334.
CTICHINSTATUS
Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	CTICHOUTSTATUS	Shows the status of the ctichout outputs. There is one bit of the field for each channel output	RO	0x0

M33: CTIGATE Register

Offset: 0x42140

Description

Enable CTI Channel Gate register

Table 335. CTIGATE
Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	CTIGATEEN3	Enable ctichout3. Set to 0 to disable channel propagation.	RW	0x1
2	CTIGATEEN2	Enable ctichout2. Set to 0 to disable channel propagation.	RW	0x1
1	CTIGATEEN1	Enable ctichout1. Set to 0 to disable channel propagation.	RW	0x1
0	CTIGATEEN0	Enable ctichout0. Set to 0 to disable channel propagation.	RW	0x1

M33: ASICCTL Register

Offset: 0x42144

Description

External Multiplexer Control register

Table 336. ASICCTL Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: ITCHOUT Register

Offset: 0x42ee4

Description

Integration Test Channel Output register

Table 337. ITCHOUT Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	CTCHOUT	Sets the value of the ctichout outputs	RW	0x0

M33: ITTRIGOUT Register

Offset: 0x42ee8

Description

Integration Test Trigger Output register

Table 338. ITTRIGOUT Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	CTTRIGOUT	Sets the value of the cttrigout outputs	RW	0x00

M33: ITCHIN Register

Offset: 0x42ef4

Description

Integration Test Channel Input register

Table 339. ITCHIN Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	CTCHIN	Reads the value of the ctichin inputs.	RO	0x0

M33: ITCTRL Register

Offset: 0x42f00

Description

Integration Mode Control register

Table 340. ITCTRL Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	IME	Integration Mode Enable	RW	0x0

M33: DEVARC Register

Offset: 0x42fbc

Description

Device Architecture register

Table 341. DEVARC
Register

Bits	Name	Description	Type	Reset
31:21	ARCHITECT	Indicates the component architect	RO	0x23b
20	PRESENT	Indicates whether the DEVARC register is present	RO	0x1
19:16	REVISION	Indicates the architecture revision	RO	0x0
15:0	ARCHID	Indicates the component	RO	0x1a14

M33: DEVID Register

Offset: 0x42fc8

Description

Device Configuration register

Table 342. DEVID
Register

Bits	Name	Description	Type	Reset
31:20	Reserved.	-	-	-
19:16	NUMCH	Number of ECT channels available	RO	0x4
15:8	NUMTRIG	Number of ECT triggers available.	RO	0x08
7:5	Reserved.	-	-	-
4:0	EXTMUXNUM	Indicates the number of multiplexers available on Trigger Inputs and Trigger Outputs that are using asicctl. The default value of 0b00000 indicates that no multiplexing is present. This value of this bit depends on the Verilog define EXTMUXNUM that you must change accordingly.	RO	0x00

M33: DEVTYPE Register

Offset: 0x42fcc

Description

Device Type Identifier register

Table 343. DEVTYPE
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	SUB	Sub-classification of the type of the debug component as specified in the ARM Architecture Specification within the major classification as specified in the MAJOR field.	RO	0x1
3:0	MAJOR	Major classification of the type of the debug component as specified in the ARM Architecture Specification for this debug and trace component.	RO	0x4

M33: PIDR4 Register

Offset: 0x42fd0

Description

CoreSight Periperal ID4

Table 344. PIDR4
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7:4	SIZE	Always 0b0000. Indicates that the device only occupies 4KB of memory	RO	0x0
3:0	DES_2	Together, PIDR1.DES_0, PIDR2.DES_1, and PIDR4.DES_2 identify the designer of the component.	RO	0x4

M33: PIDR5 Register

Offset: 0x42fd4

Description

CoreSight Periperal ID5

Table 345. PIDR5 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: PIDR6 Register

Offset: 0x42fd8

Description

CoreSight Periperal ID6

Table 346. PIDR6 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: PIDR7 Register

Offset: 0x42fdc

Description

CoreSight Periperal ID7

Table 347. PIDR7 Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

M33: PIDR0 Register

Offset: 0x42fe0

Description

CoreSight Periperal ID0

Table 348. PIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PART_0	Bits[7:0] of the 12-bit part number of the component. The designer of the component assigns this part number.	RO	0x21

M33: PIDR1 Register

Offset: 0x42fe4

Description

CoreSight Periperal ID1

Table 349. PIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DES_0	Together, PIDR1.DES_0, PIDR2.DES_1, and PIDR4.DES_2 identify the designer of the component.	RO	0xb
3:0	PART_1	Bits[11:8] of the 12-bit part number of the component. The designer of the component assigns this part number.	RO	0xd

M33: PIDR2 Register

Offset: 0x42fe8

Description

CoreSight Periperal ID2

Table 350. PIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	This device is at r1p0	RO	0x0
3	JEDEC	Always 1. Indicates that the JEDEC-assigned designer ID is used.	RO	0x1
2:0	DES_1	Together, PIDR1.DES_0, PIDR2.DES_1, and PIDR4.DES_2 identify the designer of the component.	RO	0x3

M33: PIDR3 Register

Offset: 0x42fec

Description

CoreSight Periperal ID3

Table 351. PIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVAND	Indicates minor errata fixes specific to the revision of the component being used, for example metal fixes after implementation. In most cases, this field is 0b0000. ARM recommends that the component designers ensure that a metal fix can change this field if required, for example, by driving it from registers that reset to 0b0000.	RO	0x0
3:0	CMOD	Customer Modified. Indicates whether the customer has modified the behavior of the component. In most cases, this field is 0b0000. Customers change this value when they make authorized modifications to this component.	RO	0x0

M33: CIDR0 Register

Offset: 0x42ff0

Description

CoreSight Component ID0

Table 352. CIDR0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_0	Preamble[0]. Contains bits[7:0] of the component identification code	RO	0x0d

M33: CIDR1 Register

Offset: 0x42ff4

Description

CoreSight Component ID1

Table 353. CIDR1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	CLASS	Class of the component, for example, whether the component is a ROM table or a generic CoreSight component. Contains bits[15:12] of the component identification code.	RO	0x9
3:0	PRMBL_1	Preamble[1]. Contains bits[11:8] of the component identification code.	RO	0x0

M33: CIDR2 Register

Offset: 0x42ff8

Description

CoreSight Component ID2

Table 354. CIDR2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_2	Preamble[2]. Contains bits[23:16] of the component identification code.	RO	0x05

M33: CIDR3 Register

Offset: 0x42ffc

Description

CoreSight Component ID3

Table 355. CIDR3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PRMBL_3	Preamble[3]. Contains bits[31:24] of the component identification code.	RO	0xb1

3.7.5.1. Cortex-M33 EPPB Registers

Table 356. List of M33_EPPB registers

Offset	Name	Info
0x0	NMI_MASK0	NMI mask for IRQs 0 through 31. This register is core-local, and is reset by a processor warm reset.

Offset	Name	Info
0x4	NMI_MASK1	NMI mask for IRQs 0 through 51. This register is core-local, and is reset by a processor warm reset.
0x8	SLEEPCTRL	Nonstandard sleep control register

M33_EPPB: NMI_MASK0 Register

Offset: 0x0

Table 357.
NMI_MASK0 Register

Bits	Description	Type	Reset
31:0	NMI mask for IRQs 0 through 31. This register is core-local, and is reset by a processor warm reset.	RW	0x00000000

M33_EPPB: NMI_MASK1 Register

Offset: 0x4

Table 358.
NMI_MASK1 Register

Bits	Description	Type	Reset
31:20	Reserved.	-	-
19:0	NMI mask for IRQs 0 through 51. This register is core-local, and is reset by a processor warm reset.	RW	0x00000

M33_EPPB: SLEEPCTRL Register

Offset: 0x8

Description

Nonstandard sleep control register

Table 359.
SLEEPCTRL Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	WICENACK	Status signal from the processor's interrupt controller. Changes to WICENREQ are eventually reflected in WICENACK.	RO	0x0
1	WICENREQ	Request that the next processor deep sleep is a WIC sleep. After setting this bit, before sleeping, poll WICENACK to ensure the processor interrupt controller has acknowledged the change.	RW	0x1
0	LIGHT_SLEEP	By default, any processor sleep will deassert the system-level clock request. Reenabling the clocks incurs 5 cycles of additional latency on wakeup. Setting LIGHT_SLEEP to 1 keeps the clock request asserted during a normal sleep (Arm SCR.SLEEPDEEP = 0), for faster wakeup. Processor deep sleep (Arm SCR.SLEEPDEEP = 1) is not affected, and will always deassert the system-level clock request.	RW	0x0

3.8. Hazard3 Processor

Hazard3 Source Code

The hardware source files for Hazard3 are available under Apache 2.0 licensing at:

<https://github.com/wren6991/hazard3>

Hazard3 is a low-area, high-performance RISC-V processor with a 3-stage in-order pipeline. RP2350 configures the following standard RISC-V extensions:

- **RV32I**: 32-bit base instruction set
- **M**: Integer multiply/divide/modulo instructions
- **A**: Atomic memory operations
- **C**: Compressed 16-bit instructions (equivalently spelled **Zca**)
- **Zba**: Address generation instructions
- **Zbb**: Basic bit manipulation instructions
- **Zbs**: Single-bit manipulation instructions
- **Zbkb**: Basic bit manipulation for scalar cryptography
- **Zcb**: Basic additional compressed instructions
- **Zcmp**: Push/pop and double-move compressed instructions
- **Zicsr**: CSR access instructions
- Debug, Machine and User execution modes
- Physical Memory Protection unit (PMP) with eight regions, 32-byte granule, NAPOT
- External debug support with four instruction address triggers

Additionally, it enables the following Hazard3 custom extensions:

- **Xh3power**: Power management instructions and CSRs
- **Xh3bextm**: Bit-extract-multiple instruction (used in bootrom)
- **Xh3irq**: Local interrupt controller with nested, prioritised IRQ support
- **Xh3pmpm**: Unlocked M-mode PMP regions

3.8.1. Instruction Reference

This section is a programmer's reference guide for the instructions supported by Hazard3. It covers basic assembly syntax, instruction behaviour, ranges for immediate values, and conditions for instruction compression.

When an instruction has an optional 16-bit compressed form (due to the **C/Zca** or **Zcb** extension) the limitations of the compressed form are documented. If no such limitations are mentioned, it means the instruction is always a 32-bit opcode.

The pseudocode in this section is informative only, and is no replacement for the official RISC-V specifications in [Section 3.8.1.1](#). However, it should prove a useful mnemonic aid once you have read the specifications.

3.8.1.1. Links to RISC-V Specifications

This table links ratified versions of the base instruction set and extensions implemented by Hazard3. These are the authoritative reference for the instructions documented in this reference guide.

Extension	Specification
RV32I v2.1	Unprivileged ISA 20191213
M v2.0	Unprivileged ISA 20191213
A v2.1	Unprivileged ISA 20191213
C v2.0	Unprivileged ISA 20191213
Zicsr v2.0	Unprivileged ISA 20191213
Zifencei v2.0	Unprivileged ISA 20191213
Zba v1.0.0	Bit Manipulation ISA extensions 20210628
Zbb v1.0.0	Bit Manipulation ISA extensions 20210628
Zbs v1.0.0	Bit Manipulation ISA extensions 20210628
Zbkb v1.0.1	Scalar Cryptography ISA extensions 20220218
Zcb v1.0.3-1	Code Size Reduction extensions frozen v1.0.3-1
Zcmp v1.0.3-1	Code Size Reduction extensions frozen v1.0.3-1
Machine ISA v1.12	Privileged Architecture 20211203
Debug v0.13.2	RISC-V External Debug Support 20190322

You may also refer to the [RISC-V Assembly Programmer's Manual](#) for information on assembly syntax.

Consult the [source code](#) for detailed questions about implementation-defined behaviour, which is not covered by the RISC-V specifications. RP2350 uses version [86fc4e3](#), with metal ECOs for commits [2f6e983](#) and [af08c0b](#).

3.8.1.2. Conventions for Pseudocode

Pseudocode is in Verilog 2005 syntax, and uses the following conventions for variables:

- **rs1**, **rs2** and **rd** are 32-bit unsigned vectors, referring to the two register operands and the destination register
- **imm** is a 32-bit unsigned vector referring to the instruction's immediate value
- **pc** is a 32-bit unsigned vector referring to the program counter, which is exactly the address of the current instruction
- **mem** is an array of 8-bit unsigned vectors, each corresponding to a byte address in memory
- **i** and **j** are working variables of type **integer** which may be used for loop variables

The pseudocode uses `<=` nonblocking assignments to assign to outputs: all such assignments are applied in a batch after the block of pseudocode has executed. Local variables may be assigned with `=` blocking assignments, which update the assignee immediately, similar to `=` procedural assignments in e.g. C programs. This distinction is important in some cases where e.g. **rd** and **rs1** may alias the same register, but it's generally sufficient just to be aware that **a <= b** and **a = b** are both assignments into **a**.

Other operators used widely in this section:

- Infix operators `+`, `-`, `*`, `/`, `&`, `^`, `|`, `<<`, `==`, `!=`, `<` and `>=` can be considered the same as the corresponding C operator
- `$signed()` bit-casts to a signed value; comparisons between two signed values are signed comparisons, and this reference generally avoids signed values for other operations (it's complicated)
- `>>` is always a logical (zero-extending) right shift
- `>>>` on a signed value is an arithmetic (sign-extending) right shift

- $\{a, b\}$ is the bit-concatenation of a and b , with a in the more-significant position of the result
- $a[m:l]$ is a bit slice of a , where m is the (inclusive) MSB and l is the (inclusive) LSB. For example $a[7:0]$ is the 8 least-significant bits of a .

3.8.1.3. Conventions for Registers

Register operands $rs1$, $rs2$ and rd in instruction pseudocode refer to any of the general-purpose integer registers $x0$ through $x31$, or equivalently to the corresponding ABI name for the same register:

Register	ABI Name	Description
$x0$	zero	Hardwired to zero; ignores writes
$x1$	ra	Return address (link register)
$x2$	sp	Stack pointer
$x3$	gp	Global pointer
$x4$	tp	Thread pointer
$x5 - x7$	t0 - t2	Temporaries
$x8$	s0 or fp	Saved register or frame pointer
$x9$	s1	Saved register
$x10 - x11$	a0 - a1	Function arguments and return values
$x12 - x17$	a2 - a7	Function arguments
$x18 - x27$	s2 - s11	Saved registers
$x28 - x31$	t3 - t6	Temporaries

Registers $x1$ through $x31$ are identical, and any 32-bit opcode can use any combination of these registers. However, compressed instructions give preferential treatment to commonly-used registers sp , ra , $s0$, $s1$ and $a0$ through $a5$ to improve code density. All compressed instructions implemented by Hazard3 are 16-bit aliases for existing 32-bit instructions, so you can still perform any operation on any register.

See the [RISC-V PSABI Specification](#) for more information on the ABI register assignment as well as the RISC-V procedure calling convention.

3.8.1.4. Alphabetical List of Instructions

This instruction reference covers all instructions from all extensions which Hazard3 implements on RP2350. The table below also includes common pseudo-instructions such as `not` and `ret`: the links go to the entry for the underlying hardware instruction aliased by that pseudo-instruction.

Alphabetical order: left-to-right, then top-to-bottom.					
add	addi	amoadd.w	amoand.w	amomax.w	amomaxu.w
amomin.w	amominu.w	amoor.w	amoswap.w	amoxor.w	and
andi	andn	auipc	bclr	bclri	beq
beqz	bext	bexti	bge	bgeu	bgez
bgt	bgtu	bgtz	binv	binvi	ble
bleu	blez	blt	bltu	bltz	bne
bnez	brev8	bset	bseti	clz	cm.mva01s

Alphabetical order: left-to-right, then top-to-bottom.					
cm.mvs01	cm.pop	cm.popret	cm.popretz	cm.push	cpop
ctz	div	divu	j	jal	jalr
jr	lb	lbu	lh	lhu	lr.w
lui	lw	max	maxu	min	minu
mul	mulh	mulhsu	mulhu	mv	neg
nop	not	or	orc.b	ori	orn
pack	packh	rem	remu	ret	rev8
rol	ror	rori	sb	sc.w	seqz
sext.b	sext.h	sgtz	sh1add	sh2add	sh3add
sh	sll	slli	slt	slti	sltiu
sltu	sltz	snez	sra	srai	srl
srl	sub	sw	unzip	xnor	xor
xori	zext.b	zext.h	zip		

The remainder of this reference guide groups instructions by extension:

- RV32I: base ISA (register-register)
- RV32I: base ISA (register-immediate)
- RV32I: base ISA (large immediate)
- RV32I: base ISA (control transfer)
- RV32I: base ISA (load/store)
- M: multiply and divide
- A: atomics
- C: compressed instructions
- Zba: bit manipulation for address generation
- Zbb: basic bit manipulation
- Zbs: single bit manipulation
- Zbkb: basic bit manipulation for scalar cryptography
- Zcb: additional basic compressed instructions
- Zcmp: compressed push, pop and double-move

3.8.1.5. RV32I: Base ISA (Register-register)

3.8.1.5.1. add

Add register to register.

Syntax:

```
add rd, rs1, rs2
```

Operation:

```
rd <= rs1 + rs2;
```

Compressible if either:

- **rd** matches **rs1**, no operands are **zero** (aka **c.add**)
- **rs2** is zero and neither **rd** nor **rs1** is **zero** (aka **c.mv**)

3.8.1.5.2. sub

Subtract register from register.

Syntax:

```
sub rd, rs1, rs2
neg rd, rs2      // pseudo: rs1 is zero
```

Operation:

```
rd <= rs1 - rs2;
```

Compressible if: **rd** matches **rs1**, registers are in **x8 - x15**.

3.8.1.5.3. slt

Set if less than (signed).

Syntax:

```
slt rd, rs1, rs2
sltz rd, rs1      // pseudo: rs2 is zero
sgtz rd, rs2      // pseudo: rs1 is zero
```

Operation:

```
rd <= $signed(rs1) < $signed(rs2);
```

3.8.1.5.4. sltu

Set if less than (unsigned).

Syntax:

```
sltu rd, rs1, rs2
snez rd, rs2      // pseudo: rs1 is zero
```

Operation:

```
rd <= rs1 < rs2;
```

3.8.1.5.5. and

Bitwise AND.

Syntax:

```
and rd, rs1, rs2
```

Operation:

```
rd <= rs1 & rs2;
```

Compressible if: **rd** matches **rs1**, registers are in **x8 - x15**.

3.8.1.5.6. or

Bitwise OR.

Syntax:

```
or rd, rs1, rs2
```

Operation:

```
rd <= rs1 | rs2;
```

Compressible if: **rd** matches **rs1**, registers are in **x8 - x15**.

3.8.1.5.7. xor

Bitwise XOR.

Syntax:

```
xor rd, rs1, rs2
```

Operation:

```
rd <= rs1 ^ rs2;
```

Compressible if: **rd** matches **rs1**, registers are in **x8 - x15**.

3.8.1.5.8. sll

Shift left, logical. Shift amount is modulo 32.

Syntax:

```
sll rd, rs1, rs2
```

Operation:

```
rd <= rs1 << rs2[4:0];
```

3.8.1.5.9. srl

Shift right, logical. Shift amount is modulo 32.

Syntax:

```
srl rd, rs1, rs2
```

Operation:

```
rd <= rs1 >> rs2[4:0];
```

3.8.1.5.10. sra

Shift right, arithmetic. Shift amount is modulo 32.

Syntax:

```
sra rd, rs1, rs2
```

Operation:

```
rd <= $signed(rs1) >>> rs2[4:0];
```

3.8.1.6. RV32I: Base ISA (Register-immediate)**3.8.1.6.1. addi**

Add register to immediate.

Syntax:

```
addi rd, rs1, imm
mv rd, rs1      // pseudo: imm is 0
nop            // pseudo: rd, rs1 are zero, imm is 0
```

Operation:

```
rd <= rs1 + imm
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, smaller for 16-bit.

Compressible if:

- `rd` matches `rs1`, and immediate is in the range `-0x20` through `0x1f` (aka `c.addi`)
- `rd` is not `zero`, `rs1` is `zero`, and immediate is in the range `-0x20` through `0x1f` (aka `c.li`)
- `rd` is in `x8 - x15`, `rs1` is `sp`, and immediate is a nonzero multiple of four in the range `0x000` through `0x3fc` (aka `c.addi4spn`)
- `rd` is `sp`, `rs1` is `sp`, and immediate is a nonzero multiple of 16 in the range `-0x200` through `0x1f0` (aka `c.addi16sp`)

Note compressed `c.mv` canonically expands to `add`, not `addi`.

3.8.1.6.2. slti

Set if less than immediate (signed).

Syntax:

```
slti rd, rs1, imm
```

Operation:

```
rd <= $signed(rs1) < $signed(imm);
```

Immediate range: `-0x800` through `0x7ff`

3.8.1.6.3. sltiu

Set if less than immediate (unsigned).

Syntax:

```
sltiu rd, rs1, imm
seqz rd, rs1      // pseudo: imm is 1
```

Operation:

```
rd <= rs1 < imm;
```

Immediate range: `-0x800` through `0x7ff`

3.8.1.6.4. andi

Bitwise AND with immediate.

Syntax:

```
andi rd, rs1, imm
zext.b rd, rs1      // pseudo: imm is 0xff
```

Operation:

```
rd <= rs1 & imm;
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, `-0x20` through `0x1f` for 16-bit.

Compressible if: `rd` matches `rs1`, registers are in `x8 - x15`, and immediate is in the range `-0x20` through `0x1f`.

3.8.1.6.5. ori

Bitwise OR with immediate.

Syntax:

```
ori rd, rs1, imm
```

Operation:

```
rd <= rs1 | imm;
```

Immediate range: `-0x800` through `0x7ff`

3.8.1.6.6. xori

Bitwise XOR with immediate.

Syntax:

```
xori rd, rs1, imm
not rd, rs1      // pseudo: imm is -1
```

Operation:

```
rd <= rs1 ^ imm;
```

Immediate range: `-0x800` through `0x7ff`

Compressible if: `rd` matches `rs1`, registers are in `x8 - x15`, and immediate is `-1` (aka `c.not`)

3.8.1.6.7. slli

Shift left, logical, immediate.

Syntax:

```
slli rd, rs1, imm
```

Operation:

```
rd <= rs1 << imm;
```

Immediate range: **0** through **31**.

Compressible if: **rd** matches **rs1**, registers are not **zero**.

3.8.1.6.8. srl

Shift right, logical, immediate.

Syntax:

```
srl rd, rs1, imm
```

Operation:

```
rd <= rs1 >> imm;
```

Immediate range: **0** through **31**.

Compressible if: **rd** matches **rs1**, registers are in **x8** through **x15**.

3.8.1.6.9. srai

Shift right, arithmetic, immediate.

Syntax:

```
srai rd, rs1, imm
```

Operation:

```
rd <= $signed(rs1) >>> imm;
```

Immediate range: **0** through **31**.

Compressible if: **rd** matches **rs1**, registers are in **x8** through **x15**.

3.8.1.7. RV32I: Base ISA (Large Immediate)

3.8.1.7.1. lui

Load upper immediate.

Syntax:

```
lui rd, imm
```

Operation:

```
rd <= imm << 12;
```

Immediate range: `-0x80000` through `0x7ffff` if 32-bit, `-0x20` through `0x1f` if 16-bit.

Compressible if: `rd` is neither `zero` nor `sp`, and `imm` is nonzero in the range `-0x20` through `0x1f`.

Note `-0x80000` through `-0x00001` are equivalent to `0x80000` through `0xfffff` after the left shift (on RV32 only) and the assembler may also accept these positive values.

3.8.1.7.2. auipc

Add upper immediate to program counter.

Syntax:

```
auipc rd, imm
```

Operation:

```
rd <= pc + imm << 12;
```

Immediate range: `-0x80000` through `0x7ffff`.

Note `-0x80000` through `-0x00001` are equivalent to `0x80000` through `0xfffff` after the left shift (on RV32 only) and the assembler may also accept these positive values.

3.8.1.8. RV32I: Base ISA (Control Transfer)

3.8.1.8.1. jal

Jump and link.

Syntax:

```
jal rd, label
jal label      // pseudo: rd is ra
j label        // pseudo: rd is zero
```

Operation:

```
rd <= pc + 4;  // or +2 if opcode is 16-bit
pc <= label;
```

Immediate range: even values in the range `-0x100000` through `0x0ffff` (± 1 MiB) if 32-bit, or `-0x800` through `0x7fe` (± 2 kB)

if 16-bit.

Compressible if: **rd** is **zero** or **ra**, and immediate is in the range **-0x800** through **0x7fe**.

3.8.1.8.2. jalr

Jump and link, target is register.

Syntax:

```
jalr rd, rs1, imm // imm is implicitly 0 if omitted.
jalr rd, imm(rs1) // imm is implicitly 0 if omitted.
jalr rs1, imm     // pseudo: rd is ra. imm is implicitly 0 if omitted.
jalr imm(rs1)     // pseudo: rd is ra. imm is implicitly 0 if omitted.
jr rs1, imm       // pseudo: rd is zero. imm is implicitly 0 if omitted.
jr imm(rs1)       // pseudo: rd is zero. imm is implicitly 0 if omitted.
ret              // pseudo for jr ra
```

Operation:

```
rd <= pc + 4;      // or +2 if opcode is 16-bit
pc <= rs1 + imm;
```

Immediate range: **-0x800** through **0x7ff**.

Compressible if: **rd** is **zero** or **ra**, immediate is zero, and **rs1** is not **zero**.

3.8.1.8.3. beq

Branch if equal.

Syntax:

```
beq rs1, rs2, label
beqz rs1, label    // pseudo: rs2 is zero
```

Operation:

```
if (rs1 == rs2)
    pc <= label;
```

Immediate range: even values in the range **-0x1000** through **0x0ffe** (± 4 kB) if 32-bit, or **-0x100** through **0x0fe** (± 256 B) if 16-bit.

Compressible if: **rs2** is **zero**, and immediate is in the range **-0x100** through **0x0fe** (aka **c.beqz**).

3.8.1.8.4. bne

Branch if not equal.

Syntax:

```
bne rs1, rs2, label
```

```
bnez rs1, label      // pseudo: rs2 is zero
```

Operation:

```
if (rs1 != rs2)
    pc <= label;
```

Immediate range: even values in the range `-0x1000` through `0x0ffe` (± 4 kB) if 32-bit, or `-0x100` through `0x0fe` (± 256 B) if 16-bit.

Compressible if: `rs2` is zero, and immediate is in the range `-0x100` through `0x0fe` (aka `c.bnez`).

3.8.1.8.5. blt

Branch if less than (signed).

Syntax:

```
blt rs1, rs2, label
bltz rs1, label      // pseudo: rs2 is zero
bgt rs2, rs1, label // pseudo: operands swapped by assembler
bgtz rs2, label     // pseudo: rs1 is zero
```

Operation:

```
if ($signed(rs1) < $signed(rs2))
    pc <= label;
```

Immediate range: even values in the range `-0x1000` through `0x0ffe` (± 4 kB)

3.8.1.8.6. bge

Branch if greater than or equal (signed).

Syntax:

```
bge rs1, rs2, label
bgez rs1, label      // pseudo: rs2 is zero
ble rs2, rs1, label // pseudo: operands swapped by assembler
blez rs2, label     // pseudo: rs1 is zero
```

Operation:

```
if ($signed(rs1) >= $signed(rs2))
    pc <= label;
```

Immediate range: even values in the range `-0x1000` through `0x0ffe` (± 4 kB)

3.8.1.8.7. bltu

Branch if less than (unsigned).

Syntax:

```
bltu rs1, rs2, label
bgtu rs2, rs1, label // pseudo: operands swapped by assembler
```

Operation:

```
if (rs1 < rs2)
    pc <= label;
```

Immediate range: even values in the range `-0x1000` through `0x0ffe` (± 4 kiB)

3.8.1.8.8. bgeu

Branch if less than or equal (unsigned).

Syntax:

```
bgeu rs1, rs2, label
bleu rs2, rs1, label // pseudo: operands swapped by assembler
```

Operation:

```
if (rs1 >= rs2)
    pc <= label;
```

Immediate range: even values in the range `-0x1000` through `0x0ffe` (± 4 kiB)

3.8.1.9. RV32I: Base ISA (Load and Store)

3.8.1.9.1. lw

Load word.

Syntax:

```
lw rd, imm(rs1)
lw rd, (rs1)    // imm is implicitly 0 if omitted.
```

Operation:

```
rd <= {
    mem[rs1 + imm + 3],
    mem[rs1 + imm + 2],
    mem[rs1 + imm + 1],
    mem[rs1 + imm]
```

```
};
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, smaller for 16-bit.

Compressible if:

- `rd` and `rs1` are in `x8 - x15`, and immediate is a multiple of four in the range `-0x40` through `0x3c` (aka `c.lw`)
- `rd` is not `zero`, `rs1` is `sp`, and immediate is a multiple of four in the range `0x00` through `0xfc` (aka `c.lwsp`)

3.8.1.9.2. lh

Load halfword (signed).

Syntax:

```
lh rd, imm(rs1)
lh rd, (rs1) // imm is implicitly 0 if omitted.
```

Operation:

```
rd <= {
    16{mem[rs1 + imm + 1][7]}, // Sign-extend
    mem[rs1 + imm + 1],
    mem[rs1 + imm]
};
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, or even values in the range `0x0` through `0x2` for 16-bit.

Compressible if: `rd` and `rs1` are in `x8` through `x15`, and immediate is `0x0` or `0x2`.

3.8.1.9.3. lhu

Load halfword (unsigned).

Syntax:

```
lhu rd, imm(rs1)
lhu rd, (rs1) // imm is implicitly 0 if omitted.
```

Operation:

```
rd <= {
    16'h0000, // Zero-extend
    mem[rs1 + imm + 1],
    mem[rs1 + imm]
};
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, or even values in the range `0x0` through `0x2` for 16-bit.

Compressible if: `rd` and `rs1` are in `x8` through `x15`, and immediate is `0x0` or `0x2`.

3.8.1.9.4. lb

Load byte (signed).

Syntax:

```
lb rd, imm(rs1)
lb rd, (rs1) // imm is implicitly 0 if omitted.
```

Operation:

```
rd <= {
  {24{mem[rs1 + imm][7]}}, // Sign-extend
  mem[rs1 + imm]
};
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, or `0x0` through `0x3` for 16-bit.

Compressible if: `rd` and `rs1` are in `x8` through `x15`, and immediate is in the range `0x0` through `0x3`.

3.8.1.9.5. lbu

Load byte (unsigned).

Syntax:

```
lbu rd, imm(rs1)
lbu rd, (rs1) // imm is implicitly 0 if omitted.
```

Operation:

```
rd <= {
  24'h000000, // Zero-extend
  mem[rs1 + imm]
};
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, or `0x0` through `0x3` for 16-bit.

Compressible if: `rd` and `rs1` are in `x8` through `x15`, and immediate is in the range `0x0` through `0x3`.

3.8.1.9.6. sw

Store word.

Syntax:

```
sw rs2, imm(rs1)
sw rs2, (rs1) // imm is implicitly 0 if omitted.
```

Operation:

```
mem[rs1 + imm] <= rs2[7:0];
mem[rs1 + imm + 1] <= rs2[15:8];
```

```
mem[rs1 + imm + 2] <= rs2[23:16];
mem[rs1 + imm + 3] <= rs2[31:24];
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, smaller for 16-bit.

Compressible if:

- `rs1` and `rs2` are in `x8 - x15`, and immediate is a multiple of four in the range `-0x40` through `0x3c` (aka `c.sw`)
- `rs2` is not zero, `rs1` is `sp`, and immediate is a multiple of four in the range `0x00` through `0xfc` (aka `c.swsp`)

3.8.1.9.7. sh

Store halfword.

Syntax:

```
sh rs2, imm(rs1)
sh rs2, (rs1) // imm is implicitly 0 if omitted.
```

Operation:

```
mem[rs1 + imm] <= rs2[7:0];
mem[rs1 + imm + 1] <= rs2[15:8];
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, or even values in the range `0x0` through `0x2` for 16-bit.

Compressible if: `rd` and `rs1` are in `x8` through `x15`, and immediate is `0x0` or `0x2`.

3.8.1.9.8. sb

Store byte.

Syntax:

```
sb rs2, imm(rs1)
sb rs2, (rs1) // imm is implicitly 0 if omitted.
```

Operation:

```
mem[rs1 + imm] <= rs2[7:0];
```

Immediate range: `-0x800` through `0x7ff` for 32-bit, or `0x0` through `0x3` for 16-bit.

Compressible if: `rd` and `rs1` are in `x8` through `x15`, and immediate is in the range `0x0` through `0x3`.

3.8.1.10. M: Multiply and Divide

3.8.1.10.1. mul

Multiply $32 \times 32 \rightarrow 32$.

Syntax:

```
mul rd, rs1, rs2
```

Operation:

```
rd <= rs1 * rs2;
```

Compressible if: `rd` matches `rs1`, registers are in `x8` through `x15`.

3.8.1.10.2. mulh

Multiply signed (32) by signed (32), return upper 32 bits of the 64-bit result.

Syntax:

```
mulh rd, rs1, rs2
```

Operation:

```
// Both operands are sign-extended to 64 bits:
wire [63:0] result_full = {{32{rs1[31]}}, rs1} * {{32{rs2[31]}}, rs2};
rd <= result_full[63:32];
```

3.8.1.10.3. mulhsu

Multiply signed (32) by unsigned (32), return upper 32 bits of the 64-bit result.

Syntax:

```
mulhsu rd, rs1, rs2
```

Operation:

```
// rs1 is sign-extended, rs2 is zero-extended:
wire [63:0] result_full = {{32{rs1[31]}}, rs1} * {32'h00000000, rs2};
rd <= result_full[63:32];
```

3.8.1.10.4. mulhu

Multiply unsigned (32) by unsigned (32), return upper 32 bits of the 64-bit result.

Syntax:

```
mulhu rd, rs1, rs2
```

Operation:

```
wire [63:0] result_full = {32'h00000000, rs1} * {32'h00000000, rs2};
rd <= result_full[63:32];
```

3.8.1.10.5. div

Divide (signed).

Syntax:

```
div rd, rs1, rs2
```

Operation:

```
if (rs2 == 32'h0)
    rd <= 32'hffffffff;
else if (rs1 == 32'h80000000 && rs2 == 32'hffffffff) // Signed overflow
    rd <= 32'h80000000;
else
    rd <= $signed(rs1) / $signed(rs2);
```

3.8.1.10.6. divu

Divide (unsigned).

Syntax:

```
divu rd, rs1, rs2
```

Operation:

```
if (rs2 == 32'h0)
    rd <= 32'hffffffff;
else
    rd <= rs1 / rs2;
```

3.8.1.10.7. rem

Remainder (signed).

Syntax:

```
rem rd, rs1, rs2
```

Operation:

```
if (rs2 == 32'h0)
    rd <= rs1;
else
```

```
rd <= $signed(rs1) % $signed(rs2);
```

3.8.1.10.8. remu

Remainder (unsigned).

Syntax:

```
remu rd, rs1, rs2
```

Operation:

```
if (rs2 == 32'h0)
    rd <= rs1;
else
    rd <= rs1 % rs2;
```

3.8.1.11. A: Atomics

A adds two groups of instructions:

- **lr.w** and **sc.w**, load-reserved and store-conditional instructions, which allow software to safely perform read-modify-write operations on shared variables by looping until success
- **amo*.w** instructions, which atomically modify a memory location and return its original value

See [Section 3.8.1.1](#) for a link to the RISC-V unprivileged ISA manual, which fully describes the **A** extension. See [Section 3.8.5](#) for cycle counts for these instructions on Hazard3.

3.8.1.12. C: Compressed Instructions

All instructions in the **C** extension are 16-bit aliases of 32-bit instructions from other extensions. In the case of Hazard3, which lacks the **F** extension, these are all aliases of base **I** instructions. They behave identically to their 32-bit counterparts.

C adds compressed aliases for the following instructions from RV32I:

Alphabetical order: left-to-right, then top-to-bottom.					
add	addi	and	andi	beq	bne
ebreak	jal	jalr	lui	lw	or
slli	srai	srli	sub	sw	xor

See the per-instruction documentation for the compression limitations of each instruction. The assembler automatically uses compressed variants when the limitations are met, and when the relevant compressed instruction extension is enabled for the assembler, for example by passing **c** in the **-march** ISA string.

The above also applies to **Zca** and **Zcb**: the former is an alias for the non-floating-point subset of **C**, and the latter adds 16-bit aliases for additional common instructions from the **I**, **M** and **Zbb** extensions. Each **Zcmp** instruction expands to a sequence of multiple instructions from the **I** extension.

3.8.1.13. Zba: Bit manipulation (address generation)

3.8.1.13.1. sh1add

Add, with the first addend shifted left by 1.

Syntax:

```
sh1add rd, rs1, rs2
```

Operation:

```
rd <= (rs1 << 1) + rs2;
```

3.8.1.13.2. sh2add

Add, with the first addend shifted left by 2.

Syntax:

```
sh2add rd, rs1, rs2
```

Operation:

```
rd <= (rs1 << 2) + rs2;
```

3.8.1.13.3. sh3add

Add, with the first addend shifted left by 3.

Syntax:

```
sh3add rd, rs1, rs2
```

Operation:

```
rd <= (rs1 << 3) + rs2;
```

3.8.1.14. Zbb: Bit manipulation (basic)

3.8.1.14.1. andn

Bitwise AND with inverted operand.

Syntax:

```
andn rd, rs1, rs2
```

Operation:

```
rd <= rs1 & ~rs2;
```

3.8.1.14.2. clz

Count leading zeroes (starting from MSB, searching LSB-ward).

Syntax:

```
clz rd, rs1
```

Operation:

```
rd <= 32;           // Default = 32 if no set bits
reg found = 1'b0; // Local variable

for (i = 0; i < 32; i = i + 1) begin
    if (rs1[31 - i] && !found) begin
        found = 1'b1;
        rd <= i;
    end
end
```

3.8.1.14.3. cpop

Population count.

Syntax:

```
cpop rd, rs1
```

Operation:

```
reg [5:0] sum = 6'd0;           // Local variable
for (i = 0; i < 32; i = i + 1)
    sum = sum + rs1[i];
rd <= sum;
```

3.8.1.14.4. ctz

Count trailing zeroes (starting from LSB, searching MSB-ward).

Syntax:

```
ctz rd, rs1
```

Operation:

```
rd <= 32;           // Default = 32 if no set bits
reg found = 1'b0; // Local variable

for (i = 0; i < 32; i = i + 1) begin
    if (rs1[i] && !found) begin
        found = 1'b1;
        rd <= i;
    end
end
```

3.8.1.14.5. max

Maximum of two values (signed).

Syntax:

```
max rd, rs1, rs2
```

Operation:

```
if ($signed(rs1) < $signed(rs2))
    rd <= rs2;
else
    rd <= rs1;
```

3.8.1.14.6. maxu

Maximum of two values (unsigned).

Syntax:

```
maxu rd, rs1, rs2
```

Operation:

```
if (rs1 < rs2)
    rd <= rs2;
else
    rd <= rs1;
```

3.8.1.14.7. min

Minimum of two values (signed).

Syntax:

```
min rd, rs1, rs2
```

Operation:

```
if ($signed(rs1) < $signed(rs2))
    rd <= rs1;
else
    rd <= rs2;
```

3.8.1.14.8. minu

Minimum of two values (unsigned).

Syntax:

```
minu rd, rs1, rs2
```

Operation:

```
if (rs1 < rs2)
    rd <= rs1;
else
    rd <= rs2;
```

3.8.1.14.9. orc.b

Or-combine of bits within each byte.

Syntax:

```
orc.b rd, rs1
```

Operation:

```
rd <= {
    {8{|rs1[31:24]}},
    {8{|rs1[23:16]}},
    {8{|rs1[15:8]}},
    {8{|rs1[7:0]}}}
```

3.8.1.14.10. orn

Bitwise OR with inverted operand.

Syntax:

```
orn rd, rs1, rs2
```

Operation:

```
rd <= rs1 | ~rs2;
```

3.8.1.14.11. rev8

Reverse bytes within word.

Syntax:

```
rev8 rd, rs1
```

Operation:

```
rd <= {
    rs1[7:0],
    rs1[15:8],
    rs1[23:16],
    rs1[31:24]
};
```

3.8.1.14.12. rol

Rotate left.

Syntax:

```
rol rd, rs1, rs2
```

Operation:

```
rd <= ({rs1, rs1} << rs2[4:0]) >> 32;
```

3.8.1.14.13. ror

Rotate right.

Syntax:

```
ror rd, rs1, rs2
```

Operation:

```
rd <= {rs1, rs1} >> rs2[4:0];
```

3.8.1.14.14. rori

Rotate right, immediate.

Syntax:

```
ror rd, rs1, imm
```

Operation:

```
rd <= {rs1, rs1} >> imm;
```

Immediate range: 0 through 31.

3.8.1.14.15. sext.b

Sign-extend from byte.

Syntax:

```
sext.b rd, rs1
```

Operation:

```
rd <= {
    {24{rs1[7]}},
    rs1[7:0]
};
```

Compressible if: **rd** matches **rs1**, and registers are in **x8 - x15**.

3.8.1.14.16. sext.h

Sign-extend from halfword.

Syntax:

```
sext.h rd, rs1
```

Operation:

```
rd <= {
    {16{rs1[15]}},
    rs1[15:0]
};
```

Compressible if: **rd** matches **rs1**, and registers are in **x8 - x15**.

3.8.1.14.17. xnor

Bitwise XOR with inverted operand.

Syntax:

```
xnor rd, rs1, rs2
```

Operation:

```
rd <= rs1 ^ ~rs2;
```

3.8.1.14.18. zext.b

Zero-extend from byte.

Syntax:

```
zext.b rd, rs1
```

Operation:

```
// Pseudo-op for RV32I instruction
andi rd, rs1, 0xff
```

Compressible if: **rd** matches **rs1**, and registers are in **x8 - x15**.

3.8.1.14.19. zext.h

Zero-extend from halfword.

Syntax:

```
zext.h rd, rs1
```

Operation:

```
rd <= {
    16'h0000,
    rs1[15:0]
};
```

Compressible if: **rd** matches **rs1**, and registers are in **x8 - x15**.

3.8.1.15. Zbs: Bit manipulation (single-bit)

3.8.1.15.1. bclr

Clear single bit.

Syntax:

```
bclr rd, rs1, rs2
```

Operation:

```
rd <= rs1 & ~(32'h1 << rs2[4:0]);
```

3.8.1.15.2. bclri

Clear single bit (immediate).

Syntax:

```
bclri rd, rs1, imm
```

Operation:

```
rd <= rs1 & ~(32'h1 << imm);
```

Immediate range: 0 through 31.

3.8.1.15.3. bext

Extract single bit.

Syntax:

```
bext rd, rs1, rs2
```

Operation:

```
rd <= (rs1 >> rs2[4:0]) & 32'h1;
```

3.8.1.15.4. bexti

Extract single bit (immediate).

Syntax:

```
bexti rd, rs1, imm
```

Operation:

```
rd <= (rs1 >> imm) & 32'h1;
```

Immediate range: 0 through 31.

3.8.1.15.5. binv

Invert single bit.

Syntax:

```
binv rd, rs1, rs2
```

Operation:

```
rd <= rs1 ^ (32'h1 << rs2[4:0]);
```

3.8.1.15.6. binvi

Invert single bit (immediate).

Syntax:

```
binvi rd, rs1, imm
```

Operation:

```
rd <= rs1 ^ (32'h1 << imm);
```

Immediate range: 0 through 31.

3.8.1.15.7. bset

Set single bit.

Syntax:

```
bset rd, rs1, rs2
```

Operation:

```
rd <= rs1 | (32'h1 << rs2[4:0])
```

3.8.1.15.8. bseti

Set single bit (immediate).

Syntax:

```
bseti rd, rs1, imm
```

Operation:

```
rd <= rs1 | (32'h1 << imm);
```

Immediate range: 0 through 31.

3.8.1.16. Zbkb: Basic bit manipulation for cryptography

Zbkb has a large overlap with Zbb (basic bit manipulation). This section covers instructions in Zbkb but not in Zbb.

3.8.1.16.1. brev8

Bit-reverse within each byte.

Syntax:

```
brev8 rd, rs1
```

Operation:

```
for (i = 0; i < 32; i = i + 8) begin
    for (j = 0; j < 8; j = j + 1) begin
        rd[i + j] <= rs1[i + (7 - j)];
    end
end
```

3.8.1.16.2. pack

Pack halfwords into word.

Syntax:

```
pack rd, rs1, rs2
```

Operation:

```
rd <= {
    rs2[15:0],
    rs1[15:0]
};
```

3.8.1.16.3. packh

Pack bytes into halfword.

Syntax:

```
packh rd, rs1, rs2
```

Operation:

```
rd <= {
    16'h0000,
    rs2[7:0],
    rs1[7:0]
};
```

3.8.1.16.4. zip

Interleave upper/lower half of register into odd/even bits of result.

Syntax:

```
zip rd, rs1
```

Operation:

```
for (i = 0; i < 32; i = i + 2) begin
    rd[i]    <= rs1[i / 2];
    rd[i + 1] <= rs1[i / 2 + 16];
end
```

3.8.1.16.5. unzip

Deinterleave odd/even bits of register into upper/lower half of result.

Syntax:

```
unzip rd, rs1
```

Operation:

```
for (i = 0; i < 32; i = i + 2) begin
    rd[i / 2]    <= rs1[i];
    rd[i / 2 + 16] <= rs1[i + 1];
end
```

3.8.1.17. Zcb: Additional Basic Compressed Instructions

Zcb adds 16-bit compressed aliases for the following instructions from the I, M and Zbb extensions:

Alphabetical order: left-to-right, then top-to-bottom.					
lbu	lh	lhu	mul	not	sb
sext.b	sext.h	sh	zext.h	zext.b	

See per-instruction documentation for the compressibility limitations for each instruction.

3.8.1.18. Zcmp: Compressed Push, Pop and Double Move

`Zcmp` adds 16-bit instructions which expand to common sequences of 32-bit RV32I instructions used in function prologues and epilogues: pushing to stack, popping from stack and moving arguments to/from saved registers.

See [Section 3.8.1.1](#) for a link to the `Zcmp` specification. See [Section 3.8.5](#) for cycle counts for these instructions on Hazard3.

3.8.2. Interrupts and Exceptions

In the RISC-V privileged ISA manual, a *trap* refers to either an exception or an interrupt:

- **Interrupt:** A signal from outside the processor requests that it temporarily abandons its current task to deal with some system-level event. The processor responds by transferring control to an interrupt handler function.
- **Exception:** when executing some particular instruction, the processor encountered a condition which prevented the instruction from completing normally. The processor transfers control to an exception handler to deal with the exceptional condition before it can resume execution.

The two are closely related, and they are collectively referred to as traps to avoid stating everything twice.

Hardware performs the following steps automatically and atomically when entering a trap:

1. Save the address of the interrupted or excepting instruction to `MEPC`
2. Set the MSB of `MCAUSE` to indicate the cause is an interrupt, or clear it to indicate an exception
3. Write the detailed trap cause to the LSBs of the `MCAUSE` register
4. Save the current privilege level to `MSTATUS.MPP`
5. Set the privilege to M-mode (note Hazard3 does not implement S-mode)
6. Save the current value of `MSTATUS.MIE` to `MSTATUS.MPIE`
7. Disable interrupts by clearing `MSTATUS.MIE`
8. Jump to the correct offset from `MTVEC` depending on the trap cause

NOTE

The above sequence of events is standard and is also described in the RISC-V Privileged ISA Manual. See [Section 3.8.1.1](#) for a list of links to RISC-V specifications.

All earlier instructions than the one pointed to by `MEPC` execute normally, and their effects are visible to the trap handler. These earlier instructions are not affected by the exception or interrupt. On the other hand the instruction pointed to by `MEPC`, and all later instructions, does not execute before entering the trap handler. These instructions have no visible side effects, with the possible exception of load/store fault exceptions, where the bus fault itself may have observable effects on the bus or peripheral.

Expanding on the `MEPC` behaviour in architectural terms, all traps are *precise*, meaning there exists some point in program order where the trap handler observes all earlier instructions to have retired and all later instructions to have not. All exceptions are also *synchronous*, meaning there is a particular instruction which originated the trap, and the trap architecturally takes place in between that instruction and its predecessors in program order.

M-mode software executes an `mret` instruction to return to the interrupted or excepting instruction at the end of a handler. This largely reverses the process of entering the trap:

1. Restore core privilege level to the value of `MSTATUS.MPP`
2. Write `0` (U-mode) to `MSTATUS.MPP`
3. Restore `MSTATUS.MIE` from `MSTATUS.MPIE`
4. Write `1` to `MSTATUS.MPIE`
5. Jump to the address in `MEPC`.

Often, the values restored on exit are exactly those values saved on entry. However this need not be the case, as all CSRs mentioned above are read/writable by M-mode software at any time. Hand-manipulating the trap handling CSRs is useful for low-level OS operations such as context switching, or to make exception handlers return to the instruction *after* the trap point by incrementing `MEPC` before return.

Hardware does not save or restore any other registers. In particular, it does not save the core GPRs, and software is responsible for ensuring the execution of the handler does not perturb the foreground context. For an interrupt, this may mean saving the core registers on the interruptee's stack, or using the `MSCRATCH` CSR to swap the stack pointer before saving registers on a dedicated interrupt stack. For a fatal exception this may be unimportant, as there is no requirement for the handler to return.

3.8.2.1. Exceptions

Exceptions occur for a variety of reasons. `MCAUSE` indicates the specific reason for the latest exception:

Cause	Meaning
<code>0x0</code>	Instruction alignment: Does not occur on RP2350, because 16-bit compressed instructions are implemented, and it is impossible to jump to a byte-aligned address.
<code>0x1</code>	Instruction fetch fault: Attempted to fetch from an address that does not support instruction fetch (like APB/AHB peripherals on RP2350), or lacks PMP execute permission, or is forbidden by ACCESSCTRL, or returned a fault from the memory device itself.
<code>0x2</code>	Illegal instruction: Encountered an instruction that was not a valid RISC-V opcode implemented by this processor, or attempted to access a nonexistent CSR, or attempted to execute a privileged instruction or access a privileged CSR without sufficient privilege.
<code>0x3</code>	Breakpoint: An <code>ebreak</code> or <code>c.ebreak</code> instruction was executed, and no external debug host caught it (<code>DCSR.EBREAKM</code> or <code>DCSR.EBREAKU</code> was not set).
<code>0x4</code>	Load alignment: Attempted to load from an address that was not a multiple of access size.
<code>0x5</code>	Load fault: Attempted to load from an address that does not exist, or lacks PMP read permissions, or is forbidden by ACCESSCTRL, or returned a fault from a peripheral.
<code>0x6</code>	Store/AMO alignment: Attempted to write to an address that was not a multiple of access size.
<code>0x7</code>	Store/AMO fault: Attempted to write to an address that does not exist, or lacks PMP write permissions, or is forbidden by ACCESSCTRL, or returned a fault from a peripheral. Also raised when attempting an AMO on an address that does not support AHB5 exclusives.
<code>0x8</code>	An <code>ecall</code> instruction was executed in U-mode.
<code>0xb</code>	An <code>ecall</code> instruction was executed in M-mode.

Exceptions jump to exactly the address of `MTVEC`, no matter the cause and no matter whether vectoring is enabled.

The `MSTATUS.MIE` global interrupt enable does not affect exception entry. You can still take an exception and trap into the exception handler when exceptions are disabled.

Returning from an exception will jump to `MEPC`, which hardware sets to the address of the excepting instruction before

entering the exception handler. This means by default you will return to the exact same instruction that caused the exception. When emulating illegal instructions, you should increment `mepc` before returning, so that execution resumes after the problematic instruction.

Hazard3 hardwires `mtval` to zero. To emulate a misaligned load/store instruction you must decode the instruction and read the spilled register state to calculate the address, and to emulate an illegal instruction you must read the instruction bits from memory yourself by dereferencing `mepc`.

3.8.2.2. Interrupts

Hazard3 implements the standard RISC-V interrupt scheme with a single external interrupt routed to `MIP.MEIP`, and the standard timer and soft interrupts routed to `MTIP` and `MSIP`. An interrupt controller such as a standard RISC-V PLIC can be integrated externally to route multiple interrupts through to the single external interrupt line. Alternatively, the Hazard3 interrupt controller (see `Xh3irq` extension, [Section 3.8.4.1](#)) multiplexes multiple external interrupts onto `MIP.MEIP` in such a way that interrupts can efficiently preempt one another, with configurable dynamic priority per interrupt.

RP2350 configures Hazard3 with the `Xh3irq` interrupt controller, with 52 external interrupt lines and 16 levels of preemption priority. The IRQ numbers for the system-level interrupts, documented in [Section 3.2](#), are the same on both Arm and RISC-V.

The core enters an interrupt when all of the following are true:

1. `MSTATUS.MIE` is set
2. An interrupt pending bit in the standard `MIP` CSR is set
3. The matching interrupt enable in the standard `MIE` CSR is also set

When vectoring is disabled (LSB of `MTVEC` is clear), interrupts transfer control directly to the address indicated by `mtvec`. Setting the LSB enables vectoring: interrupts transfer control to the address `mtvec + 4 * cause`, where the interrupt cause is one of:

- `meip: cause = 11`
- `mtip: cause = 7`
- `msip: cause = 3`

The pointer written to `mtvec` must be word-aligned (4 bytes). Additionally, when vectoring is enabled, it must be aligned to the size of the table, rounded up to a power of two. This works out to 64-byte alignment. On RP2350, `mtvec` is fully writable except for bit 1, which is hardwired to zero as it is only used for additional vectoring modes not supported by Hazard3.

When multiple interrupts are active, hardware picks one to enter, in the order `meip > msip > mtip`. (Note this is not quite the same order as the cause values.)

3.8.2.2.1. RISC-V Interrupt Signals

The standard timer interrupt `MIP.MTIP` connects to the RISC-V platform timer in the SIO subsystem ([Section 3.1.8](#)). This is a 64-bit timer with a per-core 64-bit comparison value. The interrupt is asserted whenever the timer is greater than or equal to the comparison value, and deasserts automatically when less than. The same interrupt signal also appears in the system-level IRQs, as `SIO_IRQ_MTIMECMP` (IRQ 40). The timer is a standard RISC-V peripheral, often used by operating systems to generate context switch interrupts.

The standard software interrupt `MIP.MSIP` connects to the `RISCV_SOFTIRQ` register in the SIO subsystem. The register has a single bit per hart (core), which asserts the soft IRQ interrupt to that hart. This can be used to interrupt the other hart, or to interrupt yourself as though the other hart had interrupted you, which can help to make handler code more symmetric.

Hazard3's internal interrupt controller drives the `MIP.MEIP` external interrupt pending bit based on its internal state and

the system-level interrupt signals, to transfer control to the interrupt vector when it is both safe and necessary. Section 3.8.4.1 describes the Xh3irq interrupt controller in depth.

3.8.2.2. Interrupt Calling Convention

The default SDK `hardware_irq` library expects function pointers registered for system-level IRQs to be normal C functions. There must be no `__attribute__((interrupt))` on an interrupt handler passed into functions such as `set_exclusive_irq_handler()`. This is an API detail that is consistent across all architectures supported by the SDK. Using regular C calling convention is also efficient under heavy interrupt load, because the cost of saving/restoring all caller save and temporary registers can be amortised over multiple interrupt handlers due to tail sharing, and a save triggered by a low-priority IRQ can be taken over by a high-priority IRQ that asserted during the save.

Conversely, handlers registered for the standard RISC-V `mtip` and `msip` interrupts via the SDK `irq_set_riscv_vector_handler()` function must be `__attribute__((interrupt))`. In terms of the generated code, this means they should use save-as-you-go calling convention, and end with an `mret`. These interrupts are entered directly by the hardware without any intermediate dispatch code.

As software is responsible for the dispatch to individual system interrupt handlers from the `meip` vector, it is possible to support other interrupt calling conventions by supplying a different implementation for the dispatch.

3.8.3. Debug

RP2350 implements a single RISC-V Debug Module, which enables debug access to the two Hazard3 processor instances. Hazard3 should be supported by any debug translator implementing version 0.13.2 of the RISC-V External Debug Support specification, but some details of its implementation-defined behaviour are described here for completeness. Note that the Debug Module source code, available in the Hazard3 repository, can be consulted to answer more detailed questions about the debug implementation.

RISC-V Debug Specification

Hazard3, and its accompanying Debug Module, implement version 0.13.2 of the RISC-V External Debug Support specification, available at:

<https://riscv.org/wp-content/uploads/2019/03/riscv-debug-release.pdf>

As configured on RP2350, Hazard3 supports the following standard RISC-V debug features:

- Run/halt/reset control of each processor
- Halt-on-reset support for all processors
- Hart array mask register, for halting/resuming multiple processors simultaneously
- Abstract access to GPRs
- Program Buffer: 2 words with an implicit `ebreak` (`impebbreak`)
- Automatic trigger of abstract commands (`abstractauto`)
- System Bus Access, arbitrated with core 1's load/store port
- An instruction address trigger unit with four hardware breakpoints

3.8.3.1. Accessing the Debug Module

The Debug Module is accessed through a CoreSight APB-AP which can be accessed in one of two ways:

- Externally, through the system's SW-DP (see Section 3.5)

- Internally, via self-hosted debug (see [Section 3.5.6](#))

The APB-AP for the Debug Module is located at offset `0xa000` in the debug address space. The Debug Module starts at address 0 in the APB-AP's downstream address space, and its registers are addressed in increments of four bytes, as APB is byte-addressed rather than word-addressed. This means the Debug Module register addresses listed in the RISC-V debug specification must be multiplied by four.

3.8.3.2. Harts

Each Hazard3 core possesses exactly one hardware thread, or *hart*. This means each processor executes only a single stream of instructions at a time. The two Hazard3 processor cores on RP2350, core 0 and 1, have hart IDs of 0 and 1 respectively. These values can be read from the [MHARTID](#) register on each processor, and match the values read from the [CPUID](#) register in SIO.

The `dmcontrol.hartsel` field in RP2350's Debug Module supports writing the values 0 and 1 only (it implements only a single writable bit), and these correspond to hart IDs 0 and 1, which execute on core 0 and core 1 respectively.

3.8.3.3. Resets

The `dmcontrol.hartreset` field resets the selected cores only. This can be a single core selected by `dmcontrol.hartsel`, or multiple cores selected by the hart array mask. It does not reset cores that are not selected, nor does it reset any other system hardware. Note that there is a one-to-one correspondence between harts and cores on this system.

The `dmcontrol.ndmreset` field resets both cores. It does not reset any other hardware. As per the specification: *"Exactly what is affected by this reset is implementation dependent, as long as it is possible to debug programs from the first instruction executed."*

3.8.3.4. Implementation-defined behaviour

The following are not implemented:

- Abstract access memory
- Abstract access CSR
- Post-incrementing abstract access GPR

The core behaves as follows:

- Branch, `jal`, `jalr` and `auipc` are illegal in debug mode, because they observe PC: attempting to execute will halt Program Buffer execution and report an exception in [abstractcs.cmderr](#)
- The `dret` instruction is not implemented (a special purpose DM-to-core signal is used to signal resume)
- The `dscratch` CSRs are not implemented
- The Debug Module's `data0` register is mapped into the core as a CSR, [DMDATA0](#)
- `dcsr.stepie` is hardwired to 0 (no interrupts during single stepping)
- `dcsr.stopcount` and `dcsr.stoptime` are hardwired to 1 (no counter or internal timer increment in debug mode)
- `dcsr.mprven` is hardwired to 0
- `dcsr.prv` accepts only the values 3 (M-mode) and 0 (U-mode), rounding to nearest on write

See also [DCSR](#) and [DPC](#) for more details on the core-side Debug Mode registers.

The trigger unit implements four exact instruction address match triggers. Triggers can be configured to trap to M-mode as well as Debug-mode, meaning M-mode can use triggers for self-hosted hardware breakpoint support. The `tcontrol.mte` and `tcontrol.mpte` fields are implemented to avoid infinite exception loops when an M-mode trigger is set on the M-mode exception handler.

3.8.4. Custom Extensions

Hazard3 implements a small number of custom extensions. All are optional: custom extensions are only included if the relevant feature flags are set to 1 when instantiating the processor (Section 3.8.6). Hazard3 is always a *conforming* RISC-V implementation, and when these extensions are disabled it is also a *standard* RISC-V implementation.

If any one of these extensions is enabled, the `x` bit in `MISA` is set to indicate the presence of a nonstandard extension.

3.8.4.1. Xh3irq: Hazard3 interrupt controller

Xh3irq controls up to 512 external interrupts, with up to 16 levels of preemption. It is architected as a layer on top of the standard `mip.meip` external interrupt line, and all standard RISC-V interrupt behaviour still applies. This extension adds no new instructions, but does add several CSRs:

- `MEIEA`: external interrupt enable array
- `MEIPA`: external interrupt pending array
- `MEIFA`: external interrupt force array
- `MEIPRA`: external interrupt priority array
- `MEINEXT`: get next external interrupt
- `MEICONTEXT`: external interrupt context register

Xh3irq is geared towards supporting interrupt handlers as bare C functions, with dispatch implemented in software and preemption priority logic implemented in hardware. However, the exact interrupt ABI is up to the implementation of the soft dispatch routine installed as the `mip.meip` external interrupt handler.

3.8.4.1.1. Array CSRs

RISC-V CSRs are ideal for interrupt controls because they are closely coupled to the processor, offer native atomic set/clear accesses, and can be accessed in a single instruction without first having to materialise an address. However there are issues with using CSRs for large bit arrays, such as interrupt enables:

- The CSR address space is limited
- CSRs can not be addressed indirectly, so are difficult to iterate over
- Using a CSR to index other CSRs is problematic for interrupt handlers due to additional mutable state

Xh3irq uses the *array CSR idiom* to expose a large bit vector at a single CSR address, such as `MEIEA`. The upper half of the CSR is a 16-bit window into the array, and the window is indexed by the LSBs of the write data for the same CSR instruction.

For example, the following assembly code writes `0xa5a5` to bits `47:32` of the interrupt enable array, since the window index is `0x2` and the window is 16 bits in size:

```
li a0, 0xa5a50002
csrw RVCSR_MEIEA_OFFSET, a0
```

The following reads bits 63:48 of the interrupt pending array into register `a0`, since the index is `0x3`, and a CSR set of `0x0000` does not modify the window contents:

```
csrrsi a0, RVCSR_MEIPA_OFFSET, 0x3
```

Setting an arbitrary IRQ enable from C works as follows:

```

void enable_irq(uint irq) {
    uint index = irq / 16;
    uint32_t mask = 1u << (irq % 16);
    asm (
        "csrs 0xbe0, %0\n"
        : : "r" (index | (mask << 16))
    );
}

```

Getting an arbitrary IRQ pending flag from C is as follows:

```

bool check_irq_pending(uint irq) {
    uint index = irq / 16;
    uint32_t csr_rdata;
    asm (
        "csrrs %0, 0xbe1, %1\n"
        : "=r" (csr_rdata)
        : "r" (index)
    );
    csr_rdata >>= 16;
    return csr_rdata & (1u << (irq % 16));
}

```

Note in the SDK these operations are already implemented by the [hardware_irq](#) API.

Hazard3 supports up to 512 interrupts, which is one 16-bit window for each of the possible values of a 5-bit CSR immediate.

3.8.4.1.2. Enable, Pending and Force Arrays

The [MEIEA](#), [MEIPA](#) and [MEIFA](#) CSRs expose the interrupt enable, pending and force arrays respectively. Each array contains one bit per system-level interrupt line, of which there are 52 lines in total. (See [Section 3.2](#) for the assignment of system IRQ numbers to peripherals.)

The interrupt enable array gates the entry of interrupt signals into the core. When a bit is clear in [MEIEA](#), the corresponding interrupt signal is ignored. When a bit is set, assertion of the corresponding interrupt signal will send the core to the [meip](#) vector as soon as it is safe and appropriate to do so. From there, the [meip](#) handler vectors to the correct handler, after saving the interruptee's context.

The SDK [irq_set_enabled\(\)](#) function in the [hardware_irq](#) library is a convenient way to manipulate the interrupt enable array.

The interrupt pending array displays the current status of the system-level interrupt signals. Interrupts are visible in [MEIPA](#) even if the corresponding bit is clear in [MEIEA](#), and even if the interrupt has insufficient priority to interrupt the core at this time. This register is read-only: bits in [MEIPA](#) clear automatically when the corresponding interrupt source deasserts. For example a UART RX FIFO interrupt should clear on its own once data has been read from the FIFO.

The interrupt force array causes interrupts to appear pending, even when the corresponding system-level interrupt signal is deasserted. When a bit is set in [MEIFA](#), the corresponding bit in [MEIPA](#) reads as 1, and will interrupt the core if it meets the usual prerequisites.

[MEIFA](#) bits clear automatically when the corresponding interrupt is sampled from [MEINEXT](#). It is not necessary to write a 1 bit to [MEINEXT.UPDATE](#) for the interrupt force bit to clear. This means setting an [MEIFA](#) bit should cause the interrupt to be taken once. Normal [csrw](#) and [csrc](#) instructions will also clear [MEIFA](#).

Six spare interrupt lines 46 through 51, referred to as [SPAREIRQ_IRQ_0](#) through [SPAREIRQ_IRQ_5](#) in the SDK, deliberately do not connect to system-level hardware. However they are still fully implemented in the interrupt controller, and fire when set

pending in [MEIFA](#). For example, a fast interrupt top-half handler may schedule its longer-running bottom half to run at a lower priority, or a high-priority context switch interrupt may schedule a context switch to take place at a lower priority in order to clear interrupt frames off the stack.

3.8.4.1.3. Next Interrupt Register

[MEINEXT](#) always displays the next interrupt that should be handled, taking priority order into account. Interrupts appear in [MEINEXT](#) when they meet all of the following criteria:

1. Pending in [MEIPA](#)
2. Enabled in [MEIEA](#)
3. Of priority greater than or equal to [MEICONTEXT.PPREEMPT](#)

The value returned is the IRQ number of the highest-priority interrupt that meets these three criteria, left-shifted by two. When multiple interrupts have the highest priority, the lowest-numbered of those interrupts is chosen, as a tie-break.

The MSB of [MEINEXT](#) is set to indicate there were no eligible interrupts, and the remaining bits are undefined in this case. Software should repeatedly read [MEINEXT](#) until all available interrupts are exhausted. The [bltz](#) and [bgez](#) instructions are a convenient way to test the MSB of a register.

The purpose of rule 3 above is to ensure that any interrupt that may already be in progress in a preempted interrupt frame is not re-entered in the current frame. Without this rule, multiple executions of the same interrupt handler could be interleaved due to preemption by other handlers. Programmers are usually surprised when this happens.

[MEINEXT.UPDATE](#) is a write-only field which instructs hardware to update [MEICONTEXT](#) with information about the interrupt displayed in [MEINEXT](#) on that cycle. [Section 3.8.4.1.5](#) goes into more detail about context register updates.

! IMPORTANT

[MEINEXT](#) is constantly changing as interrupt signals come and go. The write to [MEINEXT.UPDATE](#) must be the *same instruction* that reads the interrupt index from [MEINEXT](#) to avoid a data race. This can be achieved with a [csrrw](#) or [csrrwi](#) instruction.

3.8.4.1.4. Interrupt Priority

The interrupt priority array [MEIPRA](#) implements a four-bit field per interrupt. In hardware, numerically higher (unsigned) [MEIPRA](#) values have higher priority, taking precedence over lower-priority interrupts. The [irq_set_priority\(\)](#) SDK function uses the opposite convention, with lower numeric values indicating greater precedence. This section uses the hardware numbering.

The interrupt priority in [MEIPRA](#) determines three things:

1. Whether the interrupt source is permitted to interrupt the core at this moment: must be greater than or equal [MEICONTEXT.PREEMPT](#)
2. Whether the interrupt source can appear in [MEINEXT](#): must be greater than or equal to [MEICONTEXT.PPREEMPT](#)
3. What order this interrupt will appear in when there are multiple candidates for [MEINEXT](#)

When [MEICONTEXT](#) is correctly saved and restored, PREEMPT and PPREEMPT are both zero outside of interrupt handlers, and PREEMPT is strictly greater than PPREEMPT when inside an interrupt handler. Together they define the band of interrupt priorities which may be processed without any pushing or popping of interrupt stack frames.

Manipulating interrupt priority outside of interrupts is safe. There is no need to disable interrupts when writing to the priority array. Manipulating interrupt priority *inside* of an interrupt handler requires care: hardware operation is well-defined, but the results can be surprising. Be wary of the following cases:

1. Increasing the priority of the current handler: if still enabled and pending, you will instantly preempt yourself.
2. Increasing the priority of a different interrupt, with priority lower than [MEICONTEXT.PPREEMPT](#): this interrupt may

already be in progress in a frame that was preempted in order to run your handler. Increasing the priority may cause it to execute in a higher frame before returning to the original frame where it is still in progress, thereby interleaving with its own execution.

Note PPREEMPT is guaranteed to be no greater than the current handler priority if [MEICONTEXT](#) is correctly saved/restored, since it contains the previous value of PREEMPT at the time a preemption took place, and interrupts lower than PREEMPT can not interrupt the core. Therefore a safe approximation for case 2 above is: do not increase (by any amount) the priority of a handler with lower priority than the currently running handler.

If an interrupt *must* increase the priority of a lower-priority interrupt, one solution is to queue up interrupt priority updates, and pend a lowest-priority handler assigned to one of the spare IRQs, which processes the enqueued updates. You can pend this handler manually by setting its bit in [MEIFA](#). The handler will run last thing before returning to foreground code. This is safe because an interrupt of the lowest priority by definition can not have preempted any other interrupts.

3.8.4.1.5. Interrupt Context Management

The [MEICONTEXT](#) register has two functions: manage the core preemption priority across multiple preempting interrupt stack frames, and help software track which interrupt handler it is currently executing, if any.

[MEICONTEXT.PREEMPT](#), PPREEMPT and PPPREEMPT form a three-level stack of preemption priorities:

- PREEMPT sets the minimum interrupt priority which interrupts the core
- PPREEMPT sets the minimum interrupt priority which appears in [MEINEXT](#): this avoids redundant execution of interrupt handlers which may have been preempted
- PPPREEMPT has no hardware function other than save/restore of PPREEMPT

When entering the [MIP.MEIP](#) vector, hardware atomically performs the following updates to [MEICONTEXT](#) simultaneous to the standard trap entry sequence described in [Section 3.8.2](#):

1. Save the current value of [MEICONTEXT.PPREEMPT](#) to PPPREEMPT
2. Save the current value of [MEICONTEXT.PREEMPT](#) to PPREEMPT
3. Write one plus the priority of the IRQ which caused this interrupt to [MEICONTEXT.PREEMPT](#)
4. Write 1 to [MEICONTEXT.MRETEIRQ](#), to enable priority restore on next [mret](#)

The standard trap entry sequence includes clearing [MSTATUS.MIE](#), so interrupts are disabled at the start of the handler. To implement preemption, the [MIP.MEIP](#) handler must re-enable interrupts after its context save critical section. This should include saving [MEICONTEXT](#), [MSTATUS](#), [MEPC](#), and the caller-saved general-purpose registers.

Any trap entry not caused by [MIP.MEIP](#) clears MRETEIRQ. Trap exit ([mret](#)) also clears MRETEIRQ.

A trap exit where [MEICONTEXT.MRETEIRQ](#) is set atomically performs the following updates to [MEICONTEXT](#) simultaneous to the standard trap exit sequence:

1. Restore [MEICONTEXT.PREEMPT](#) from [MEICONTEXT.PPREEMPT](#)
2. Restore [MEICONTEXT.PPREEMPT](#) from [MEICONTEXT.PPPREEMPT](#)
3. Write 0 to [MEICONTEXT.PPPREEMPT](#)

The MRETEIRQ flag allows hardware to match each [MIP.MEIP](#) vector entry with its associated [mret](#). This balances pushes and pops of the PREEMPT priority stack. When there is no preemption, and no exceptions raised within interrupt handlers, MRETEIRQ can be left in place in the [MEICONTEXT.MRETEIRQ](#) register. Otherwise, you must save [MEICONTEXT](#) upon entering the external interrupt vector and restore it before the [mret](#) at the end of the handler. Interrupts must be disabled during save/restore.

Writing 1 to [MEINEXT.UPDATE](#) updates [MEICONTEXT](#) as follows:

1. Write [MEINEXT.NOIRQ](#) to [MEICONTEXT.NOIRQ](#)

2. Write **MEINEXT.IRQ** (the IRQ number) to **MEICONTEXT.IRQ**
3. If **MEINEXT.NOIRQ** is...
 - o Clear: Write one plus the priority of **MEINEXT.IRQ** to **MEICONTEXT.PREEMPT**
 - o Set: Write **0x10** to **MEICONTEXT.PREEMPT** (greater than any interrupt priority in **MEIPRA**)

MEICONTEXT.IRQ and **NOIRQ** help code determine in which interrupt handler it is running. **MEICONTEXT** should be saved/restored by interrupts which preempt the current one, so is safe to check these fields *during* the handler.

The update to **MEICONTEXT.PREEMPT** upon writing **MEINEXT.UPDATE** ensures the core will be preempted by interrupts higher-priority than the one it is about to enter. Equally important, it ensures the core is *not* preempted by lower or equal priority interrupts, including the one whose handler it is about to enter.

To avoid awkward interactions between the **MIP.MEIP** handler, which should be aware of the Xh3irq extension, and the MTIP/MSIP handlers, which may not be, it's best to avoid preemption of the former by the latter. **MEICONTEXT.CLEARSTS**, **MTIESAVE** and **MSIESAVE** support disabling and restoring the timer/software interrupt enables as part of the **MEICONTEXT** CSR accesses that take place during context save/restore in the MEIP handler.

3.8.4.1.6. Minimal Handler Example

This example demonstrates a minimal **meip** handler which dispatches to an array of C-function interrupt handlers, without enabling preemption. In this case the priorities configured in **MEIPRA** still determine the order in which interrupts are entered when multiple are asserted, but once an interrupt handler starts running, no other interrupts are serviced until that handler completes.

```
#include "hardware/regs/rvcsr.h"

isr_riscv_machine_external_irq:
    // Save all caller saves and temporaries before entering a C ABI function.
    // Note mstatus.mie is cleared by hardware on interrupt entry, and
    // we're going to leave it clear.
    addi sp, sp, -64
    sw ra, 0(sp)
    sw t0, 4(sp)
    sw t1, 8(sp)
    sw t2, 12(sp)
    sw a0, 16(sp)
    sw a1, 20(sp)
    sw a2, 24(sp)
    sw a3, 28(sp)
    sw a4, 32(sp)
    sw a5, 36(sp)
    sw a6, 40(sp)
    sw a7, 44(sp)
    sw t3, 48(sp)
    sw t4, 52(sp)
    sw t5, 56(sp)
    sw t6, 60(sp)

get_first_irq:
    // Sample the current highest-priority active IRQ (left-shifted by 2) from
    // meinext. Don't set the `update` bit as we aren't saving/restoring meicontext --
    // this is fine, just means you can't check meicontext to see whether you are in an IRQ.
    csrr a0, RVCSR_MEINEXT_OFFSET

    // MSB will be set if there is no active IRQ at the current priority level
    bltz a0, no_more_irqs
dispatch_irq:
    // Load indexed table entry and jump through it. No bounds checking is necessary
```

```

// because the hardware will not return a nonexistent IRQ.
lui a1, %hi(__soft_vector_table)
add a1, a1, a0
lw a1, %lo(__soft_vector_table)(a1)
jalr ra, a1
get_next_irq:
    // Get the next-highest-priority IRQ
    csrr a0, RVCSR_MEINEXT_OFFSET
    // MSB will be set if there is no active IRQ at the current priority level
    bgez a0, dispatch_irq

no_more_irqs:
    // Restore saved context and return from IRQ
    lw ra, 0(sp)
    lw t0, 4(sp)
    lw t1, 8(sp)
    lw t2, 12(sp)
    lw a0, 16(sp)
    lw a1, 20(sp)
    lw a2, 24(sp)
    lw a3, 28(sp)
    lw a4, 32(sp)
    lw a5, 36(sp)
    lw a6, 40(sp)
    lw a7, 44(sp)
    lw t3, 48(sp)
    lw t4, 52(sp)
    lw t5, 56(sp)
    lw t6, 60(sp)
    addi sp, sp, 64
    mret

// Array of function pointers for interrupt handlers
.section ".bss"
.p2align 2
.global __soft_vector_table
__soft_vector_table:
.space 52 * 4

```

Since the handler loops on `meinext` until no more interrupts are pending, multiple interrupts are processed with a single save/restore of the caller saves and temporaries.

The pending status of each IRQ in `MEIPA` clears once the corresponding peripheral deasserts its interrupt output. A correctly programmed interrupt handler should cause the peripheral interrupt to deassert, so each successive read from `meinext` will return a new interrupt. Because `meinext` always returns the highest-priority active interrupt, this loop iterates over active interrupts in descending priority order.

Note the overhead of performing the register save/restore in software is minimal, because the save/restore routine is actually limited by bus bandwidth, not by instruction execution overhead. This also makes the hardware more flexible, because the same hardware can support multiple interrupt ABIs.

3.8.4.2. Xh3pmpm: M-mode PMP regions

This extension adds a new M-mode CSR, `PMPCFGM0`, which allows a PMP region to be enforced in M-mode without locking the region.

This is useful when the PMP is used for non-security-related purposes such as stack guarding, or trapping and emulation of peripheral accesses.

3.8.4.3. Xh3power: Hazard3 power management

This extension adds a new M-mode CSR ([MSLEEP](#)), and two new hint instructions, `h3.block` and `h3.unblock`, in the `slt` nop-compatible custom hint space.

The `msleep` CSR controls how deeply the processor sleeps in the WFI sleep state. By default, a WFI is implemented as a normal pipeline stall. By configuring `msleep` appropriately, the processor can gate its own clock when asleep or, with a simple 4-phase req/ack handshake, negotiate power up/down of external hardware with an external power controller. These options can improve the sleep current at the cost of greater wakeup latency.

The hints allow processors to sleep until woken by other processors in a multiprocessor environment. They are implemented on top of the standard WFI state, which means they interact in the same way with external debug, and benefit from the same deep sleep states in `msleep`.

3.8.4.3.1. h3.block

Enter a WFI sleep state until either an unblock signal is received, or an interrupt is asserted that would cause a WFI to exit.

If `mstatus.tw` is set, attempting to execute this instruction in privilege modes lower than M-mode will generate an illegal instruction exception.

If an unblock signal has been received in the time since the last `h3.block`, this instruction executes as a `nop`, and the processor does not enter the sleep state. Conceptually, the sleep state falls through immediately because the corresponding unblock signal has already been received.

An unblock signal is received when a neighbouring processor (the exact definition of "neighbouring" being left to the implementer) executes an `h3.unblock` instruction, or for some other platform-defined reason.

This instruction is encoded as `slt x0, x0, x0`, which is part of the custom nop-compatible hint encoding space.

Example C macro:

```
#define __h3_block() asm ("slt x0, x0, x0")
```

Example assembly macro:

```
.macro h3.block
    slt x0, x0, x0
.endm
```

3.8.4.3.2. h3.unblock

Post an unblock signal to other processors in the system. For example, to notify another processor that a work queue is now nonempty.

If `mstatus.tw` is set, attempting to execute this instruction in privilege modes lower than M-mode will generate an illegal instruction exception.

This instruction is encoded as `slt x0, x0, x1`, which is part of the custom nop-compatible hint encoding space.

Example C macro:

```
#define __h3_unblock() asm ("slt x0, x0, x1")
```

Example assembly macro:

```
.macro h3.unblock
    slt x0, x0, x1
.endm
```

3.8.4.4. Xh3.bextm: Hazard3 bit extract multiple

This is a small extension with multi-bit versions of the "bit extract" instructions from Zbs, used for extracting small, contiguous bit fields.

3.8.4.4.1. h3.bextm

"Bit extract multiple", a multi-bit version of the `bext` instruction from Zbs. Perform a right-shift followed by a mask of 1-8 LSBs.

Encoding (R-type):

Bits	Name	Value	Description
31:29	<code>funct7[6:4]</code>	<code>0b000</code>	RES0
28:26	<code>size</code>	-	Number of ones in mask, values 0→7 encode 1→8 bits.
25	<code>funct7[0]</code>	<code>0b0</code>	RES0, because aligns with <code>shamt[5]</code> of potential RV64 version of <code>h3.bextm</code>
24:20	<code>rs2</code>	-	Source register 2 (shift amount)
19:15	<code>rs1</code>	-	Source register 1
14:12	<code>funct3</code>	<code>0b000</code>	<code>h3.bextm</code>
11:7	<code>rd</code>	-	Destination register
6:2	<code>opc</code>	<code>0b01011</code>	custom0 opcode
1:0	<code>size</code>	<code>0b11</code>	32-bit instruction

Example C macro (using GCC statement expressions):

```
// nbits must be a constant expression
#define _h3_bextm(nbts, rs1, rs2) ({\
    uint32_t __h3_bextm_rd; \
    asm (".insn r 0x0b, 0, %3, %0, %1, %2" \
        : "=r" (__h3_bextm_rd) \
        : "r" (rs1), "r" (rs2), "i" (((nbts) - 1) & 0x7) << 1) \
    ); \
    __h3_bextm_rd; \
})
```

Example assembly macro:

```
// rd = (rs1 >> rs2[4:0]) & ~(-1 << nbts)
.macro h3.bextm rd rs1 rs2 nbts
.if (\nbts < 1) || (\nbts > 8)
.err
```

```

.endif
#if NO_HAZARD3_CUSTOM
    srl \rd, \rs1, \rs2
    andi \rd, \rd, ((1 << \nbits) - 1)
#else
    .insn r 0x0b, 0x0, (((\nbits - 1) & 0x7) << 1), \rd, \rs1, \rs2
#endif
.endm

```

3.8.4.4.2. h3.bextmi

Immediate variant of [h3.bextm](#).

Encoding (I-type):

Bits	Name	Value	Description
31:29	imm[11:9]	0b000	RES0
28:26	size	-	Number of ones in mask, values 0→7 encode 1→8 bits.
25	imm[5]	0b0	RES0, for potential future RV64 version
24:20	shamt	-	Shift amount, 0 through 31
19:15	rs1	-	Source register 1
14:12	funct3	0b100	h3.bextmi
11:7	rd	-	Destination register
6:2	opc	0b01011	custom0 opcode
1:0	size	0b11	32-bit instruction

Example C macro (using GCC statement expressions):

```

// nbits and shamt must be constant expressions
#define __h3_bextmi(nbits, rs1, shamt) ({\
    uint32_t __h3_bextmi_rd; \
    asm (".insn i 0x0b, 0x4, %0, %1, %2" \
        : "=r" (__h3_bextmi_rd) \
        : "r" (rs1), "i" (((nbits) - 1) & 0x7) << 6 | ((shamt) & 0x1f)) \
    ); \
    __h3_bextmi_rd; \
})

```

Example assembly macro:

```

// rd = (rs1 >> shamt) & ~(-1 << nbits)
.macro h3.bextmi rd rs1 shamt nbits
.if (\nbits < 1) || (\nbits > 8)
.err
.endif
.if (\shamt < 0) || (\shamt > 31)
.err
.endif
#if NO_HAZARD3_CUSTOM
    srl \rd, \rs1, \shamt
    andi \rd, \rd, ((1 << \nbits) - 1)

```

```

#else
.insn i 0x0b, 0x4, \rd, \rs1, (\shamt & 0x1f) | (((\nbits - 1) & 0x7) << 6)
#endif
.endm

```

3.8.5. Instruction Cycle Counts

All timings are given assuming perfect bus behaviour (no downstream bus stalls).

3.8.5.1. RV32I

Instruction	Cycles	Note
Integer Register-register		
add rd, rs1, rs2	1	
sub rd, rs1, rs2	1	
slt rd, rs1, rs2	1	
sltu rd, rs1, rs2	1	
and rd, rs1, rs2	1	
or rd, rs1, rs2	1	
xor rd, rs1, rs2	1	
sll rd, rs1, rs2	1	
srl rd, rs1, rs2	1	
sra rd, rs1, rs2	1	
Integer Register-immediate		
addi rd, rs1, imm	1	nop is a pseudo-op for addi x0, x0, 0
slti rd, rs1, imm	1	
sltiu rd, rs1, imm	1	
andi rd, rs1, imm	1	
ori rd, rs1, imm	1	
xori rd, rs1, imm	1	
slli rd, rs1, imm	1	
srlti rd, rs1, imm	1	
srai rd, rs1, imm	1	
Large Immediate		
lui rd, imm	1	
auipc rd, imm	1	
Control Transfer		
jal rd, label	$2^{[1]}$	
jalr rd, rs1, imm	$2^{[1]}$	

Instruction	Cycles	Note
<code>beq rs1, rs2, label</code>	1 or 2 ^[1]	1 if correctly predicted, 2 if mispredicted.
<code>bne rs1, rs2, label</code>	1 or 2 ^[1]	1 if correctly predicted, 2 if mispredicted.
<code>blt rs1, rs2, label</code>	1 or 2 ^[1]	1 if correctly predicted, 2 if mispredicted.
<code>bge rs1, rs2, label</code>	1 or 2 ^[1]	1 if correctly predicted, 2 if mispredicted.
<code>bltu rs1, rs2, label</code>	1 or 2 ^[1]	1 if correctly predicted, 2 if mispredicted.
<code>bgeu rs1, rs2, label</code>	1 or 2 ^[1]	1 if correctly predicted, 2 if mispredicted.
Load and Store		
<code>lw rd, imm(rs1)</code>	1 or 2	1 if next instruction is independent, 2 if dependent. ^[2]
<code>lh rd, imm(rs1)</code>	1 or 2	1 if next instruction is independent, 2 if dependent. ^[2]
<code>lhu rd, imm(rs1)</code>	1 or 2	1 if next instruction is independent, 2 if dependent. ^[2]
<code>lb rd, imm(rs1)</code>	1 or 2	1 if next instruction is independent, 2 if dependent. ^[2]
<code>lbu rd, imm(rs1)</code>	1 or 2	1 if next instruction is independent, 2 if dependent. ^[2]
<code>sw rs2, imm(rs1)</code>	1	
<code>sh rs2, imm(rs1)</code>	1	
<code>sb rs2, imm(rs1)</code>	1	

3.8.5.2. M Extension

Instruction	Cycles	Note
32 × 32 → 32 Multiply		
<code>mul rd, rs1, rs2</code>	1	
32 × 32 → 64 Multiply, Upper Half		
<code>mulh rd, rs1, rs2</code>	1	
<code>mulhsu rd, rs1, rs2</code>	1	
<code>mulhu rd, rs1, rs2</code>	1	
Divide and Remainder		
<code>div rd, rs1, rs2</code>	18 or 19	Depending on sign correction
<code>divu rd, rs1, rs2</code>	18	
<code>rem rd, rs1, rs2</code>	18 or 19	Depending on sign correction
<code>remu rd, rs1, rs2</code>	18	

3.8.5.3. A Extension

Instruction	Cycles	Note
Load-Reserved/Store-Conditional		
<code>lr.w rd, (rs1)</code>	1 or 2	2 if next instruction is dependent ^[2] , an <code>lr.w, sc.w</code> or <code>amo*.w</code> . ^[3]
<code>sc.w rd, rs2, (rs1)</code>	1 or 2	2 if next instruction is dependent ^[2] , an <code>lr.w, sc.w</code> or <code>amo*.w</code> . ^[3]

Instruction	Cycles	Note
Atomic Memory Operations		
<code>amoswap.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amoadd.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amoxor.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amoand.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amoor.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amomin.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amamax.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amominu.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]
<code>amamaxu.w rd, rs2, (rs1)</code>	4+	4 per attempt. Multiple attempts if reservation is lost. ^[4]

3.8.5.4. C Extension

All C extension 16-bit instructions are aliases of base RV32I instructions. On Hazard3, they perform identically to their 32-bit counterparts.

A consequence of the C extension is that 32-bit instructions can be non-naturally-aligned. This has no penalty during sequential execution, but branching to a 32-bit instruction that is not 32-bit-aligned carries a 1 cycle penalty, because the instruction fetch is cracked into two naturally-aligned bus accesses.

3.8.5.5. Privileged Instructions (including Zicsr)

Instruction	Cycles	Note
CSR Access		
<code>csrrw rd, csr, rs1</code>	1	
<code>csrrc rd, csr, rs1</code>	1	
<code>csrrs rd, csr, rs1</code>	1	
<code>csrrwi rd, csr, imm</code>	1	
<code>csrrci rd, csr, imm</code>	1	
<code>csrrsi rd, csr, imm</code>	1	
Trap Request		
<code>ecall</code>	3	Time given is for jumping to <code>mtvec</code>
<code>ebreak</code>	3	Time given is for jumping to <code>mtvec</code>

3.8.5.6. Bit Manipulation

Instruction	Cycles	Note
Zba (address generation)		
<code>sh1add rd, rs1, rs2</code>	1	

Instruction	Cycles	Note
sh2add rd, rs1, rs2	1	
sh3add rd, rs1, rs2	1	
Zbb (basic bit manipulation)		
andn rd, rs1, rs2	1	
clz rd, rs1	1	
cpop rd, rs1	1	
ctz rd, rs1	1	
max rd, rs1, rs2	1	
maxu rd, rs1, rs2	1	
min rd, rs1, rs2	1	
minu rd, rs1, rs2	1	
orc.b rd, rs1	1	
orn rd, rs1, rs2	1	
rev8 rd, rs1	1	
rol rd, rs1, rs2	1	
ror rd, rs1, rs2	1	
rori rd, rs1, imm	1	
sext.b rd, rs1	1	
sext.h rd, rs1	1	
xnor rd, rs1, rs2	1	
zext.h rd, rs1	1	
zext.b rd, rs1	1	zext.b is a pseudo-op for andi rd, rs1, 0xff
Zbs (single-bit manipulation)		
bclr rd, rs1, rs2	1	
bclri rd, rs1, imm	1	
bext rd, rs1, rs2	1	
bexti rd, rs1, imm	1	
binv rd, rs1, rs2	1	
binvi rd, rs1, imm	1	
bset rd, rs1, rs2	1	
bseti rd, rs1, imm	1	
Zbkb (basic bit manipulation for cryptography)		
pack rd, rs1, rs2	1	
packh rd, rs1, rs2	1	
brev8 rd, rs1	1	
zip rd, rs1	1	

Instruction	Cycles	Note
unzip rd, rs1	1	

3.8.5.7. Zcb Extension

Similarly to the C extension, this extension contains 16-bit variants of common 32-bit instructions:

- RV32I base ISA: `lbu`, `lh`, `lhu`, `sb`, `sh`, `zext.b` (alias of `andi`), `not` (alias of `xori`)
- Zbb extension: `sext.b`, `zext.h`, `sext.h`
- M extension: `mul`

They perform identically to their 32-bit counterparts.

3.8.5.8. Zcmp Extension

Instruction	Cycles	Note
<code>cm.push rlist, -imm</code>	$1 + n$	n is number of registers in <code>rlist</code>
<code>cm.pop rlist, imm</code>	$1 + n$	n is number of registers in <code>rlist</code>
<code>cm.popret rlist, imm</code>	$4 (n = 1)^{[5]} \text{ or } 2 + n (n \geq 2)^{[1]}$	n is number of registers in <code>rlist</code>
<code>cm.popretz rlist, imm</code>	$5 (n = 1)^{[5]} \text{ or } 3 + n (n \geq 2)^{[1]}$	n is number of registers in <code>rlist</code>
<code>cm.mva01s r1s', r2s'</code>	2	
<code>cm.mvs01 r1s', r2s'</code>	2	

3.8.5.9. Branch Predictor

Hazard3 includes a minimal branch predictor, to accelerate tight loops:

- The instruction frontend remembers the last taken, backward branch
- If the same branch is seen again, it is predicted taken
- All other branches are predicted nontaken
- If a predicted-taken branch is not taken, the predictor state is cleared, and it will be predicted nontaken on its next execution.

Correctly predicted branches execute in one cycle: the frontend is able to stitch together the two nonsequential fetch paths so that they appear sequential. Mispredicted branches incur a penalty cycle, since a nonsequential fetch address must be issued when the branch is executed.

3.8.5.10. Table Footnotes

- [1] A jump or branch to a 32-bit instruction which is not 32-bit-aligned requires one additional cycle, because two naturally aligned bus cycles are required to fetch the target instruction.
- [2] If an instruction in stage 2 (e.g. an `add`) uses data from stage 3 (e.g. a `lw` result), a 1-cycle bubble is inserted between the pair. A load data → store data dependency is *not* an example of this, because data is produced and consumed in stage 3. However, load data → load address *would* qualify, as would e.g. `sc.w` → `beqz`.

- [3] AMOs are issued as a paired exclusive read and exclusive write on the bus, at the maximum speed of 2 cycles per access, since the bus does not permit pipelining of exclusive reads/writes. If the write phase fails due to the global monitor reporting a lost reservation, the instruction loops at a rate of 4 cycles per loop, until success. If the read reservation is refused by the global monitor, the instruction generates a Store/AMO Fault exception, to avoid an infinite loop.
- [4] A pipeline bubble is inserted between `lr.w/sc.w` and an immediately-following `lr.w/sc.w/amo*`, because the AHB5 bus standard does not permit pipelined exclusive accesses. A stall would be inserted between `lr.w` and `sc.w` anyhow, so the local monitor can be updated based on the `lr.w` data phase in time to suppress the `sc.w` address phase.
- [5] The single-register variants of `cm.popret` and `cm.popretz` take the same number of cycles as the two-register variants, because of an internal load-use dependency on the loaded return address.

3.8.6. Configuration

Hazard3 uses the parameters given in the `hazard3_config.vh` header to customise the core. These values are set before taping out a Hazard3 instance on silicon, so they are *fixed* from a user point of view. They determine which instructions the processor supports, the area-performance tradeoff for certain instructions, and static configuration for core peripherals like the PMP. RP2350 uses the following values for these parameters:

Parameter	Value
<code>EXTENSION_A</code>	1
<code>EXTENSION_C</code>	1
<code>EXTENSION_M</code>	1
<code>EXTENSION_ZBA</code>	1
<code>EXTENSION_ZBB</code>	1
<code>EXTENSION_ZBC</code>	0
<code>EXTENSION_ZBS</code>	1
<code>EXTENSION_ZCB</code>	1
<code>EXTENSION_ZCMP</code>	1
<code>EXTENSION_ZBKB</code>	1
<code>EXTENSION_ZIFENCEI</code>	1
<code>EXTENSION_XH3BEXTM</code>	1
<code>EXTENSION_XH3IRQ</code>	1
<code>EXTENSION_XH3PPPM</code>	1
<code>EXTENSION_XH3POWER</code>	1
<code>CSR_M_MANDATORY</code>	1
<code>CSR_M_TRAP</code>	1
<code>CSR_COUNTER</code>	1
<code>U_MODE</code>	1
<code>PMP_REGIONS</code>	11
<code>PMP_GRAIN</code>	3
<code>PMP_HARDWIRED</code>	<code>11'h700</code>
<code>PMP_HARDWIRED_ADDR</code>	See Section 3.8.6.1

Parameter	Value
PMP_HARDWIRED_CFG	See Section 3.8.6.1
DEBUG_SUPPORT	1
BREAKPOINT_TRIGGER	4
NUM_IRQS	52
IRQ_PRIORITY_BITS	4
IRQ_INPUT_BYPASS	{NUM_IRQS{1'b1}}
MVENDORID_VAL	32'h000000493
MIMPID_VAL	32'h86fc4e3f
MCONFIGPTR_VAL	32'h0
REDUCED_BYPASS	0
MULDIV_UNROLL	2
MUL_FAST	1
MUL_FASTER	1
MULH_FAST	1
FAST_BRANCHCMP	1
RESET_REGFILE	1
BRANCH_PREDICTOR	1
MTVEC_WMASK	32'hfffffffffd

3.8.6.1. Hardwired PMP Regions

RP2350 configures Hazard3 with eight dynamically configured PMP regions, and three static ones. The static regions provide default U-mode RWX permissions on the following ranges:

- ROM: `0x00000000` through `0xffffffff`
- Peripherals: `0x40000000` through `0x5fffffff`
- SIO: `0xd0000000` through `0xdfffffff`

These addresses appear in [PMPADDR8](#), [PMPADDR9](#) and [PMPADDR10](#). The hardwired PMP address registers behave the same as dynamic registers, except that they ignore writes (exercising the WARL rule). The permissions for these regions are in [PMPCFG2](#).

The hardwired regions have a similar role to the Exempt regions added to the Cortex-M33 IDAU address map specified in [Section 10.2.2](#).

RP2350 puts default U-mode permissions on AHB/APB peripherals because these are expected to be assigned using ACCESSCTRL ([Section 10.5](#)). ACCESSCTRL can assign each peripheral individually, using the existing address decoders in the busfabric, whereas PMP regions are in limited supply so are less useful for peripheral assignment.

Similarly, SIO has internal banking over Secure/Non-secure bus attribution, which is mapped onto Machine and User modes as described in [Section 10.5.2](#).

The dynamic regions 0 through 7 take priority over the hardwired regions, because the PMP prioritises lower-numbered regions.

3.8.7. Control and Status Registers

Control and status registers (CSRs) are control registers internal to the processor, accessed with dedicated CSR instructions. They are not accessible through the system bus.

The RISC-V privileged specification affords flexibility as to which CSRs are implemented, and how they behave. This section documents the concrete behaviour of Hazard3's standard and nonstandard M-mode CSRs, as implemented.

All CSRs are 32-bit; MXLEN is fixed at 32 bits on Hazard3. All CSR addresses not listed in this section are unimplemented. Accessing an unimplemented CSR will cause an illegal instruction exception (`mcause` = 2). This includes all S-mode CSRs.

! IMPORTANT

The [RISC-V Privileged Specification](#) should be your primary reference for writing software to run on Hazard3. This section specifies those details which are left implementation-defined by the RISC-V Privileged Specification, for sake of completeness, but portable RISC-V software should not rely on these details.

Table 360. List of RVCSR registers

Offset	Name	Info
0x300	MSTATUS	Machine status register
0x301	MISA	Summary of ISA extension support
0x302	MEDELEG	Machine exception delegation register. Not implemented, as no S-mode support.
0x303	MIDELEG	Machine interrupt delegation register. Not implemented, as no S-mode support.
0x304	MIE	Machine interrupt enable register
0x305	MTVEC	Machine trap handler base address.
0x306	MCOUNTEREN	Counter enable. Control access to counters from U-mode. Not to be confused with <code>mcountinhibit</code> .
0x30a	MENVCFG	Machine environment configuration register, low half
0x310	MSTATUSH	High half of <code>mstatus</code> , hardwired to 0.
0x31a	MENVCFGH	Machine environment configuration register, high half
0x320	MCOUNTINHIBIT	Count inhibit register for <code>mcycle/minstret</code>
0x323	MHPMEVENT3	Extended performance event selector, hardwired to 0.
0x324	MHPMEVENT4	Extended performance event selector, hardwired to 0.
0x325	MHPMEVENT5	Extended performance event selector, hardwired to 0.
0x326	MHPMEVENT6	Extended performance event selector, hardwired to 0.
0x327	MHPMEVENT7	Extended performance event selector, hardwired to 0.
0x328	MHPMEVENT8	Extended performance event selector, hardwired to 0.
0x329	MHPMEVENT9	Extended performance event selector, hardwired to 0.
0x32a	MHPMEVENT10	Extended performance event selector, hardwired to 0.
0x32b	MHPMEVENT11	Extended performance event selector, hardwired to 0.
0x32c	MHPMEVENT12	Extended performance event selector, hardwired to 0.
0x32d	MHPMEVENT13	Extended performance event selector, hardwired to 0.
0x32e	MHPMEVENT14	Extended performance event selector, hardwired to 0.

Offset	Name	Info
0x32f	MHPMEVENT15	Extended performance event selector, hardwired to 0.
0x330	MHPMEVENT16	Extended performance event selector, hardwired to 0.
0x331	MHPMEVENT17	Extended performance event selector, hardwired to 0.
0x332	MHPMEVENT18	Extended performance event selector, hardwired to 0.
0x333	MHPMEVENT19	Extended performance event selector, hardwired to 0.
0x334	MHPMEVENT20	Extended performance event selector, hardwired to 0.
0x335	MHPMEVENT21	Extended performance event selector, hardwired to 0.
0x336	MHPMEVENT22	Extended performance event selector, hardwired to 0.
0x337	MHPMEVENT23	Extended performance event selector, hardwired to 0.
0x338	MHPMEVENT24	Extended performance event selector, hardwired to 0.
0x339	MHPMEVENT25	Extended performance event selector, hardwired to 0.
0x33a	MHPMEVENT26	Extended performance event selector, hardwired to 0.
0x33b	MHPMEVENT27	Extended performance event selector, hardwired to 0.
0x33c	MHPMEVENT28	Extended performance event selector, hardwired to 0.
0x33d	MHPMEVENT29	Extended performance event selector, hardwired to 0.
0x33e	MHPMEVENT30	Extended performance event selector, hardwired to 0.
0x33f	MHPMEVENT31	Extended performance event selector, hardwired to 0.
0x340	MSCRATCH	Scratch register for machine trap handlers
0x341	MEPC	Machine exception program counter
0x342	MCAUSE	Machine trap cause. Set when entering a trap to indicate the reason for the trap. Readable and writable by software.
0x343	MTVAL	Machine bad address or instruction. Hardwired to zero.
0x344	MIP	Machine interrupt pending
0x3a0	PMPCFG0	Physical memory protection configuration for regions 0 through 3
0x3a1	PMPCFG1	Physical memory protection configuration for regions 4 through 7
0x3a2	PMPCFG2	Physical memory protection configuration for regions 8 through 11
0x3a3	PMPCFG3	Physical memory protection configuration for regions 12 through 15
0x3b0	PMPADDR0	Physical memory protection address for region 0
0x3b1	PMPADDR1	Physical memory protection address for region 1
0x3b2	PMPADDR2	Physical memory protection address for region 2
0x3b3	PMPADDR3	Physical memory protection address for region 3
0x3b4	PMPADDR4	Physical memory protection address for region 4
0x3b5	PMPADDR5	Physical memory protection address for region 5

Offset	Name	Info
0x3b6	PMPADDR6	Physical memory protection address for region 6
0x3b7	PMPADDR7	Physical memory protection address for region 7
0x3b8	PMPADDR8	Physical memory protection address for region 8
0x3b9	PMPADDR9	Physical memory protection address for region 9
0x3ba	PMPADDR10	Physical memory protection address for region 10
0x3bb	PMPADDR11	Physical memory protection address for region 11
0x3bc	PMPADDR12	Physical memory protection address for region 12
0x3bd	PMPADDR13	Physical memory protection address for region 13
0x3be	PMPADDR14	Physical memory protection address for region 14
0x3bf	PMPADDR15	Physical memory protection address for region 15
0x7a0	TSELECT	Select trigger to be configured via tdata1/tdata2
0x7a1	TDATA1	Trigger configuration data 1
0x7a2	TDATA2	Trigger configuration data 2
0x7b0	DCSR	Debug control and status register (Debug Mode only)
0x7b1	DPC	Debug program counter (Debug Mode only)
0xb00	MCYCLE	Machine-mode cycle counter, low half
0xb02	MINSTRET	Machine-mode instruction retire counter, low half
0xb03	MHPMCOUNTER3	Extended performance counter, hardwired to 0.
0xb04	MHPMCOUNTER4	Extended performance counter, hardwired to 0.
0xb05	MHPMCOUNTER5	Extended performance counter, hardwired to 0.
0xb06	MHPMCOUNTER6	Extended performance counter, hardwired to 0.
0xb07	MHPMCOUNTER7	Extended performance counter, hardwired to 0.
0xb08	MHPMCOUNTER8	Extended performance counter, hardwired to 0.
0xb09	MHPMCOUNTER9	Extended performance counter, hardwired to 0.
0xb0a	MHPMCOUNTER10	Extended performance counter, hardwired to 0.
0xb0b	MHPMCOUNTER11	Extended performance counter, hardwired to 0.
0xb0c	MHPMCOUNTER12	Extended performance counter, hardwired to 0.
0xb0d	MHPMCOUNTER13	Extended performance counter, hardwired to 0.
0xb0e	MHPMCOUNTER14	Extended performance counter, hardwired to 0.
0xb0f	MHPMCOUNTER15	Extended performance counter, hardwired to 0.
0xb10	MHPMCOUNTER16	Extended performance counter, hardwired to 0.
0xb11	MHPMCOUNTER17	Extended performance counter, hardwired to 0.
0xb12	MHPMCOUNTER18	Extended performance counter, hardwired to 0.
0xb13	MHPMCOUNTER19	Extended performance counter, hardwired to 0.
0xb14	MHPMCOUNTER20	Extended performance counter, hardwired to 0.
0xb15	MHPMCOUNTER21	Extended performance counter, hardwired to 0.

Offset	Name	Info
0xb16	MHPMCOUNTER22	Extended performance counter, hardwired to 0.
0xb17	MHPMCOUNTER23	Extended performance counter, hardwired to 0.
0xb18	MHPMCOUNTER24	Extended performance counter, hardwired to 0.
0xb19	MHPMCOUNTER25	Extended performance counter, hardwired to 0.
0xb1a	MHPMCOUNTER26	Extended performance counter, hardwired to 0.
0xb1b	MHPMCOUNTER27	Extended performance counter, hardwired to 0.
0xb1c	MHPMCOUNTER28	Extended performance counter, hardwired to 0.
0xb1d	MHPMCOUNTER29	Extended performance counter, hardwired to 0.
0xb1e	MHPMCOUNTER30	Extended performance counter, hardwired to 0.
0xb1f	MHPMCOUNTER31	Extended performance counter, hardwired to 0.
0xb80	MCYCLEH	Machine-mode cycle counter, high half
0xb82	MINSTRETH	Machine-mode instruction retire counter, low half
0xb83	MHPMCOUNTER3H	Extended performance counter, hardwired to 0.
0xb84	MHPMCOUNTER4H	Extended performance counter, hardwired to 0.
0xb85	MHPMCOUNTER5H	Extended performance counter, hardwired to 0.
0xb86	MHPMCOUNTER6H	Extended performance counter, hardwired to 0.
0xb87	MHPMCOUNTER7H	Extended performance counter, hardwired to 0.
0xb88	MHPMCOUNTER8H	Extended performance counter, hardwired to 0.
0xb89	MHPMCOUNTER9H	Extended performance counter, hardwired to 0.
0xb8a	MHPMCOUNTER10H	Extended performance counter, hardwired to 0.
0xb8b	MHPMCOUNTER11H	Extended performance counter, hardwired to 0.
0xb8c	MHPMCOUNTER12H	Extended performance counter, hardwired to 0.
0xb8d	MHPMCOUNTER13H	Extended performance counter, hardwired to 0.
0xb8e	MHPMCOUNTER14H	Extended performance counter, hardwired to 0.
0xb8f	MHPMCOUNTER15H	Extended performance counter, hardwired to 0.
0xb90	MHPMCOUNTER16H	Extended performance counter, hardwired to 0.
0xb91	MHPMCOUNTER17H	Extended performance counter, hardwired to 0.
0xb92	MHPMCOUNTER18H	Extended performance counter, hardwired to 0.
0xb93	MHPMCOUNTER19H	Extended performance counter, hardwired to 0.
0xb94	MHPMCOUNTER20H	Extended performance counter, hardwired to 0.
0xb95	MHPMCOUNTER21H	Extended performance counter, hardwired to 0.
0xb96	MHPMCOUNTER22H	Extended performance counter, hardwired to 0.
0xb97	MHPMCOUNTER23H	Extended performance counter, hardwired to 0.
0xb98	MHPMCOUNTER24H	Extended performance counter, hardwired to 0.
0xb99	MHPMCOUNTER25H	Extended performance counter, hardwired to 0.
0xb9a	MHPMCOUNTER26H	Extended performance counter, hardwired to 0.

Offset	Name	Info
0xb9b	MHPMCOUNTER27H	Extended performance counter, hardwired to 0.
0xb9c	MHPMCOUNTER28H	Extended performance counter, hardwired to 0.
0xb9d	MHPMCOUNTER29H	Extended performance counter, hardwired to 0.
0xb9e	MHPMCOUNTER30H	Extended performance counter, hardwired to 0.
0xb9f	MHPMCOUNTER31H	Extended performance counter, hardwired to 0.
0xbd0	PMPCFGM0	Set PMP regions to M-mode, without locking
0xbe0	MEIEA	External interrupt enable array
0xbe1	MEIPA	External interrupt pending array
0xbe2	MEIFA	External interrupt force array
0xbe3	MEIPRA	External interrupt priority array
0xbe4	MEINEXT	Get next external interrupt
0xbe5	MEICONTEXT	External interrupt context register
0xbf0	MSLEEP	M-mode sleep control register
0xbff	DMDATA0	Debug Module DATA0 access register (Debug Mode only)
0xc00	CYCLE	Read-only U-mode alias of mcycle, accessible when <code>mcounteren.cy</code> is set
0xc02	INSTRET	Read-only U-mode alias of minstret, accessible when <code>mcounteren.ir</code> is set
0xc80	CYCLEH	Read-only U-mode alias of mcycleh, accessible when <code>mcounteren.cy</code> is set
0xc82	INSTRETH	Read-only U-mode alias of minstreth, accessible when <code>mcounteren.ir</code> is set
0xf11	MVENDORID	Vendor ID
0xf12	MARCHID	Architecture ID (Hazard3)
0xf13	MIMPID	Implementation ID
0xf14	MHARTID	Hardware thread ID
0xf15	MCONFIGPTR	Pointer to configuration data structure (hardwired to 0)

RVCSR: MSTATUS Register

Offset: 0x300

Description

Machine status register

Table 361. MSTATUS Register

Bits	Name	Description	Type	Reset
31:22	Reserved.	-	-	-
21	TW	Timeout wait. When 1, attempting to execute a WFI instruction in U-mode will instantly cause an illegal instruction exception.	RW	0x0
20:18	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
17	MPRV	Modify privilege. If 1, loads and stores behave as though the current privilege level were <code>mpp</code> . This includes physical memory protection checks, and the privilege level asserted on the system bus alongside the load/store address.	RW	0x0
16:13	Reserved.	-	-	-
12:11	MPP	Previous privilege level. Can store the values 3 (M-mode) or 0 (U-mode). If another value is written, hardware rounds to the nearest supported mode.	RW	0x3
10:8	Reserved.	-	-	-
7	MPIE	Previous interrupt enable. Readable and writable. Is set to the current value of <code>mstatus.mie</code> on trap entry. Is set to 1 on trap return.	RW	0x0
6:4	Reserved.	-	-	-
3	MIE	Interrupt enable. Readable and writable. Is set to 0 on trap entry. Is set to the current value of <code>mstatus.mpie</code> on trap return.	RW	0x0
2:0	Reserved.	-	-	-

RVCSR: MISA Register

Offset: 0x301

Description

Summary of ISA extension support

On RP2350, Hazard3's full `-march` string is: `rv32ima_zicsr_zifencei_zba_zbb_zbs_zbkb_zca_zcb_zcmp`

Note Zca is equivalent to the C extension in this case; all instructions from the RISC-V C extension relevant to a 32-bit non-floating-point processor are supported. On older toolchains which do not support the Zc extensions, the appropriate `-march` string is: `rv32imac_zicsr_zifencei_zba_zbb_zbs_zbkb`

In addition the following custom extensions are configured: Xh3bm, Xh3power, Xh3irq, Xh3pmpm

Table 362. MISA Register

Bits	Name	Description	Type	Reset
31:30	MXL	Value of 0x1 indicates this is a 32-bit processor.	RO	0x1
29:24	Reserved.	-	-	-
23	X	Value of 1 indicates nonstandard extensions are present. (Xh3b bit manipulation, and custom sleep and interrupt control CSRs)	RO	0x1
22:21	Reserved.	-	-	-
20	U	Value of 1 indicates U-mode is implemented.	RO	0x1
19:13	Reserved.	-	-	-
12	M	Value of 1 indicates the M extension (integer multiply/divide) is implemented.	RO	0x1
11:9	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
8	I	Value of 1 indicates the RVI base ISA is implemented (as opposed to RVE)	RO	0x1
7:3	Reserved.	-	-	-
2	C	Value of 1 indicates the C extension (compressed instructions) is implemented.	RO	0x1
1	Reserved.	-	-	-
0	A	Value of 1 indicates the A extension (atomics) is implemented.	RO	0x1

RVCSR: MEDELEG Register

Offset: 0x302

Table 363. MEDELEG Register

Bits	Description	Type	Reset
31:0	Machine exception delegation register. Not implemented, as no S-mode support.	RW	-

RVCSR: MIDELEG Register

Offset: 0x303

Table 364. MIDELEG Register

Bits	Description	Type	Reset
31:0	Machine interrupt delegation register. Not implemented, as no S-mode support.	RW	-

RVCSR: MIE Register

Offset: 0x304

Description

Machine interrupt enable register

Table 365. MIE Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	MEIE	External interrupt enable. The processor transfers to the external interrupt vector when <code>mie.meie</code> , <code>mip.meip</code> and <code>mstatus.mie</code> are all 1. Hazard3 has internal registers to individually filter external interrupts (see <code>meie</code>), but this standard control can be used to mask all external interrupts at once.	RW	0x0
10:8	Reserved.	-	-	-
7	MTIE	Timer interrupt enable. The processor transfers to the timer interrupt vector when <code>mie.mtie</code> , <code>mip.mtip</code> and <code>mstatus.mie</code> are all 1, unless a software or external interrupt request is also valid at this time.	RW	0x0
6:4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3	MSIE	Software interrupt enable. The processor transfers to the software interrupt vector <code>mie.msie</code> , <code>mip.msip</code> and <code>mstatus.mie</code> are all 1, unless an external interrupt request is also valid at this time.	RW	0x0
2:0	Reserved.	-	-	-

RVCSR: MTVEC Register

Offset: 0x305

Description

Machine trap handler base address.

Table 366. MTVEC Register

Bits	Name	Description	Type	Reset
31:2	BASE	The upper 30 bits of the trap vector address (2 LSBs are implicitly 0). Must be 64-byte-aligned if vectoring is enabled. Otherwise, must be 4-byte-aligned.	RW	0x00001fff
1:0	MODE	If 0 (direct mode), all traps set pc to the trap vector base. If 1 (vectored), exceptions set pc to the trap vector base, and interrupts set pc to 4 times the interrupt cause (3=soft IRQ, 7=timer IRQ, 11=external IRQ). The upper bit is hardwired to zero, so attempting to set mode to 2 or 3 will result in a value of 0 or 1 respectively. 0x0 → Direct entry to mtvec 0x1 → Vectored entry to a 16-entry jump table starting at mtvec	RW	0x0

RVCSR: MCOUNTEREN Register

Offset: 0x306

Description

Counter enable. Control access to counters from U-mode. Not to be confused with mcountinhibit.

Table 367.
MCOUNTEREN Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	IR	If 1, U-mode is permitted to access the <code>instret/instreth</code> instruction retire counter CSRs. Otherwise, U-mode accesses to these CSRs will trap.	RW	0x0
1	TM	No hardware effect, as the <code>time/timeh</code> CSRs are not implemented. However, this field still exists, as M-mode software can use it to track whether it should emulate U-mode attempts to access those CSRs.	RW	0x0
0	CY	If 1, U-mode is permitted to access the <code>cycle/cycleh</code> cycle counter CSRs. Otherwise, U-mode accesses to these CSRs will trap.	RW	0x0

RVCSR: MENVCFG Register

Offset: 0x30a

Description

Machine environment configuration register, low half

Table 368. MENVCFG Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	FIOM	<p>When set, fence instructions in modes less privileged than M-mode which specify that IO memory accesses are ordered will also cause ordering of main memory accesses.</p> <p>FIOM is hardwired to zero on Hazard3, because S-mode is not supported, and because fence instructions execute as NOPs (with the exception of <code>fence.i</code>)</p>	RO	0x0

RVCSR: MSTATUSH Register

Offset: 0x310

Table 369. MSTATUSH Register

Bits	Description	Type	Reset
31:0	High half of mstatus, hardwired to 0.	RO	0x00000000

RVCSR: MENVCFGH Register

Offset: 0x31a

Description

Machine environment configuration register, high half

This register is fully reserved, as Hazard3 does not implement the relevant extensions. It is implemented as hardwired-0.

Table 370. MENVCFGH Register

Bits	Name	Description	Type	Reset
31:0	Reserved.	-	-	-

RVCSR: MCOUNTINHIBIT Register

Offset: 0x320

Description

Count inhibit register for `mcycle/minstret`

Table 371. MCOUNTINHIBIT Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	IR	Inhibit counting of the <code>minstret</code> and <code>minstreth</code> registers. Set by default to save power.	RW	0x1
1	Reserved.	-	-	-
0	CY	Inhibit counting of the <code>mcycle</code> and <code>mcycleh</code> registers. Set by default to save power.	RW	0x1

RVCSR: MHPMEVENT3, MHPMEVENT4, ..., MHPMEVENT30, MHPMEVENT31 Registers

Offsets: 0x323, 0x324, ..., 0x33e, 0x33f

Table 372.
MHPMEVENT3,
MHPMEVENT4, ...,
MHPMEVENT30,
MHPMEVENT31
Registers

Bits	Description	Type	Reset
31:0	Extended performance event selector, hardwired to 0.	RO	0x00000000

RVCSR: MSCRATCH Register

Offset: 0x340

Table 373.
MSCRATCH Register

Bits	Description	Type	Reset
31:0	Scratch register for machine trap handlers. 32-bit read/write register with no specific hardware function. Software may use this to do a fast save/restore of a core register in a trap handler.	RW	0x00000000

RVCSR: MEPC Register

Offset: 0x341

Table 374. MEPC Register

Bits	Description	Type	Reset
31:2	Machine exception program counter. When entering a trap, the current value of the program counter is recorded here. When executing an <code>mret</code> , the processor jumps to <code>mepc</code> . Can also be read and written by software.	RW	0x00000000
1:0	Reserved.	-	-

RVCSR: MCAUSE Register

Offset: 0x342

Description

Machine trap cause. Set when entering a trap to indicate the reason for the trap. Readable and writable by software.

Table 375. MCAUSE Register

Bits	Name	Description	Type	Reset
31	INTERRUPT	If 1, the trap was caused by an interrupt. If 0, it was caused by an exception.	RW	0x0
30:4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3:0	CODE	If <code>interrupt</code> is set, <code>code</code> indicates the index of the bit in <code>mip</code> that caused the trap (3=soft IRQ, 7=timer IRQ, 11=external IRQ). Otherwise, <code>code</code> is set according to the cause of the exception. 0x0 → Instruction fetch was misaligned. Will never fire on RP2350, since the C extension is enabled. 0x1 → Instruction access fault. Instruction fetch failed a PMP check, or encountered a downstream bus fault, and then passed the point of no speculation. 0x2 → Illegal instruction was executed (including illegal CSR accesses) 0x3 → Breakpoint. An ebreak instruction was executed when the relevant <code>dcsr.ebreak</code> bit was clear. 0x4 → Load address misaligned. Hazard3 requires natural alignment of all accesses. 0x5 → Load access fault. A load failed a PMP check, or encountered a downstream bus error. 0x6 → Store/AMO address misaligned. Hazard3 requires natural alignment of all accesses. 0x7 → Store/AMO access fault. A store/AMO failed a PMP check, or encountered a downstream bus error. Also set if an AMO is attempted on a region that does not support atomics (on RP2350, anything but SRAM). 0x8 → Environment call from U-mode. 0xb → Environment call from M-mode.	RW	0x0

RVCSR: MVAL Register

Offset: 0x343

Table 376. MVAL Register

Bits	Description	Type	Reset
31:0	Machine bad address or instruction. Hardwired to zero.	RO	0x00000000

RVCSR: MIP Register

Offset: 0x344

Description

Machine interrupt pending

Table 377. MIP Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	MEIP	External interrupt pending. The processor transfers to the external interrupt vector when <code>mie.meie</code> , <code>mip.meip</code> and <code>mstatus.mie</code> are all 1. Hazard3 has internal registers to individually filter which external IRQs appear in <code>meip</code> . When <code>meip</code> is 1, this indicates there is at least one external interrupt which is asserted (hence pending in <code>meipa</code>), enabled in <code>meiea</code> , and of priority greater than or equal to the current preemption level in <code>meicontext.preempt</code> .	RO	0x0
10:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7	MTIP	Timer interrupt pending. The processor transfers to the timer interrupt vector when <code>mie.mtie</code> , <code>mip.mtip</code> and <code>mstatus.mie</code> are all 1, unless a software or external interrupt request is also valid at this time.	RW	0x0
6:4	Reserved.	-	-	-
3	MSIP	Software interrupt pending. The processor transfers to the software interrupt vector <code>mie.msie</code> , <code>mip.msip</code> and <code>mstatus.mie</code> are all 1, unless an external interrupt request is also valid at this time.	RW	0x0
2:0	Reserved.	-	-	-

RVCSR: PMPCFG0 Register

Offset: 0x3a0

Description

Physical memory protection configuration for regions 0 through 3

Table 378. PMPCFG0 Register

Bits	Name	Description	Type	Reset
31	R3_L	Lock region 3, and apply it to M-mode as well as U-mode.	RW	0x0
30:29	Reserved.	-	-	-
28:27	R3_A	Address matching type for region 3. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0
26	R3_R	Read permission for region 3. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
25	R3_W	Write permission for region 3	RW	0x0
24	R3_X	Execute permission for region 3. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
23	R2_L	Lock region 2, and apply it to M-mode as well as U-mode.	RW	0x0
22:21	Reserved.	-	-	-
20:19	R2_A	Address matching type for region 2. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0
18	R2_R	Read permission for region 2. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
17	R2_W	Write permission for region 2	RW	0x0

Bits	Name	Description	Type	Reset
16	R2_X	Execute permission for region 2. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
15	R1_L	Lock region 1, and apply it to M-mode as well as U-mode.	RW	0x0
14:13	Reserved.	-	-	-
12:11	R1_A	Address matching type for region 1. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0
10	R1_R	Read permission for region 1. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
9	R1_W	Write permission for region 1	RW	0x0
8	R1_X	Execute permission for region 1. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
7	R0_L	Lock region 0, and apply it to M-mode as well as U-mode.	RW	0x0
6:5	Reserved.	-	-	-
4:3	R0_A	Address matching type for region 0. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0
2	R0_R	Read permission for region 0. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
1	R0_W	Write permission for region 0	RW	0x0
0	R0_X	Execute permission for region 0. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0

RVCSR: PMPCFG1 Register

Offset: 0x3a1

Description

Physical memory protection configuration for regions 4 through 7

Table 379. PMPCFG1 Register

Bits	Name	Description	Type	Reset
31	R7_L	Lock region 7, and apply it to M-mode as well as U-mode.	RW	0x0
30:29	Reserved.	-	-	-
28:27	R7_A	Address matching type for region 7. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0

Bits	Name	Description	Type	Reset
26	R7_R	Read permission for region 7. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
25	R7_W	Write permission for region 7	RW	0x0
24	R7_X	Execute permission for region 7. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
23	R6_L	Lock region 6, and apply it to M-mode as well as U-mode.	RW	0x0
22:21	Reserved.	-	-	-
20:19	R6_A	Address matching type for region 6. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0
18	R6_R	Read permission for region 6. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
17	R6_W	Write permission for region 6	RW	0x0
16	R6_X	Execute permission for region 6. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
15	R5_L	Lock region 5, and apply it to M-mode as well as U-mode.	RW	0x0
14:13	Reserved.	-	-	-
12:11	R5_A	Address matching type for region 5. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0
10	R5_R	Read permission for region 5. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
9	R5_W	Write permission for region 5	RW	0x0
8	R5_X	Execute permission for region 5. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0
7	R4_L	Lock region 4, and apply it to M-mode as well as U-mode.	RW	0x0
6:5	Reserved.	-	-	-
4:3	R4_A	Address matching type for region 4. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RW	0x0
2	R4_R	Read permission for region 4. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0

Bits	Name	Description	Type	Reset
1	R4_W	Write permission for region 4	RW	0x0
0	R4_X	Execute permission for region 4. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RW	0x0

RVCSR: PMPCFG2 Register

Offset: 0x3a2

Description

Physical memory protection configuration for regions 8 through 11

Table 380. PMPCFG2 Register

Bits	Name	Description	Type	Reset
31	R11_L	Lock region 11, and apply it to M-mode as well as U-mode.	RO	0x0
30:29	Reserved.	-	-	-
28:27	R11_A	Address matching type for region 11. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x0
26	R11_R	Read permission for region 11. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
25	R11_W	Write permission for region 11	RO	0x0
24	R11_X	Execute permission for region 11. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
23	R10_L	Lock region 10, and apply it to M-mode as well as U-mode.	RO	0x0
22:21	Reserved.	-	-	-
20:19	R10_A	Address matching type for region 10. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x3
18	R10_R	Read permission for region 10. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x1
17	R10_W	Write permission for region 10	RO	0x1
16	R10_X	Execute permission for region 10. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x1
15	R9_L	Lock region 9, and apply it to M-mode as well as U-mode.	RO	0x0
14:13	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
12:11	R9_A	Address matching type for region 9. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x3
10	R9_R	Read permission for region 9. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x1
9	R9_W	Write permission for region 9	RO	0x1
8	R9_X	Execute permission for region 9. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x1
7	R8_L	Lock region 8, and apply it to M-mode as well as U-mode.	RO	0x0
6:5	Reserved.	-	-	-
4:3	R8_A	Address matching type for region 8. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x3
2	R8_R	Read permission for region 8. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x1
1	R8_W	Write permission for region 8	RO	0x1
0	R8_X	Execute permission for region 8. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x1

RVCSR: PMPCFG3 Register

Offset: 0x3a3

Description

Physical memory protection configuration for regions 12 through 15

Table 381. PMPCFG3 Register

Bits	Name	Description	Type	Reset
31	R15_L	Lock region 15, and apply it to M-mode as well as U-mode.	RO	0x0
30:29	Reserved.	-	-	-
28:27	R15_A	Address matching type for region 15. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x0
26	R15_R	Read permission for region 15. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
25	R15_W	Write permission for region 15	RO	0x0

Bits	Name	Description	Type	Reset
24	R15_X	Execute permission for region 15. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
23	R14_L	Lock region 14, and apply it to M-mode as well as U-mode.	RO	0x0
22:21	Reserved.	-	-	-
20:19	R14_A	Address matching type for region 14. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x0
18	R14_R	Read permission for region 14. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
17	R14_W	Write permission for region 14	RO	0x0
16	R14_X	Execute permission for region 14. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
15	R13_L	Lock region 13, and apply it to M-mode as well as U-mode.	RO	0x0
14:13	Reserved.	-	-	-
12:11	R13_A	Address matching type for region 13. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x0
10	R13_R	Read permission for region 13. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
9	R13_W	Write permission for region 13	RO	0x0
8	R13_X	Execute permission for region 13. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
7	R12_L	Lock region 12, and apply it to M-mode as well as U-mode.	RO	0x0
6:5	Reserved.	-	-	-
4:3	R12_A	Address matching type for region 12. Writing an unsupported value (TOR) will set the region to OFF. 0x0 → Disable region 0x2 → Naturally aligned 4-byte 0x3 → Naturally aligned power-of-two (8 bytes to 4 GiB)	RO	0x0
2	R12_R	Read permission for region 12. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0
1	R12_W	Write permission for region 12	RO	0x0
0	R12_X	Execute permission for region 12. Note R and X are transposed from the standard bit order due to erratum RP2350-E6.	RO	0x0

RVCSR: PMPADDR0 Register

Offset: 0x3b0

Table 382. PMPADDR0 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 0. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR1 Register

Offset: 0x3b1

Table 383. PMPADDR1 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 1. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR2 Register

Offset: 0x3b2

Table 384. PMPADDR2 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 2. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR3 Register

Offset: 0x3b3

Table 385. PMPADDR3 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 3. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR4 Register

Offset: 0x3b4

Table 386. PMPADDR4 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 4. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR5 Register

Offset: 0x3b5

Table 387. PMPADDR5 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-

Bits	Description	Type	Reset
29:0	Physical memory protection address for region 5. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR6 Register

Offset: 0x3b6

Table 388. PMPADDR6 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 6. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR7 Register

Offset: 0x3b7

Table 389. PMPADDR7 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 7. Note all PMP addresses are in units of four bytes.	RW	0x00000000

RVCSR: PMPADDR8 Register

Offset: 0x3b8

Table 390. PMPADDR8 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 8. Note all PMP addresses are in units of four bytes. Hardwired to the address range 0x00000000 through 0x0fffffff , which contains the boot ROM. This range is made accessible to User mode by default. User mode access to this range can be disabled using one of the dynamically configurable PMP regions, or using the permission registers in ACCESSCTRL.	RO	0x01ffff

RVCSR: PMPADDR9 Register

Offset: 0x3b9

Table 391. PMPADDR9 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 9. Note all PMP addresses are in units of four bytes. Hardwired to the address range 0x40000000 through 0x5fffffff , which contains the system peripherals. This range is made accessible to User mode by default. User mode access to this range can be disabled using one of the dynamically configurable PMP regions, or using the permission registers in ACCESSCTRL.	RO	0x13fffff

RVCSR: PMPADDR10 Register

Offset: 0x3ba

Table 392. PMPADDR10 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 10. Note all PMP addresses are in units of four bytes. Hardwired to the address range 0xd0000000 through 0xdfffffff , which contains the core-local peripherals (SIO). This range is made accessible to User mode by default. User mode access to this range can be disabled using one of the dynamically configurable PMP regions, or using the permission registers in ACCESSCTRL.	RO	0x35fffff

RVCSR: PMPADDR11 Register

Offset: 0x3bb

Table 393. PMPADDR11 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 11. Note all PMP addresses are in units of four bytes. Hardwired to all-zeroes. This region is not implemented.	RO	0x00000000

RVCSR: PMPADDR12 Register

Offset: 0x3bc

Table 394. PMPADDR12 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 12. Note all PMP addresses are in units of four bytes. Hardwired to all-zeroes. This region is not implemented.	RO	0x00000000

RVCSR: PMPADDR13 Register

Offset: 0x3bd

Table 395.
PMPADDR13 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 13. Note all PMP addresses are in units of four bytes. Hardwired to all-zeroes. This region is not implemented.	RO	0x00000000

RVCSR: PMPADDR14 Register

Offset: 0x3be

Table 396.
PMPADDR14 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 14. Note all PMP addresses are in units of four bytes. Hardwired to all-zeroes. This region is not implemented.	RO	0x00000000

RVCSR: PMPADDR15 Register

Offset: 0x3bf

Table 397.
PMPADDR15 Register

Bits	Description	Type	Reset
31:30	Reserved.	-	-
29:0	Physical memory protection address for region 15. Note all PMP addresses are in units of four bytes. Hardwired to all-zeroes. This region is not implemented.	RO	0x00000000

RVCSR: TSELECT Register

Offset: 0x7a0

Table 398. TSELECT Register

Bits	Description	Type	Reset
31:2	Reserved.	-	-
1:0	Select trigger to be configured via tdata1/tdata2 On RP2350, four instruction address triggers are implemented, so only the two LSBs of this register are writable.	RW	0x0

RVCSR: TDATA1 Register

Offset: 0x7a1

Description

Trigger configuration data 1

Hazard 3 only supports address/data match triggers (type=2) so this register description includes the [mcontrol](#) fields for this type.

More precisely, Hazard3 only supports exact instruction address match triggers (hardware breakpoints) so many of this register's fields are hardwired.

Table 399. TDATA1 Register

Bits	Name	Description	Type	Reset
31:28	TYPE	Trigger type. Hardwired to type=2, meaning an address/data match trigger	RO	0x2
27	DMODE	If 0, both Debug and M-mode can write the <code>tdata</code> registers at the selected <code>tselect</code> . If 1, only Debug Mode can write the <code>tdata</code> registers at the selected <code>tselect</code> . Writes from other modes are ignored. This bit is only writable from Debug Mode	RW	0x0
26:21	MASKMAX	Value of 0 indicates only exact address matches are supported	RO	0x00
20	HIT	Trigger hit flag. Not implemented, hardwired to 0.	RO	0x0
19	SELECT	Hardwired value of 0 indicates that only address matches are supported, not data matches	RO	0x0
18	TIMING	Hardwired value of 0 indicates that trigger fires before the triggering instruction executes, not afterward	RO	0x0
17:16	SIZENO	Hardwired value of 0 indicates that access size matching is not supported	RO	0x0
15:12	ACTION	Select action to be taken when the trigger fires. 0x0 → Raise a breakpoint exception, which can be handled by the M-mode exception handler 0x1 → Enter debug mode. This action is only selectable when <code>tdata1.dmode</code> is 1.	RW	0x0
11	CHAIN	Hardwired to 0 to indicate trigger chaining is not supported.	RO	0x0
10:7	MATCH	Hardwired to 0 to indicate match is always on the full address specified by <code>tdata2</code>	RO	0x0
6	M	When set, enable this trigger in M-mode	RW	0x0
5:4	Reserved.	-	-	-
3	U	When set, enable this trigger in U-mode	RW	0x0
2	EXECUTE	When set, the trigger fires on the address of an instruction that is executed.	RW	0x0
1	STORE	Hardwired to 0 to indicate store address/data triggers are not supported	RO	0x0
0	LOAD	Hardwired to 0 to indicate load address/data triggers are not supported	RO	0x0

RVCSR: TDATA2 Register

Offset: 0x7a2

Table 400. TDATA2 Register

Bits	Description	Type	Reset
31:0	Trigger configuration data 2 Contains the address for instruction address triggers (hardware breakpoints)	RW	0x00000000

RVCSR: DCSR Register

Offset: 0x7b0

Description

Debug control and status register. Access outside of Debug Mode will cause an illegal instruction exception.

Table 401. DCSR Register

Bits	Name	Description	Type	Reset
31:28	XDEBUGVER	Hardwired to 4: external debug support as per RISC-V 0.13.2 debug specification.	RO	0x4
27:16	Reserved.	-	-	-
15	EBREAKM	When 1, <code>ebreak</code> instructions executed in M-mode will break to Debug Mode instead of trapping	RW	0x0
14:13	Reserved.	-	-	-
12	EBREAKU	When 1, <code>ebreak</code> instructions executed in U-mode will break to Debug Mode instead of trapping.	RW	0x0
11	STEPIE	Hardwired to 0: no interrupts are taken during hardware single-stepping.	RO	0x0
10	STOPCOUNT	Hardwired to 1: <code>mcycle/mcycleh</code> and <code>minstret/minstreth</code> do not increment in Debug Mode.	RO	0x1
9	STOPTIME	Hardwired to 1: core-local timers don't increment in debug mode. External timers (e.g. hart-shared) may be configured to ignore this.	RO	0x1
8:6	CAUSE	Set by hardware when entering debug mode. 0x1 → An ebreak instruction was executed when the relevant <code>dcsr.ebreakx</code> bit was set. 0x2 → The trigger module caused a breakpoint exception. 0x3 → Processor entered Debug Mode due to a halt request, or a reset-halt request present when the core reset was released. 0x4 → Processor entered Debug Mode after executing one instruction with single-stepping enabled.	RO	0x0
5:3	Reserved.	-	-	-
2	STEP	When 1, re-enter Debug Mode after each instruction executed in M-mode or U-mode.	RW	0x0
1:0	PRV	Read the privilege mode the core was in when entering Debug Mode, and set the privilege mode the core will execute in when returning from Debug Mode.	RW	0x3

RVCSR: DPC Register

Offset: 0x7b1

Table 402. DPC Register

Bits	Description	Type	Reset
31:1	Debug program counter. When entering Debug Mode, <code>dpc</code> samples the current program counter, e.g. the address of an <code>ebreak</code> which caused Debug Mode entry. When leaving debug mode, the processor jumps to <code>dpc</code> . The host may read/write this register whilst in Debug Mode.	RW	0x00000000
0	Reserved.	-	-

RVCSR: MCYCLE Register

Offset: 0xb00

Description

Machine-mode cycle counter, low half

Table 403. MCYCLE Register

Bits	Description	Type	Reset
31:0	Counts up once per cycle, when <code>mcountinhibit.cy</code> is 0. Disabled by default to save power.	RW	0x00000000

RVCSR: MINSTRET Register

Offset: 0xb02

Description

Machine-mode instruction retire counter, low half

Table 404. MINSTRET Register

Bits	Description	Type	Reset
31:0	Counts up once per instruction, when <code>mcountinhibit.ir</code> is 0. Disabled by default to save power.	RW	0x00000000

RVCSR: MHPMCOUNTER3, MHPMCOUNTER4, ..., MHPMCOUNTER30, MHPMCOUNTER31 Registers

Offsets: 0xb03, 0xb04, ..., 0xb1e, 0xb1f

Table 405. MHPMCOUNTER3, MHPMCOUNTER4, ..., MHPMCOUNTER30, MHPMCOUNTER31 Registers

Bits	Description	Type	Reset
31:0	Extended performance counter, hardwired to 0.	RO	0x00000000

RVCSR: MCYCLEH Register

Offset: 0xb80

Description

Machine-mode cycle counter, high half

Table 406. MCYCLEH Register

Bits	Description	Type	Reset
31:0	Counts up once per 1 << 32 cycles, when <code>mcountinhibit.cy</code> is 0. Disabled by default to save power.	RW	0x00000000

RVCSR: MINSTRETH Register

Offset: 0xb82

Description

Machine-mode instruction retire counter, low half

Table 407.
MINSTRETH Register

Bits	Description	Type	Reset
31:0	Counts up once per 1 << 32 instructions, when <code>mcountinhibit.ir</code> is 0. Disabled by default to save power.	RW	0x00000000

RVCSR: MHPMCOUNTER3H, MHPMCOUNTER4H, ..., MHPMCOUNTER30H, MHPMCOUNTER31H Registers

Offsets: 0xb83, 0xb84, ..., 0xb9e, 0xb9f

Table 408.
MHPMCOUNTER3H,
MHPMCOUNTER4H, ...,
MHPMCOUNTER30H,
MHPMCOUNTER31H
Registers

Bits	Description	Type	Reset
31:0	Extended performance counter, hardwired to 0.	RO	0x00000000

RVCSR: PMPCFGM0 Register

Offset: 0xbd0

Table 409.
PMPCFGM0 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>PMP M-mode configuration. One bit per PMP region. Setting a bit makes the corresponding region apply to M-mode (like the <code>pmpcfg.L</code> bit) but does not lock the region.</p> <p>PMP is useful for non-security-related purposes, such as stack guarding and peripheral emulation. This extension allows M-mode to freely use any currently unlocked regions for its own purposes, without the inconvenience of having to lock them.</p> <p>Note that this does not grant any new capabilities to M-mode, since in the base standard it is already possible to apply unlocked regions to M-mode by locking them. In general, PMP regions should be locked in ascending region number order so they can't be subsequently overridden by currently unlocked regions.</p> <p>Note also that this is not the same as the rule locking bypass bit in the ePMP extension, which does not permit locked and unlocked M-mode regions to coexist.</p> <p>This is a Hazard3 custom CSR.</p>	RW	0x0000

RVCSR: MEIEA Register

Offset: 0xbe0

Description

External interrupt enable array.

The array contains a read-write bit for each external interrupt request: a 1 bit indicates that interrupt is currently enabled. At reset, all external interrupts are disabled.

If enabled, an external interrupt can cause assertion of the standard RISC-V machine external interrupt pending flag (`mip.meip`), and therefore cause the processor to enter the external interrupt vector. See `meipa`.

There are up to 512 external interrupts. The upper half of this register contains a 16-bit window into the full 512-bit vector. The window is indexed by the 5 LSBs of the write data.

Table 410. MEIEA Register

Bits	Name	Description	Type	Reset
31:16	WINDOW	16-bit read/write window into the external interrupt enable array	RW	0x0000
15:5	Reserved.	-	-	-
4:0	INDEX	Write-only self-clearing field (no value is stored) used to control which window of the array appears in <code>window</code> .	WO	0x00

RVCSR: MEIPA Register

Offset: 0xbe1

Description

External interrupt pending array

Contains a read-only bit for each external interrupt request. Similarly to `meiea`, this register is a window into an array of up to 512 external interrupt flags. The status appears in the upper 16 bits of the value read from `meipa`, and the lower 5 bits of the value written by the same CSR instruction (or 0 if no write takes place) select a 16-bit window of the full interrupt pending array.

A 1 bit indicates that interrupt is currently asserted. IRQs are assumed to be level-sensitive, and the relevant `meipa` bit is cleared by servicing the requestor so that it deasserts its interrupt request.

When any interrupt of sufficient priority is both set in `meipa` and enabled in `meiea`, the standard RISC-V external interrupt pending bit `mip.meip` is asserted. In other words, `meipa` is filtered by `meiea` to generate the standard `mip.meip` flag.

Table 411. MEIPA Register

Bits	Name	Description	Type	Reset
31:16	WINDOW	16-bit read-only window into the external interrupt pending array	RO	-
15:5	Reserved.	-	-	-
4:0	INDEX	Write-only, self-clearing field (no value is stored) used to control which window of the array appears in <code>window</code> .	WO	0x00

RVCSR: MEIFA Register

Offset: 0xbe2

Description

External interrupt force array

Contains a read-write bit for every interrupt request. Writing a 1 to a bit in the interrupt force array causes the corresponding bit to become pending in `meipa`. Software can use this feature to manually trigger a particular interrupt.

There are no restrictions on using `meifa` inside of an interrupt. The more useful case here is to schedule some lower-priority handler from within a high-priority interrupt, so that it will execute before the core returns to the foreground code. Implementers may wish to reserve some external IRQs with their external inputs tied to 0 for this purpose.

Bits can be cleared by software, and are cleared automatically by hardware upon a read of `meinext` which returns the corresponding IRQ number in `meinext.irq` with `meinext.noirq` clear (no matter whether `meinext.update` is written).

`meifa` implements the same array window indexing scheme as `meiea` and `meipa`.

Table 412. MEIFA Register

Bits	Name	Description	Type	Reset
31:16	WINDOW	16-bit read/write window into the external interrupt force array	RW	0x0000
15:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	INDEX	Write-only, self-clearing field (no value is stored) used to control which window of the array appears in <code>window</code> .	WO	0x00

RVCSR: MEIPRA Register

Offset: 0xbe3

Description

External interrupt priority array

Each interrupt has an (up to) 4-bit priority value associated with it, and each access to this register reads and/or writes a 16-bit window containing four such priority values. When less than 16 priority levels are available, the LSBs of the priority fields are hardwired to 0.

When an interrupt's priority is lower than the current preemption priority `meicontext.ppreempt`, it is treated as not being pending for the purposes of `mip.meip`. The pending bit in `meipa` will still assert, but the machine external interrupt pending bit `mip.meip` will not, so the processor will ignore this interrupt. See `meicontext`.

Table 413. MEIPRA Register

Bits	Name	Description	Type	Reset
31:16	WINDOW	16-bit read/write window into the external interrupt priority array, containing four 4-bit priority values.	RW	0x0000
15:5	Reserved.	-	-	-
4:0	INDEX	Write-only, self-clearing field (no value is stored) used to control which window of the array appears in <code>window</code> .	WO	0x00

RVCSR: MEINEXT Register

Offset: 0xbe4

Description

Get next external interrupt

Contains the index of the highest-priority external interrupt which is both asserted in `meipa` and enabled in `meiea`, left-shifted by 2 so that it can be used to index an array of 32-bit function pointers. If there is no such interrupt, the MSB is set.

When multiple interrupts of the same priority are both pending and enabled, the lowest-numbered wins. Interrupts with priority less than `meicontext.ppreempt` – the previous preemption priority – are treated as though they are not pending. This is to ensure that a preempting interrupt frame does not service interrupts which may be in progress in the frame that was preempted.

Table 414. MEINEXT Register

Bits	Name	Description	Type	Reset
31	NOIRQ	Set when there is no external interrupt which is enabled, pending, and has priority greater than or equal to <code>meicontext.ppreempt</code> . Can be efficiently tested with a <code>bltz</code> or <code>bgez</code> instruction.	RO	0x0
30:11	Reserved.	-	-	-
10:2	IRQ	Index of the highest-priority active external interrupt. Zero when no external interrupts with sufficient priority are both pending and enabled.	RO	0x000
1	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
0	UPDATE	Writing 1 (self-clearing) causes hardware to update <code>meicontext</code> according to the IRQ number and preemption priority of the interrupt indicated in <code>noirq/irq</code> . This should be done in a single atomic operation, i.e. <code>csrrsi a0, meinext, 0x1</code> .	SC	0x0

RVCSR: MEICONTEXT Register

Offset: 0xbe5

Description

External interrupt context register

Configures the priority level for interrupt preemption, and helps software track which interrupt it is currently in. The latter is useful when a common interrupt service routine handles interrupt requests from multiple instances of the same peripheral.

A three-level stack of preemption priorities is maintained in the `preempt`, `ppreempt` and `pppreempt` fields. The priority stack is saved when hardware enters the external interrupt vector, and restored by an `mret` instruction if `meicontext.mreteirq` is set.

The top entry of the priority stack, `preempt`, is used by hardware to ensure that only higher-priority interrupts can preempt the current interrupt. The next entry, `ppreempt`, is used to avoid servicing interrupts which may already be in progress in a frame that was preempted. The third entry, `pppreempt`, has no hardware effect, but ensures that `preempt` and `ppreempt` can be correctly saved/restored across arbitrary levels of preemption.

Table 415.
MEICONTEXT Register

Bits	Name	Description	Type	Reset
31:28	PPPREEMPT	Previous <code>ppreempt</code> . Set to <code>ppreempt</code> on priority save, set to zero on priority restore. Has no hardware effect, but ensures that when <code>meicontext</code> is saved/restored correctly, <code>preempt</code> and <code>ppreempt</code> stack correctly through arbitrarily many preemption frames.	RW	0x0
27:24	PPREEMPT	Previous <code>preempt</code> . Set to <code>preempt</code> on priority save, restored to <code>pppreempt</code> on priority restore. IRQs of lower priority than <code>ppreempt</code> are not visible in <code>meinext</code> , so that a preemptee is not re-taken in the preempting frame.	RW	0x0
23:21	Reserved.	-	-	-
20:16	PREEMPT	Minimum interrupt priority to preempt the current interrupt. Interrupts with lower priority than <code>preempt</code> do not cause the core to transfer to an interrupt handler. Updated by hardware when <code>meinext.update</code> is written, or when hardware enters the external interrupt vector. If an interrupt is present in <code>meinext</code> when this field is updated, then <code>preempt</code> is set to one level greater than that interrupt's priority. Otherwise, <code>ppreempt</code> is set to one level greater than the maximum interrupt priority, disabling preemption.	RW	0x00
15	NOIRQ	Not in interrupt (read/write). Set to 1 at reset. Set to <code>meinext.noirq</code> when <code>meinext.update</code> is written. No hardware effect.	RW	0x1

Bits	Name	Description	Type	Reset
14:13	Reserved.	-	-	-
12:4	IRQ	Current IRQ number (read/write). Set to <code>meinext.irq</code> when <code>meinext.update</code> is written. No hardware effect.	RW	0x000
3	MTIESAVE	Reads as the current value of <code>mie.mtie</code> , if <code>clearts</code> is set by the same CSR access instruction. Otherwise reads as 0. Writes are ORed into <code>mie.mtie</code> .	RO	0x0
2	MSIESAVE	Reads as the current value of <code>mie.msie</code> , if <code>clearts</code> is set by the same CSR access instruction. Otherwise reads as 0. Writes are ORed into <code>mie.msie</code> .	RO	0x0
1	CLEARTS	Write-1 self-clearing field. Writing 1 will clear <code>mie.mtie</code> and <code>mie.msie</code> , and present their prior values in the <code>mtiesave</code> and <code>msiesave</code> of this register. This makes it safe to re-enable IRQs (via <code>mstatus.mie</code>) without the possibility of being preempted by the standard timer and soft interrupt handlers, which may not be aware of Hazard3's interrupt hardware. The clear due to <code>clearts</code> takes precedence over the set due to <code>mtiesave/msiesave</code> , although it would be unusual for software to write both on the same cycle.	SC	0x0
0	MRETEIRQ	If 1, enable restore of the preemption priority stack on <code>mret</code> . This bit is set on entering the external interrupt vector, cleared by <code>mret</code> , and cleared upon taking any trap other than an external interrupt. Provided <code>meicontext</code> is saved on entry to the external interrupt vector (before enabling preemption), is restored before exiting, and the standard software/timer IRQs are prevented from preempting (e.g. by using <code>clearts</code>), this flag allows the hardware to safely manage the preemption priority stack even when an external interrupt handler may take exceptions.	RW	0x0

RVCSR: MSLEEP Register

Offset: 0xbf0

Description

M-mode sleep control register

Table 416. MSLEEP Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	SLEEPONBLOCK	Enter the deep sleep state configured by <code>msleep.deepsleep/msleep.powerdown</code> on a <code>h3.block</code> instruction, as well as a standard <code>wfi</code> . If this bit is clear, a <code>h3.block</code> is always implemented as a simple pipeline stall.	RW	0x0

Bits	Name	Description	Type	Reset
1	POWERDOWN	<p>Release the external power request when going to sleep. The function of this is platform-defined – it may do nothing, it may do something simple like clock-gating the fabric, or it may be tied to some complex system-level power controller.</p> <p>When waking, the processor reasserts its external power-up request, and will not fetch any instructions until the request is acknowledged. This may add considerable latency to the wakeup.</p>	RW	0x0
0	DEEPSLEEP	Deassert the processor clock enable when entering the sleep state. If a clock gate is instantiated, this allows most of the processor (everything except the power state machine and the interrupt and halt input registers) to be clock gated whilst asleep, which may reduce the sleep current. This adds one cycle to the wakeup latency.	RW	0x0

RVCSR: DMDATA0 Register

Offset: 0xbff

Table 417. DMDATA0 Register

Bits	Description	Type	Reset
31:0	The Debug Module's DATA0 register is mapped into Hazard3's CSR space so that the Debug Module can exchange data with the core by executing CSR access instructions (this is used to implement the Abstract Access Register command). Only accessible in Debug Mode.	RW	0x00000000

RVCSR: CYCLE Register

Offset: 0xc00

Table 418. CYCLE Register

Bits	Description	Type	Reset
31:0	Read-only U-mode alias of mcycle, accessible when <code>mcounteren.cy</code> is set	RO	0x00000000

RVCSR: INSTRET Register

Offset: 0xc02

Table 419. INSTRET Register

Bits	Description	Type	Reset
31:0	Read-only U-mode alias of minstret, accessible when <code>mcounteren.ir</code> is set	RO	0x00000000

RVCSR: CYCLEH Register

Offset: 0xc80

Table 420. CYCLEH Register

Bits	Description	Type	Reset
31:0	Read-only U-mode alias of mcycleh, accessible when <code>mcounteren.cy</code> is set	RO	0x00000000

RVCSR: INSTRETH Register

Offset: 0xc82

Table 421. INSTRETH Register

Bits	Description	Type	Reset
31:0	Read-only U-mode alias of minstreth, accessible when <code>mcounteren.ir</code> is set	RO	0x00000000

RVCSR: MVENDORID Register

Offset: 0xf11

Description

Vendor ID

Table 422. MVENDORID Register

Bits	Name	Description	Type	Reset
31:7	BANK		RO	-
6:0	OFFSET		RO	-

RVCSR: MARCHID Register

Offset: 0xf12

Table 423. MARCHID Register

Bits	Description	Type	Reset
31:0	Architecture ID (Hazard3)	RO	0x0000001b

RVCSR: MIMPID Register

Offset: 0xf13

Table 424. MIMPID Register

Bits	Description	Type	Reset
31:0	Implementation ID	RO	-

RVCSR: MHARTID Register

Offset: 0xf14

Description

Hardware thread ID

Table 425. MHARTID Register

Bits	Description	Type	Reset
31:0	On RP2350, core 0 has a hart ID of 0, and core 1 has a hart ID of 1.	RO	-

RVCSR: MCONFIGPTR Register

Offset: 0xf15

Table 426. MCONFIGPTR Register

Bits	Description	Type	Reset
31:0	Pointer to configuration data structure (hardwired to 0)	RO	0x00000000

3.9. Arm/RISC-V Architecture Switching

RP2350 supports both Arm and RISC-V processor architectures. SDK-based programs which do not contain assembly code typically run unmodified on either architecture by providing the appropriate build flag.

There are two processor sockets on RP2350, referred to as core 0 and core 1 throughout this document. Each socket can be occupied *either* by a Cortex-M33 processor (implementing the Armv8-M Main architecture, plus extensions) or by a Hazard3 processor (implementing the RV32IMAC architecture, plus extensions).

When a processor reset is removed, hardware samples the [ARCHSEL](#) register in the OTP control register block to determine which processor to connect to that socket. The unused processor is held in reset indefinitely, with its clock inputs gated. The default and allowable values of the [ARCHSEL](#) register are determined by critical OTP flags:

1. If [CRIT0_ARM_DISABLE](#) is set, only RISC-V is allowed.
2. Else if [CRIT0_RISCV_DISABLE](#) is set, only Arm is allowed.
3. Else if [CRIT1_SECURE_BOOT_ENABLE](#) is set, only Arm is allowed.
4. Else if [CRIT1_BOOT_ARCH](#) is set, both architectures are permitted, and the default is RISC-V.
5. If none of the above flags are set, both architectures are permitted, and the default is Arm.

No [CRIT1](#) flags are set by default, so on devices where both architectures are available, the default is Arm. To change the default architecture to RISC-V, set the [CRIT1_BOOT_ARCH](#) flag to 1.

Enabling secure boot disables the RISC-V cores because the RP2350 bootrom does not implement secure boot for RISC-V. This prevents a bad actor from side-stepping secure boot by switching architectures.

RP2350 only samples the [ARCHSEL](#) register when a processor is reset. Its value is ignored at all other times, so software can program the register before a watchdog reset to implement a software-initiated switch between architectures.

Read the [ARCHSEL_STATUS](#) register to check the ARCHSEL value most recently sampled by each processor.

3.9.1. Automatic Switching

RP2350 binaries contain a binary marker recognised by the bootrom. This marker:

- contains additional information such as the binary's entry point and the intended architecture: Arm, RISC-V, or both
- helps detect when a flash device is connected
- helps verify that the flash device was accessed using the correct SPI parameters

When booting with core 0 in Arm architecture mode, upon detecting a bootable RISC-V binary, the bootrom automatically resets both cores and switches them to RISC-V architecture mode. After the reset, the bootrom detects that the binary and processor architectures match, so the binary launches normally.

Likewise, when booting with core 0 in RISC-V architecture mode, upon detecting a bootable Arm binary, the bootrom automatically resets both cores and switches them to RISC-V architecture mode.

As a result, the USB bootloader, which runs on both Arm and RISC-V, can accept a UF2 image download for either architecture, and automatically boot it using the correct processors.

3.9.2. Mixed Architecture Combinations

The [ARCHSEL](#) register has one bit for each processor socket, so it is possible to request mixed combinations of Arm and RISC-V processors: either Arm core 0 and RISC-V core 1, or RISC-V core 0 and Arm core 1.

Practical applications for this are limited, since this requires two separate program images. The two cores interoperate normally, including shared exclusives via the global monitor: a shared variable can be safely, concurrently accessed by an Arm processor performing [ldrex](#), [strex](#) instructions and a RISC-V processor performing [amoadd.w](#) instructions, for example.

Hardware supports debugging for a mixture of Arm and RISC-V processors, though this may prove challenging on the host software side. Debug resources for unused processors are dynamically marked as non-PRESENT in the top-level CoreSight ROM table.

Chapter 4. Memory

RP2350 has embedded ROM, OTP and SRAM. RP2350 provides access to external flash via a QSPI interface.

4.1. ROM

A 32 KiB read-only memory (ROM) appears at address `0x00000000`. The ROM contents are fixed permanently at the time the silicon is manufactured. [Chapter 5](#) describes the ROM contents in detail, but in summary it contains:

- Boot code:
 - Initial entry point for both cores
 - Core 1 low-power holding pen ([Section 5.9](#))
 - Booting programs in flash (XIP), loaded from flash to SRAM, loaded from OTP, or preloaded in SRAM
 - Secure boot with public key fingerprint in OTP ([Section 10.1.1](#))
 - Anti-rollback version protection with version number in OTP ([Section 5.1.5](#))
 - Flash partitions for protection, A/B image boot, or separate code/data upgrades ([Section 5.1.2](#))
 - Position-independent XIP from flash partitions, using hardware address translation ([Section 5.1.17](#))
 - "Try before you buy" phased upgrades ([Section 5.1.16](#))
 - Automatic architecture switch for a RISC-V binary run on Arm, or vice versa ([Section 5.1.18](#))
- USB bootloader:
 - Mass storage interface for drag and drop of UF2 flash and SRAM binaries ([Section 5.5](#))
 - PICOBLOCK interface for advanced operations like OTP programming ([Section 5.6](#))
- UART bootloader: minimal shell to load an SRAM binary from a host microcontroller ([Section 5.8](#))
- Runtime APIs, exported through ROM symbol table:
 - Flash programming
 - OTP programming
 - Querying partition table, and translating flash virtual to physical addresses
 - APIs to support phased upgrades and A/B image boot
 - Get system information, including per-boot random number
 - Several others, as documented in [Section 5.4](#)

The ROM offers single-cycle access, and has a dedicated AHB5 arbiter, so it can be accessed simultaneously with other memory devices. Writing to the ROM has no effect, and no bus fault is generated on write.

The ROM is covered by IDAU regions enumerated in [Section 10.2.2](#). These aid in partitioning the bootrom between Secure and Non-secure code: in particular the USB/UART bootloader runs as a Non-secure client application on Arm, to reduce the attack surface of the secure boot implementation.

Certain ROM features are not implemented on RISC-V, most notably secure boot.

4.2. SRAM

There is a total of 520 KiB (520×1024 bytes) of on-chip SRAM. For performance reasons, this memory is physically partitioned into ten *banks*, but logically it still behaves as a single, flat 520 KiB memory. RP2350 does not restrict the data stored in each bank: you can use any bank to store processor code, data buffers, or a mixture of the two. There are eight $16\text{ KiB} \times 32\text{-bit}$ banks (64 KiB each) and two $1\text{k} \times 32\text{-bit}$ banks (4 KiB each).

NOTE

Banking is a *physical* partitioning of SRAM which improves performance by allowing multiple simultaneous accesses. *Logically*, there is a single 520 KiB contiguous memory.

Each SRAM bank is accessed via a dedicated AHB5 arbiter. This means different bus managers can access different SRAM banks in parallel, so up to six 32-bit SRAM accesses can take place every system clock cycle (one per manager).

SRAM is mapped to system addresses starting at `0x20000000`. The first 256 KiB address region, up to and including `0x2003ffff`, is word-striped across the first four 64 KiB banks. The next 256 KiB address region, up to `0x2007ffff` is word-striped across the remaining four 64 KiB banks. The watermark between these two striped regions, at `0x20040000`, marks the boundary between the SRAM0 and SRAM1 power domains.

Consecutive words in the system address space are routed to different RAM banks as shown in [Table 427](#). This scheme is referred to as **sequential interleaving**, and improves bus parallelism for typical memory access patterns.

Table 427. SRAM bank0/1/2/3 striped mapping.

System address	SRAM Bank	SRAM word address
<code>0x20000000</code>	Bank 0	0
<code>0x20000004</code>	Bank 1	0
<code>0x20000008</code>	Bank 2	0
<code>0x2000000c</code>	Bank 3	0
<code>0x20000010</code>	Bank 0	1
<code>0x20000014</code>	Bank 1	1
<code>0x20000018</code>	Bank 2	1
<code>0x2000001c</code>	Bank 3	1
<code>0x20000020</code>	Bank 0	2
<code>0x20000024</code>	Bank 1	2
<code>0x20000028</code>	Bank 2	2
<code>0x2000002c</code>	Bank 3	2
etc		

The top two 4 KiB regions (starting at `0x20080000` and `0x20081000`) map directly to the smaller 4 KiB memory banks. Software may choose to use these for per-core purposes (e.g. stack and frequently-executed code), guaranteeing that the processors never stall on these accesses. Like all SRAM on RP2350, these banks have single-cycle access from all managers, (provided no other managers access the bank in the same cycle) so it is reasonable to treat memory as a single 520 KiB device.

NOTE

RP2040 had a non-striped SRAM mirror. RP2350 no longer has a non-striped mirror, to avoid mapping the same SRAM location as both Secure and Non-secure. You can still achieve some explicit bandwidth partitioning by allocating data across two 256 KiB blocks of 4-way-striped SRAM.

4.2.1. Other On-chip Memory

Besides the 520 KiB main memory, there are two other dedicated RAM blocks that may be used in some circumstances:

- Cache lines can be individually pinned within the XIP address space for use as SRAM, up to the total cache size of 16 KiB (see [Section 4.4.1.3](#)). Unpinned cache lines remain available for transparent caching of XIP accesses.
- If USB is not used, the USB data DPRAM can be used as a 4 KiB memory starting at `0x50100000`.

There is also 1 KiB of dedicated boot RAM, hardwired to Secure access only, whose contents and layout is defined by the boot ROM – see [Chapter 5](#).

NOTE

Memory in the peripheral address space (addresses starting with `0x4`, `0x5` or `0xd`) does not support code execution. This includes USB RAM and boot RAM. These address ranges are made IDAU-Exempt to simplify assigning peripherals to security domains using ACCESSCTRL, and consequently must be made non-executable to avoid the possibility of Non-secure-writable, Secure-executable memory.

4.3. Boot RAM

Boot RAM is a 1 KiB (256×32) SRAM dedicated for use by the bootrom. It is slower than main SRAM, as it is accessed over APB, taking three cycles for a read and four cycles for a write.

Boot RAM is used for myriad purposes during boot, including the initial pre-boot stack. After the bootrom enters the user application, boot RAM contains state for the user-facing ROM APIs, such as the resident partition table used for flash programming protection, and a copy of the flash XIP setup function (formerly known as `boot2`) to quickly re-initialise flash XIP modes following serial programming operations.

Boot RAM is hardwired to permit Secure access only (Arm) or Machine-mode access only (RISC-V). It is physically impossible to execute from boot RAM, regardless of MPU configuration, as it is on the APB peripheral bus segment, which is not wired to the processor instruction fetch ports.

Since boot RAM is in the XIP RAM power domain, it is always powered when the switched core domain is powered. This simplifies SRAM power management in the bootrom, because it doesn't have to power up any RAM before it has a place to store the call stack.

Boot RAM supports the standard atomic set/clear/XOR accesses used by other peripherals on RP2350 ([Section 2.1.3](#)).

It is possible to use boot RAM for user-defined purposes, but this is not recommended, as it may cause ROM APIs to behave unpredictably. Calling into the ROM could modify data stored in the boot RAM.

4.3.1. List of Registers

A small number of registers are located on the same bus endpoint as boot RAM, starting from an offset of `0x800` above the boot RAM base address of `0x400e0000` (defined as `BOOTRAM_BASE` in SDK).

Table 428. List of
BOOTRAM registers

Offset	Name	Info
0x800	WRITE_ONCE0	This registers always ORs writes into its current contents. Once a bit is set, it can only be cleared by a reset.
0x804	WRITE_ONCE1	This registers always ORs writes into its current contents. Once a bit is set, it can only be cleared by a reset.
0x808	BOOTLOCK_STAT	Bootlock status register. 1=unclaimed, 0=claimed. These locks function identically to the SIO spinlocks, but are reserved for bootrom use.
0x80c	BOOTLOCK0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.
0x810	BOOTLOCK1	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.
0x814	BOOTLOCK2	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.
0x818	BOOTLOCK3	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.
0x81c	BOOTLOCK4	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.
0x820	BOOTLOCK5	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.
0x824	BOOTLOCK6	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.
0x828	BOOTLOCK7	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.

BOOTRAM: WRITE_ONCE0 Register

Offset: 0x800

Table 429.
WRITE_ONCE0
Register

Bits	Description	Type	Reset
31:0	This registers always ORs writes into its current contents. Once a bit is set, it can only be cleared by a reset.	RW	0x00000000

BOOTRAM: WRITE_ONCE1 Register

Offset: 0x804

Table 430.
WRITE_ONCE1
Register

Bits	Description	Type	Reset
31:0	This registers always ORs writes into its current contents. Once a bit is set, it can only be cleared by a reset.	RW	0x00000000

BOOTRAM: BOOTLOCK_STAT Register

Offset: 0x808

Table 431.
BOOTLOCK_STAT
Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	Bootlock status register. 1=unclaimed, 0=claimed. These locks function identically to the SIO spinlocks, but are reserved for bootrom use.	RW	0xff

BOOTRAM: BOOTLOCK0 Register

Offset: 0x80c

Table 432.
BOOTLOCK0 Register

Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

BOOTRAM: BOOTLOCK1 Register

Offset: 0x810

Table 433.
BOOTLOCK1 Register

Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

BOOTRAM: BOOTLOCK2 Register

Offset: 0x814

Table 434.
BOOTLOCK2 Register

Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

BOOTRAM: BOOTLOCK3 Register

Offset: 0x818

Table 435.
BOOTLOCK3 Register

Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

BOOTRAM: BOOTLOCK4 Register

Offset: 0x81c

Table 436.
BOOTLOCK4 Register

Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

BOOTRAM: BOOTLOCK5 Register

Offset: 0x820

Table 437.
BOOTLOCK5 Register

Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

BOOTRAM: BOOTLOCK6 Register

Offset: 0x824

Table 438.
BOOTLOCK6 Register

Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

BOOTRAM: BOOTLOCK7 Register

Offset: 0x828

Table 439.
BOOTLOCK7 Register

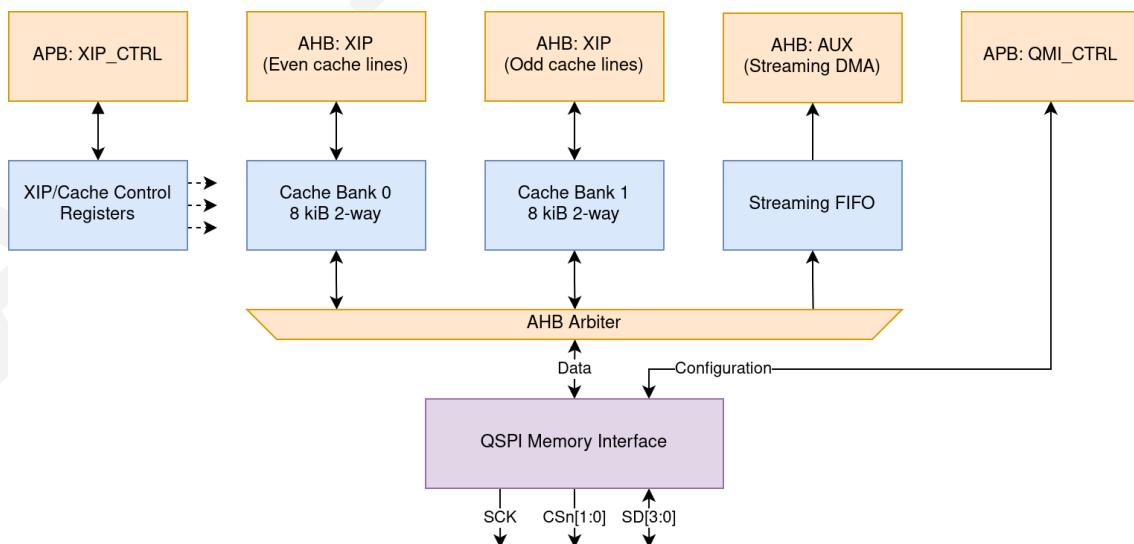
Bits	Description	Type	Reset
31:0	Read to claim and check. Write to unclaim. The value returned on successful claim is $1 \ll n$, and on failed claim is zero.	RW	0x00000000

4.4. External Flash and PSRAM (XIP)

RP2350 can access external flash and PSRAM via its **execute-in-place** (XIP) subsystem. The term execute-in-place refers to external memory mapped directly into the chip's internal address space. This enables you to execute code as-is from the external memory without explicitly copying into on-chip SRAM. For example, a processor instruction fetch from AHB address [0x10001234](#) results in a QSPI memory interface fetch from address [0x001234](#) in an external flash device.

A 16 KiB on-chip cache retains the values of recent reads and writes. This reduces the chances that XIP bus accesses must access external memory, improving the average throughput and latency of the XIP interface. The cache is physically structured as two 8 KiB banks, interleaving odd and even cache lines of 8-byte granularity over the two banks. This allows processors to access multiple cache lines during the same cycle. Logically, the XIP cache behaves as a single 16 KiB cache.

Figure 15. Flash execute-in-place (XIP) subsystem. The cache is split into two banks for performance, but behaves as a single 16 KiB cache. XIP accesses first hit the cache. If a cache entry is not found, the QMI generates an external serial access, adds the resulting data to the cache, and forwards it on to the system bus (for reads) or merges it with the AHB write data (for writes).



When booting from flash, the RP2350 bootrom (Chapter 5) sets up a baseline QMI execute-in-place configuration. User code may later reconfigure this to improve performance for a specific flash device. QSPI clock divisors can be changed at any time, including whilst executing from XIP. Other reconfiguration requires a momentary disable of the interface.

4.4.1. XIP Cache

The cache is 16 KiB, two-way set-associative, 1 cycle hit. It is internal to the XIP subsystem, and only involved in accesses to the QSPI memory interface, so software does not have to consider cache coherence unless performing flash programming operations. It caches accesses to a 26-bit downstream XIP address space. On RP2350, the lower half of this space is occupied by two 16 MiB windows for the two QMI chip selects. RP2350 reserves the remainder for future expansion, but you can use the space to pin cache lines outside of the QMI address space for use as cache-as-SRAM (Section 4.4.1.3). The 26-bit XIP address space is mirrored multiple times in the RP2350 address space, decoded on bits 27:26 of the system bus address:

- **0x10...** : Cached XIP access
- **0x14...** : Uncached XIP access
- **0x18...** : Cache maintenance writes
- **0x1c...** : Uncached, untranslated XIP access – bypass QMI address translation

You can disable cache lookup separately for Secure and Non-secure accesses via the `CTRL.EN_SECURE` and `CTRL.EN_NONSECURE` register bits. The `CTRL` register contains controls to disable Secure/Non-secure access to the uncached and uncached/untranslated XIP windows, which avoids duplicate mappings that may otherwise require additional SAU or PMP regions.

4.4.1.1. Cache Maintenance

Cache maintenance is performed on a line-by-line basis by writing into the cache maintenance mirror of the XIP address space, starting at **0x18000000**. Cache lines are 8 bytes in size. Write data is ignored; instead, the 3 LSBs of the address select the maintenance operation:

- **0x0**: Invalidate by set/way
- **0x1**: Clean by set/way
- **0x2**: Invalidate by address
- **0x3**: Clean by address
- **0x7**: Pin cache set/way at address (Section 4.4.1.3)

invalidate

marks a cache line as no longer containing meaningful data. The cache deletes the cache line. Used when the external memory has been modified in a way that the cache would not automatically know about, such as a flash programming operation.

clean

instructs the cache to write out any data stored in the cache as a result of a previous cached write access that has not yet been written out to the downstream memory. Used to make cached writes available to uncached reads or when cache contents are about to be lost, but the external memory is to stay powered (for example, when the system is about to power down).

clean by set/way

used when a particular physical line of the cache is to be maintained. Bit 13 of the system bus address selects the cache way. Bits 12:3 of the address select a particular cache line within that way. Mainly used to iterate exhaustively over all cache lines (for example, during a full cache flush).

clean by address

looks up the address in the cache, then performs the requested clean/invalidate if that line is currently allocated in the cache. Used when only a particular range of XIP addresses needs to be maintained, for example, a flash page that was just programmed. Usually faster than a full flush, because the real cost of a cache flush is not in the maintenance operations, but the large number of resulting cache misses.

pin

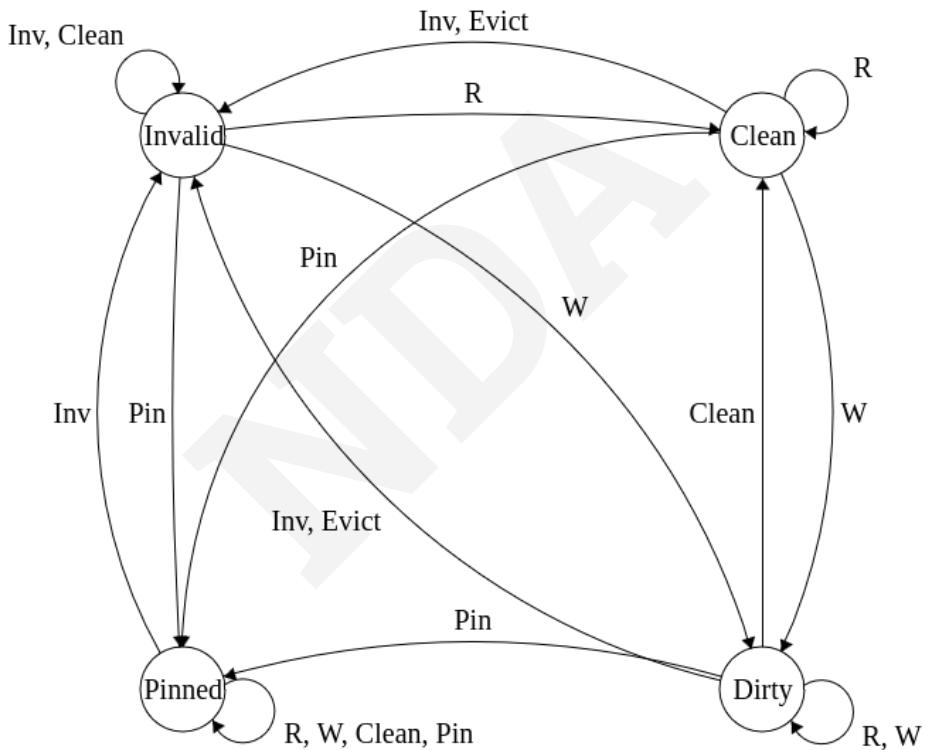
used to prevent a particular physical line of the cache from being evicted. Used to mark important external memory contents that must get guaranteed cache hits, or to allocate cache lines for use as cache-as-SRAM. If a cached access to some other address misses the cache and attempts to evict a pinned cache line, the eviction fails, and the access is downgraded to an uncached access.

By default, cache maintenance is Secure-only. Non-secure writes to the cache maintenance address window have no effect and return a bus error. Non-secure cache maintenance can be enabled by setting the `CTRL.MAINT_NONSEC` register bit, but this is not recommended if Secure software may perform cached XIP accesses.

4.4.1.2. Cache Line States

The changes to a cache line caused by cached accesses and maintenance operations can be summarised by a set of state transitions.

Figure 16. State transition diagram for each cache line. Inv, Clean and Pin represent invalidate/clean/pin maintenance operations, respectively. R and W represent cached reads and writes. Evict represents a cache line deallocation to make room for a new allocation due to a read/write cache miss.



Initially, the state of all cache lines is undefined. When booting from flash, the bootrom performs an invalidate by set/way on every line of the cache to force them to a known state. In the diagram above, all states have an **Inv** arc to the invalid state.

A **dirty** cache line contains data not yet propagated to downstream memory.

A **clean** cache line does not contain data not yet propagated to downstream memory.

Accessing an invalid cache line causes an **allocation**: the cache fetches the corresponding data from downstream memory, stores it in the cache, then marks the cache line as clean or dirty. The cache also stores part of the downstream address, known as the **tag**, to recall the downstream address stored in each cache line. Read allocations enter the clean state, so the cache line can be safely freed at any time. Write allocations enter the dirty state, so the cache line must propagate downstream before it can be freed.

Writing to a clean cache line marks it as dirty because the cache now contains write data that has not propagated downstream. The line can be explicitly returned to the clean state using a clean maintenance operation (`0x1` or `0x3`), but this is not required. Typically, the cache automatically propagates dirty cache lines downstream when it needs to

reallocate them.

Evictions happen when a cached read or write needs to allocate a cache line that is already in the clean or dirty state. The eviction transitions the line momentarily to the invalid state, ready for allocation. For clean cache lines, this happens instantaneously. For dirty cache lines, the cache must first propagate the cache line contents downstream before it can safely enter the invalid state.

Cache lines enter the pinned state using a pin maintenance operation (0x7) and exit only by an invalidate maintenance operation (0x0 or 0x2).

ⓘ NOTE

The pin maintenance operation only marks the line as pinned; it does not perform any copying of data. When pinning lines that exist in external memory devices, you must first pin the line, then copy the downstream data into the pinned line by reading from the uncached XIP window.

4.4.1.3. Cache-as-SRAM

When you disabled the cache of RP2040, the cache would map the entire cache memory at 0x15000000. RP2350 replaces this with the ability to pin individual cache lines. You can use this in the following ways:

- Pin the entire cache at some address range to use the entire cache as SRAM
- Pin one full cache way to make half of the cache available for cache-as-SRAM use (the remaining cache way still functions as usual)
- Pin an address range that maps critical flash contents

ⓘ NOTE

Pinned cache lines are not accessible when the cache is disabled via the `CTRL` register (`CTRL.EN_SECURE` or `CTRL.EN_NONSECURE` depending on security level of the bus access).

Because the QMI only occupies the lower half of the 64 MiB XIP address space, you can pin cache lines outside of the QMI address range (e.g. at the top of the XIP space) to avoid interfering with any QMI accesses. As a general rule, the more cache you pin, the lower the cache hit rate for other accesses.

Cache lines are pinned using the pin maintenance operation (0x7), which performs the following steps:

1. An implicit invalidate-by-address operation (0x2) using the full address of the maintenance operation
 - This ensures that each address is allocated in only one cache way (required for correct cache operation)
2. Select the cache line to be pinned, using bit 13 to select the cache way, and bits 12:3 to select the cache set (as with 0x0/0x1 invalidate/clean by set/way commands)
3. Write the address to the cache line's tag entry
4. Change the cache line's state to pinned (as per the state diagram in [Section 4.4.1.2](#))
5. Update the cache line's tag with the full address of the maintenance operation

After a pin operation, cached reads and writes to the specified address always hit the cache until that cache line is either invalidated or pinned to a different address.

NOTE

Pinning two addresses which are equal modulo cache size pins the same cache line twice. It does *not* pin two different cache lines. The second pin will overwrite the first.

When a cached access hits a pinned cache line, it behaves the same as a dirty line. The cache reads and writes as if allocated in the cache by normal means.

Cache eviction policy is random, and the cache only makes one attempt to select an eviction way. If the cache selects to evict a pinned line, the eviction fails, and the access is demoted to an uncached access. As a result, a cache with one way pinned does not behave exactly the same as a direct-mapped 8 KiB cache, but average-case performance is similar.

Cache line states are stored in the cache tag memory stored in the XIP memory power domain. This memory contents do not change on reset, so pinned lines remain pinned across resets. If the XIP memory power domain is not powered down, memory contents do not change across power cycles of the switched core reset domain. The bootrom clears the tag memory upon entering the flash boot or NSBOOT (USB boot) path, but watchdog scratch vector reboots can boot directly into pinned XIP cache lines.

4.4.2. QSPI Memory Interface (QMI)

Uncached accesses and cache misses require access to external memory. The QSPI memory interface (QMI) provides this access, as documented in [Section 12.14](#). The QMI supports:

- Up to two external QSPI devices, with separate chip selects and shared clock/data pins
- Memory-mapped reads and writes (writes must be enabled via `CTRL.WRITABLE_M0`/`CTRL.WRITABLE_M1`)
- Serial/dual/quad-SPI transfer formats, configured per chip select
- SCK speeds as high as `clk_sys`, configured per chip select
- 8/16/32-bit accesses for uncached accesses, and 64-bit accesses for cache line fills
- Automatic chaining of sequentially addressed accesses into a single QSPI transfer
- Address translation (4 × 4 MiB windows per QSPI device)
 - Flash storage addresses can differ from runtime addresses, e.g. for multiple OTA upgrade image slots
 - Allows code and data segments, or Secure and Non-secure images, to be mapped separately
- Direct-mode FIFO interface for programming and configuring external QSPI devices

XIP accesses via the two cache AHB ports, and from the DMA streaming hardware, arbitrate for access to the QMI. A separate APB port configures the QMI.

The QMI is a new memory interface designed for RP2350, replacing the SSI peripheral on RP2040.

4.4.3. Streaming DMA Interface

As the flash is generally much larger than on-chip SRAM, it's often useful to stream chunks of data into memory from flash. It's convenient to have the DMA stream this data in the background while software in the foreground does other things. It's even more convenient if code can continue to execute from flash whilst this takes place.

This doesn't interact well with standard XIP operation because QMI serial transfers force lengthy bus stalls on the DMA. These stalls are tolerable for a processor because an in-order processor tends to have nothing better to do while waiting for an instruction fetch to retire, and because typical code execution tends to have much higher cache hit rates than bulk streaming of infrequently accessed data. In contrast, stalling the DMA prevents any other active DMA channels from making progress during this time, slowing overall DMA throughput.

The `STREAM_ADDR` and `STREAM_CTR` registers are used to program a linear sequence of flash reads. The XIP subsystem performs these reads in the background in a best-effort fashion. To minimise impact on code executed from flash whilst the stream is ongoing, the streaming hardware has lower priority access to the QMI than regular XIP accesses, and there is a brief cooldown (9 cycles) between the last XIP cache miss and resuming streaming. This avoids increases in initial access latency on XIP cache misses.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/flash/xip_stream/flash_xip_stream.c Lines 45 - 48

```
45     while (!(xip_ctrl_hw->stat & XIP_STAT_FIFO_EMPTY))
46         (void) xip_ctrl_hw->stream_fifo;
47     xip_ctrl_hw->stream_addr = (uint32_t) &random_test_data[0];
48     xip_ctrl_hw->stream_ctr = count_of(random_test_data);
```

The streamed data is pushed to a small FIFO, which generates DREQ signals that tell the DMA to collect the streamed data. As the DMA does not initiate a read until *after* reading the data from flash, the DMA does not stall when accessing the data. The DMA can then retrieve this data through the auxiliary AHB port, which provides direct single-cycle access to the streaming data FIFO.

On RP2350, you can also use the auxiliary AHB port to access the QMI direct-mode FIFOs. This is faster than accessing the FIFOs through the QMI APB configuration port. When QMI access chaining is enabled, the streaming XIP DMA is close to the maximum theoretical QSPI throughput, but the direct-mode FIFOs are available on AHB for situations that require 100% of the theoretical throughput.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/flash/xip_stream/flash_xip_stream.c Lines 58 - 70

```
58     const uint dma_chan = 0;
59     dma_channel_config cfg = dma_channel_get_default_config(dma_chan);
60     channel_config_set_read_increment(&cfg, false);
61     channel_config_set_write_increment(&cfg, true);
62     channel_config_set_dreq(&cfg, DREQ_XIP_STREAM);
63     dma_channel_configure(
64         dma_chan,
65         &cfg,
66         (void *) buf,           // Write addr
67         (const void *) XIP_AUX_BASE, // Read addr
68         count_of(random_test_data), // Transfer count
69         true                   // Start immediately!
70     );
```

4.4.4. Performance Counters

The XIP subsystem provides two performance counters. These are 32 bits in size, saturate upon reaching `0xffffffff`, and are cleared by writing any value. They count:

1. The total number of XIP accesses, to any alias
2. The number of XIP accesses which resulted in a cache hit

This provides a way to profile the cache hit rate for common use cases.

4.4.5. List of XIP_CTRL Registers

The XIP control registers start at a base address of `0x400c8000` (defined as `XIP_CTRL_BASE` in SDK).

Table 440. List of XIP registers

Offset	Name	Info
0x00	CTRL	Cache control register. Read-only from a Non-secure context.
0x08	STAT	
0x0c	CTR_HIT	Cache Hit counter
0x10	CTR_ACC	Cache Access counter
0x14	STREAM_ADDR	FIFO stream address
0x18	STREAM_CTR	FIFO stream control
0x1c	STREAM_FIFO	FIFO stream data

XIP: CTRL Register

Offset: 0x00

Description

Cache control register. Read-only from a Non-secure context.

Table 441. CTRL Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	WRITABLE_M1	<p>If 1, enable writes to XIP memory window 1 (addresses 0x11000000 through 0x11fffff, and their uncached mirrors). If 0, this region is read-only.</p> <p>XIP memory is read-only by default. This bit must be set to enable writes if a RAM device is attached on QSPI chip select 1.</p> <p>The default read-only behaviour avoids two issues with writing to a read-only QSPI device (e.g. flash). First, a write will initially appear to succeed due to caching, but the data will eventually be lost when the written line is evicted, causing unpredictable behaviour.</p> <p>Second, when a written line is evicted, it will cause a write command to be issued to the flash, which can break the flash out of its continuous read mode. After this point, flash reads will return garbage. This is a security concern, as it allows Non-secure software to break Secure flash reads if it has permission to write to any flash address.</p> <p>Note the read-only behaviour is implemented by downgrading writes to reads, so writes will still cause allocation of an address, but have no other effect.</p>	RW	0x0

Bits	Name	Description	Type	Reset
10	WRITABLE_M0	<p>If 1, enable writes to XIP memory window 0 (addresses 0x10000000 through 0x10fffff, and their uncached mirrors). If 0, this region is read-only.</p> <p>XIP memory is read-only by default. This bit must be set to enable writes if a RAM device is attached on QSPI chip select 0.</p> <p>The default read-only behaviour avoids two issues with writing to a read-only QSPI device (e.g. flash). First, a write will initially appear to succeed due to caching, but the data will eventually be lost when the written line is evicted, causing unpredictable behaviour.</p> <p>Second, when a written line is evicted, it will cause a write command to be issued to the flash, which can break the flash out of its continuous read mode. After this point, flash reads will return garbage. This is a security concern, as it allows Non-secure software to break Secure flash reads if it has permission to write to any flash address.</p> <p>Note the read-only behaviour is implemented by downgrading writes to reads, so writes will still cause allocation of an address, but have no other effect.</p>	RW	0x0
9	SPLIT_WAYS	<p>When 1, route all cached+Secure accesses to way 0 of the cache, and route all cached+Non-secure accesses to way 1 of the cache.</p> <p>This partitions the cache into two half-sized direct-mapped regions, such that Non-secure code can not observe cache line state changes caused by Secure execution.</p> <p>A full cache flush is required when changing the value of SPLIT_WAYS. The flush should be performed whilst SPLIT_WAYS is 0, so that both cache ways are accessible for invalidation.</p>	RW	0x0
8	MAINT_NONSEC	<p>When 0, Non-secure accesses to the cache maintenance address window (addr[27] == 1, addr[26] == 0) will generate a bus error. When 1, Non-secure accesses can perform cache maintenance operations by writing to the cache maintenance address window.</p> <p>Cache maintenance operations may be used to corrupt Secure data by invalidating cache lines inappropriately, or map Secure content into a Non-secure region by pinning cache lines. Therefore this bit should generally be set to 0, unless Secure code is not using the cache.</p> <p>Care should also be taken to clear the cache data memory and tag memory before granting maintenance operations to Non-secure code.</p>	RW	0x0

Bits	Name	Description	Type	Reset
7	NO_UNTRANSLATED_NONSEC	When 1, Non-secure accesses to the uncached, untranslated window (addr[27:26] == 3) will generate a bus error.	RW	0x1
6	NO_UNTRANSLATED_SEC	When 1, Secure accesses to the uncached, untranslated window (addr[27:26] == 3) will generate a bus error.	RW	0x0
5	NO_UNCACHED_NONSEC	When 1, Non-secure accesses to the uncached window (addr[27:26] == 1) will generate a bus error. This may reduce the number of SAU/MPU/PMP regions required to protect flash contents. Note this does not disable access to the uncached, untranslated window – see NO_UNTRANSLATED_SEC.	RW	0x0
4	NO_UNCACHED_SEC	When 1, Secure accesses to the uncached window (addr[27:26] == 1) will generate a bus error. This may reduce the number of SAU/MPU/PMP regions required to protect flash contents. Note this does not disable access to the uncached, untranslated window – see NO_UNTRANSLATED_SEC.	RW	0x0
3	POWER_DOWN	When 1, the cache memories are powered down. They retain state, but can not be accessed. This reduces static power dissipation. Writing 1 to this bit forces CTRL_EN_SECURE and CTRL_EN_NONSECURE to 0, i.e. the cache cannot be enabled when powered down.	RW	0x0
2	Reserved.	-	-	-
1	EN_NONSECURE	When 1, enable the cache for Non-secure accesses. When enabled, Non-secure XIP accesses to the cached (addr[26] == 0) window will query the cache, and QSPI accesses are performed only if the requested data is not present. When disabled, Secure access ignore the cache contents, and always access the QSPI interface. Accesses to the uncached (addr[26] == 1) window will never query the cache, irrespective of this bit.	RW	0x1
0	EN_SECURE	When 1, enable the cache for Secure accesses. When enabled, Secure XIP accesses to the cached (addr[26] == 0) window will query the cache, and QSPI accesses are performed only if the requested data is not present. When disabled, Secure access ignore the cache contents, and always access the QSPI interface. Accesses to the uncached (addr[26] == 1) window will never query the cache, irrespective of this bit. There is no cache-as-SRAM address window. Cache lines are allocated for SRAM-like use by individually pinning them, and keeping the cache enabled.	RW	0x1

XIP: STAT Register

Offset: 0x08

Table 442. STAT Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	FIFO_FULL	When 1, indicates the XIP streaming FIFO is completely full. The streaming FIFO is 2 entries deep, so the full and empty flag allow its level to be ascertained.	RO	0x0
1	FIFO_EMPTY	When 1, indicates the XIP streaming FIFO is completely empty.	RO	0x1
0	Reserved.	-	-	-

XIP: CTR_HIT Register

Offset: 0x0c

Description

Cache Hit counter

Table 443. CTR_HIT Register

Bits	Description	Type	Reset
31:0	A 32 bit saturating counter that increments upon each cache hit, i.e. when an XIP access is serviced directly from cached data. Write any value to clear.	WC	0x00000000

XIP: CTR_ACC Register

Offset: 0x10

Description

Cache Access counter

Table 444. CTR_ACC Register

Bits	Description	Type	Reset
31:0	A 32 bit saturating counter that increments upon each XIP access, whether the cache is hit or not. This includes noncacheable accesses. Write any value to clear.	WC	0x00000000

XIP: STREAM_ADDR Register

Offset: 0x14

Description

FIFO stream address

Table 445. STREAM_ADDR Register

Bits	Description	Type	Reset
31:2	The address of the next word to be streamed from flash to the streaming FIFO. Increments automatically after each flash access. Write the initial access address here before starting a streaming read.	RW	0x00000000
1:0	Reserved.	-	-

XIP: STREAM_CTR Register

Offset: 0x18

Description

FIFO stream control

Table 446.
STREAM_CTR Register

Bits	Description	Type	Reset
31:22	Reserved.	-	-
21:0	Write a nonzero value to start a streaming read. This will then progress in the background, using flash idle cycles to transfer a linear data block from flash to the streaming FIFO. Decrement automatically (1 at a time) as the stream progresses, and halts on reaching 0. Write 0 to halt an in-progress stream, and discard any in-flight read, so that a new stream can immediately be started (after draining the FIFO and reinitialising STREAM_ADDR)	RW	0x000000

XIP: STREAM_FIFO Register

Offset: 0x1c

Description

FIFO stream data

Table 447.
STREAM_FIFO Register

Bits	Description	Type	Reset
31:0	Streamed data is buffered here, for retrieval by the system DMA. This FIFO can also be accessed via the XIP_AUX slave, to avoid exposing the DMA to bus stalls caused by other XIP traffic.	RF	0x00000000

4.4.6. List of XIP_AUX Registers

The [XIP_AUX](#) port provides fast AHB access to the streaming FIFO and the QMI Direct Mode FIFOs, to reduce the cost of DMA access to these FIFOs.

Table 448. List of XIP_AUX registers

Offset	Name	Info
0x0	STREAM	Read the XIP stream FIFO (fast bus access to XIP_CTRL_STREAM_FIFO)
0x4	QMI_DIRECT_TX	Write to the QMI direct-mode TX FIFO (fast bus access to QMI_DIRECT_TX)
0x8	QMI_DIRECT_RX	Read from the QMI direct-mode RX FIFO (fast bus access to QMI_DIRECT_RX)

XIP_AUX: STREAM Register

Offset: 0x0

Table 449. STREAM Register

Bits	Description	Type	Reset
31:0	Read the XIP stream FIFO (fast bus access to XIP_CTRL_STREAM_FIFO)	RF	0x00000000

XIP_AUX: QMI_DIRECT_TX Register

Offset: 0x4

Description

Write to the QMI direct-mode TX FIFO (fast bus access to QMI_DIRECT_TX)

Table 450.
QMI_DIRECT_TX
Register

Bits	Name	Description	Type	Reset
31:21	Reserved.	-	-	-
20	NOPUSH	<p>Inhibit the RX FIFO push that would correspond to this TX FIFO entry.</p> <p>Useful to avoid garbage appearing in the RX FIFO when pushing the command at the beginning of a SPI transfer.</p>	WF	0x0
19	OE	<p>Output enable (active-high). For single width (SPI), this field is ignored, and SD0 is always set to output, with SD1 always set to input.</p> <p>For dual and quad width (DSPI/QSPI), this sets whether the relevant SDx pads are set to output whilst transferring this FIFO record. In this case the command/address should have OE set, and the data transfer should have OE set or clear depending on the direction of the transfer.</p>	WF	0x0
18	DWIDTH	Data width. If 0, hardware will transmit the 8 LSBs of the DIRECT_TX DATA field, and return an 8-bit value in the 8 LSBs of DIRECT_RX. If 1, the full 16-bit width is used. 8-bit and 16-bit transfers can be mixed freely.	WF	0x0
17:16	IWIDTH	<p>Configure whether this FIFO record is transferred with single/dual/quad interface width (0/1/2). Different widths can be mixed freely.</p> <p>0x0 → Single width</p> <p>0x1 → Dual width</p> <p>0x2 → Quad width</p>	WF	0x0
15:0	DATA	<p>Data pushed here will be clocked out falling edges of SCK (or before the very first rising edge of SCK, if this is the first pulse). For each byte clocked out, the interface will simultaneously sample one byte, on rising edges of SCK, and push this to the DIRECT_RX FIFO.</p> <p>For 16-bit data, the least-significant byte is transmitted first.</p>	WF	0x0000

XIP_AUX: QMI_DIRECT_RX Register

Offset: 0x8

Description

Read from the QMI direct-mode RX FIFO (fast bus access to QMI_DIRECT_RX)

Table 451.
QMI_DIRECT_RX
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>With each byte clocked out on the serial interface, one byte will simultaneously be clocked in, and will appear in this FIFO. The serial interface will stall when this FIFO is full, to avoid dropping data.</p> <p>When 16-bit data is pushed into the TX FIFO, the corresponding RX FIFO push will also contain 16 bits of data. The least-significant byte is the first one received.</p>	RF	0x0000

4.5. OTP

RP2350 contains 8 KiB of one-time-programmable storage (OTP), which stores:

- Manufacturing information such as unique device ID
- Boot configuration such as non-default crystal oscillator frequency
- Public key fingerprint(s) for boot signature enforcement
- Symmetric keys for decryption of external flash contents into SRAM
- User-defined contents, including bootable program images ([Section 5.2.6](#))

The OTP storage is structured as 4096×24 -bit rows. Each row contains 16 bits of data and 8 bits of parity information, providing 8 KiB of data storage. OTP bit cells are initially **0** and can be programmed to **1**. However, they cannot be cleared back to **0** under any circumstance. This ensures that security-critical flags, such as debug disables, are physically impossible to clear once set. However, you must also take care to program the correct values.

For more information about the OTP subsystem, see [Chapter 13](#).

Chapter 5. Bootrom

The bootrom size is limited to 32 KiB. It contains code for both Arm and RISC-V:

- Processor core 0 initial boot sequence
- Processor core 1 low power wait and launch protocol
- Support for booting signed/unsigned/hashed applications
- Support for dividing flash into multiple separately attributed partitions
- Routines for programming and manipulating the external flash, with access permissions
- Routines for programming and accessing OTP
- Support for calling certain bootrom routines from Non-secure (NS) Arm code, with individual permission enables controlled by the Secure code
- USB MSC class-compliant bootloader with [UF2](#) support for downloading code/data to FLASH or RAM, including version upgrades and downgrades
- USB PICOBLOCK bootloader interface for advanced management
- UART bootloader

Bootrom Source Code

The bootrom source will be made public on GitHub at launch.

5.1. Bootrom Concepts

5.1.1. Image Definitions

Image definitions, referred to throughout this chapter as [IMAGE_DEF](#)s, provide metadata for executable images or data images, for example, so that the bootrom knows where to enter an executable. An image is any contiguous blob of data, either stored in flash/OTP or preloaded into RAM, with a valid [IMAGE_DEF](#).

The [IMAGE_DEF](#) metadata can be used to determine which of multiple executable images to boot, how to boot it, whether it is to be run in-place or loaded to RAM first, and so on. [IMAGE_DEF](#)s can contain version numbers, hashes and signatures for the image contents.

Additionally, for [IMAGE_DEF](#)s for executable images, the following can be specified:

- Target chip (RP2040 / RP2350), target architecture (Arm Secure, Arm Non-secure, RISC-V) (see [Section 5.10.3.1](#))
- Entry point. (see [Section 5.10.3.4](#)).
- Vector Table pointer (on Arm) (see [Section 5.10.3.3](#)).
- Anti-rollback information (see [Section 5.1.6](#))
- A "Load Map" which defines zeroing of memory, copying of code/data into RAM (by the bootrom) and what contents are hashed for hash or signature checking (see [Section 5.10.3.2](#)).
- A window roll (see [Section 5.1.17](#))

An [IMAGE_DEF](#) is required for any binary that is launched by the bootrom; see [Section 5.1.4](#) for information on how the [IMAGE_DEF](#) is stored within the binary.

Details of the [IMAGE_DEF](#) format can be found in [Section 5.10.3](#). See [Section 5.10.5](#) for the minimum requirements for a

bootable image.

5.1.2. Partition Tables

Partition tables (aka `PARTITION_TABLE`s) are used to both divide the flash into up to sixteen distinct regions (partitions) and specify various attributes for those partitions.

The partition attributes include:

- Start/End offsets within the logical 16 MiB address space of the first flash window. These offsets are specified in multiples of a flash sector (4 KiB).
- Access permissions for the partition; read/write for each of Secure, Non-secure and Boot Loader access.
- Bootability of the partition under Arm or RISC-V.
- Information on what UF2 family-ids may be dropped into the partition via the UF2 bootloader.
- An optional 64-bit identifier.
- An optional name.
- Information to group partitions together (see [Section 5.1.8](#) and [Section 5.1.9](#)).

The partition table includes default permissions for unpartitioned space.

Partition tables can also be versioned, hashed and signed.

Details of the `PARTITION_TABLE` format can be found in [Section 5.10.4](#).

See [Section 5.1.4](#) for information on how the `PARTITION_TABLE` is discovered during flash boot.

5.1.3. Flash Permissions

As mentioned in [Section 5.1.2](#), permissions for each partition are stored in the partition table, as well as permissions for the un-partitioned space.

Separate read/write permissions are specified for each of Secure, Non-secure and Boot Loader access. "Boot Loader" permissions control what can be done via the PICOBLOCK interface or where UF2 images can be downloaded.

Because flash permissions may be changed dynamically at runtime, the resident part of the partition table may be modified at runtime to add permissions for other areas of flash. There is no bootrom API for this however the in memory partition table format is documented and a pointer to it is available to secure code. The SDK however provides APIs to wrap this functionality.

5.1.4. Blocks And Block Loops

5.1.4.1. Blocks

`IMAGE_DEFS` and `PARTITION_TABLE`s are types of Block. A "Block" is a recognisable, self-checking data structure containing one or distinct data "Items". The type of a "Block" is determined by the type of its first Item.

Blocks are designed to be backwards and forwards compatible; Individual Item types will not be changed in the future in ways that would cause existing code to misinterpret the data. Consumers of Blocks can safely skip Items they do not understand.

Valid Blocks are recognised by a 4 byte header, 4 byte footer, and the linked list of items inside the Block.

Blocks are found in an arbitrary region of memory, by searching for a "Block Loop" within the first 4 kiB of that given region (for flash boot) or the entire region (for RAM/OTP image boots).

5.1.4.2. Block Loops

A Block Loop is a linked loop of Blocks. Each Block has a relative pointer to the next Block, and the last Block must link to the first. A single Block can form a Block Loop by linking back to itself, with a relative pointer of 0.

The purpose of a Block Loop is both to collect multiple Blocks belonging to the same entity, but also to make locating Blocks more robust/efficient.

The first Block in a Block Loop must be at the lowest memory address, and is always searched for within the first 4 kB of data. For example in an executable image, there must be a Block within the first 4 kB of the image, allowing both flexibility for different languages' memory layouts, but also for efficient location of [IMAGE_DEF](#) Blocks.

Block Loops may contain multiple Blocks because:

1. Signing an image will duplicate the existing [IMAGE_DEF](#) and add another (bigger) one with signature information.
2. An image may contain multiple [IMAGE_DEF](#)s, e.g. with different signing keys.
3. Placing a Block at both the beginning and end of an image, is a cheap way of making sure that the image has not been partially overwritten. Of course hashing or signing this image is better.
4. An image might contain code for both Arm and RISC-V and have [IMAGE_DEF](#)s for both.
5. [PARTITION_TABLES](#) and [IMAGE_DEF](#)s may be mixed in the same Block Loop (see [Section 5.1.14](#)).

If a Block Loop contains multiple [IMAGE_DEF](#)s or [PARTITION_TABLES](#), then the last one seen wins, except for [IMAGE_DEF](#)s where an [IMAGE_DEF](#) for the current architecture always beats one for a different architecture.

5.1.5. Versioning

Any Block may contain version information. Version Information is of the form [\(Rollback\).Major.Minor](#); each field is 16 bits.

The Rollback version is 0 if not present, and is only used in [IMAGE_DEF](#)s.

The version number is used to pick the latest version between to [IMAGE_DEF](#)s or two [PARTITION_TABLES](#) (see [Section 5.1.8](#)).

The Rollback version is additionally utilized on a secure RP2350 in the mechanism for preventing running potentially vulnerable back-level code (see [Section 5.1.6](#)).

See [Section 5.10.2.1](#) for details on the VERSION item.

5.1.6. Anti-Rollback

[IMAGE_DEF](#) version information is of the form [\(Rollback\).Major.Minor](#). If a Rollback version is present, it is accompanied by a list of OTP rows that are to be used to form a *thermometer* indicating the minimum Rollback version that may be run.

An [IMAGE_DEF](#) is not valid if there aren't enough OTP rows specified to hold the Rollback version number.

On a secure RP2350 if an [IMAGE_DEF](#) is successfully launched which has a Rollback version, then that version will be written to OTP, so that only that Rollback version or higher may be executed in the future.

The [BOOT_FLAGS0.ROLLBACK_REQUIRED](#) flag may be used to *require* an [IMAGE_DEF](#) have a Rollback version on a secure RP2350. This flag will be automatically set after writing the first rollback version to OTP.

NOTE

An `IMAGE_DEF` with a rollback version of 0 will not automatically set the `BOOT_FLAGS0.ROLLBACK_REQUIRED` flag, so it is recommended that the minimum rollback version used is 1, unless the `BOOT_FLAGS0.ROLLBACK_REQUIRED` flag will be manually set during provisioning.

5.1.7. Packaged Binaries

The most common case for signed binaries in flash on a secure RP2350 is for them to be loaded from flash into RAM first, have the signature verified there, and then executed from RAM. Since this loading happens before the executable runs, it is specified by a `LOAD_MAP` item within the `IMAGE_DEF`.

A "Packaged Binary" is a term used to describe such a binary which is stored in flash, but runs entirely from RAM. The "Package" terms refers to the fact that the binary is likely compiled to be run from RAM like a RAM only binary, but is packaged (and potentially signed) by tooling with a new `IMAGE_DEF` describing the load process (and the signature).

5.1.8. A/B Partitions

A pair of partitions may be grouped into an "A/B" pair. The purpose of having two logically grouped partitions is to have the current executable image (or piece of data), and write the newer version of the data into the other partition, before safely moving to that.

Use of A/B partitions can prevent failures during software update from making the RP2350 un-bootable. A/B partitions are used:

- When booting; the partition (out of A/B) with a valid `IMAGE_DEF` with the higher version will generally be booted. See [Section 5.1.15](#) for scenarios where this is not the case.
- When dragging a UF2 onto the `BOOTSEL` USB drive, the UF2 will be targeted to the A/B partition which is not currently the one which boots. See [Section 5.1.9.1](#) for scenarios where this is not the case.

5.1.9. UF2 Targeting

[Section 5.5](#) describes the USB Mass Storage Drive, and the ability to download UF2 files to that drive to store and/or execute code/data on the RP2350.

Since RP2350 supports multiple processor architectures, and partition tables with multiple partitions, some information on the device must be used to determine what to do with *flash-targeting* UF2 (as opposed to simply writing all flash data to the start of flash).

UF2 supports a 32 bit family ID embedded in the file for the purposes of firmware recognising data targeting it. The RP2350 bootrom defines a number of UF2 family IDs `rp2040`, `rp2350-arm-s`, `rp2350-arm-ns`, `rp2350-riscv`, `data` and `absolute`. The user can also define their own family IDs for more refined targeting.

The UF2 family ID is used as follows:

- A UF2 with the `absolute` family ID is downloaded without regard to partition boundaries. A partition table (if present) or OTP configuration can define whether `absolute` family ID downloads are allowed, and download to the start of flash. The default factory settings allow for `absolute` family ID downloads
- Any other family IDs target download to a single partition if there is a partition table present; if there is no partition table present then the `data`, `rp2350-arm-s` (if Arm architecture is enabled) and `rp2350-riscv` (if RISC-V architecture is enabled) family IDs are allowed by default, and the UF2 is downloaded to the start of flash.
- Each partition indicates which family IDs it supports
- With A/B partitions; the A partition indicates the family IDs supported, and the UF2 goes to the partition that isn't currently the boot choice. Note this choice is determined at runtime based on the current state of flash

- Further refinement of A/B is allowed to support other A/B partitions containing data/executables used by the main partitions; see [Section 5.1.9.1](#) for more information.

For details of the exact rules used when picking a UF2 target partition see [Section 5.5.3](#).

5.1.9.1. Owned Partitions

It is often the case that an executable may use data from another partition (e.g. WiFi firmware). In the case of an executable housed in A/B partitions, it may be desirable to associate other A/B partitions C/D with the primary A/B partitions, such that the data in partition C is used for executable in partition A, and the data in partition D is used for the executable in partition B.

In this scenario A can be marked as the owner for C, and this will affect UF2 image whose family ID targets C & D.

By default, when B is would be the target partition for a UF2 with the A/B compatible family ID, then D will be the target for a UF2 with the C/D compatible family ID. However, a flag in the partition table can be used to flip this mapping.

NOTE

Owned partitions are most commonly used when the data within them does not contain an `IMAGE_DATA` Block. Using an `IMAGE_DEF` even with data, allows regular "top-level" A/B decisions to be made, since the bootrom can determine which alternative has the higher version.

5.1.10. Hashing and Signing

Any block may be hashed or signed. If a block is hashed only, then the hash value is stored in the block (see `<item-HASH_VALUE>`), and the hash calculated at runtime will be compared against the stored hash to determine if the block is valid.

On a secure RP2350 a hash is not sufficient – a signature is required – so the signed hash value is stored in the block instead. The signature of the hash is checked at runtime against the runtime hash value using a `secp256k1` public key. The public key is also stored in the block in a `SIGNATURE` item (see [Section 5.10.2.4](#)). When we say a secure RP2350, we refer to a device where `CRIT1.SECURE_BOOT_ENABLE` is programmed to 1.

The public key must match one of the up to four valid boot keys defined by the user (see `BOOT_FLAGS1.KEY_VALID` and `BOOTKEY0_0` onwards).

The input to/type of the hash is defined by a `HASH_DEF` item (see [Section 5.10.2.2](#)), which can include contents from the block itself. For a signature, it **must** contain all contents of the block up to the `SIGNATURE` item.

Additionally, for `IMAGE_DEFS` all the runtime contents defined by `LOAD_MAP` item (see [Section 5.10.3.2](#)) are hashed. The common use case here is that the `LOAD_MAP` defines code to be copied from flash to RAM, which is hashed *after* copying – it is dangerous to verify flash contents before copying, because someone with physical access to the QSPI bus could swap the flash contents in between the check and the copy. However, other storage address range can be hashed by setting the runtime address and storage addresses to the same value in the `LOAD_MAP`.

On a secure RP2350 with the `BOOT_FLAGS0.SECURE_PARTITION_TABLE` flag set, the bootrom will ignore any `PARTITION_TABLE` that is not correctly signed.

5.1.11. Flash `IMAGE_DEF` Boot

In the simplest form of flash boot, there is a Block Loop starting within the first 4 kiB of flash which that contains an `IMAGE_DEF` (and no `PARTITION_TABLE`).

This is just "please boot the image in flash!"; there is no opportunity for A/B, multiple versions or anything else.

If the `IMAGE_DEF` is executable and valid (and signed if necessary) then it will be booted.

For the unsigned case, this can be as little as a 20-byte marker anywhere in the first 4 kiB of the image: see [Section 5.10.5](#).

5.1.12. Boot Slots

Similarly to how a choice can be made between `IMAGE_DEF`s in A/B partitions, a choice can be made between `PARTITION_TABLE`s in Boot Slots. This allows for versioning Partition Tables, drag and drop of UF2s containing partition tables, etc. in the same way as is done for images.

The two boot slots; Slot 0 is the first `n` 4 kiB sectors of flash, and Slot 1 is the second `n` 4 kiB sectors of flash. `n` defaults to 1, i.e. each slot is 4 kiB in size, however this value can be overridden by specifying a value in `FLASH_PARTITION_SLOT_SIZE` and then setting `BOOT_FLAGS0.OVERRIDE_FLASH_PARTITION_SLOT_SIZE`.

If there is a Block Loop in Slot 0, then its first Block must be found within the first 4 kiB of Slot 0.

If there is a Block Loop in Slot 1, then its first Block must be found within the first 4 kiB of Slot 1.

Note in the simple `IMAGE_DEF`-only scenario in the previous section, Slot 1 is not considered at all. The reasons for this are described in more detail in [Section 5.1.14](#).

5.1.13. Flash `PARTITION_TABLE` Boot

Another common boot scenario is to just have a `PARTITION_TABLE` in the Block Loop in Slot 0 (and possibly Slot 1).

In the case there are valid `PARTITION_TABLE`s in both Slots, the (valid, and signed if necessary) Partition Table with the higher version number is generally chosen. See [Section 5.1.15](#) for exceptions to this rule.

To avoid an unnecessary search for the simple case where there is just a `PARTITION_TABLE` in Slot 0, the `PARTITION_TABLE` can mark itself as a *singleton*. This is an optimisation, and is not used if the Partition Table in Slot 1 is invalid.

In the `PARTITION_TABLE` only scenario, once a Partition Table from one of the Slots has been chosen, each partition within the Partition Table is searched, in order of ascending partition number, for Block Loops containing `IMAGE_DEF`s that can be booted. Again, as an optimisation, partitions can be marked non-bootable to exclude them from this search on either processor architecture.

5.1.14. Flash `IMAGE_DEF` + `PARTITION_TABLE` Boot

The third and final case involves at least one `IMAGE_DEF` and at least one `PARTITION_TABLE` in the Block Loop. For predictability, since the presence of both Block types in the same Block List changes behaviour, the presence of a recognisable `IMAGE_DEF` and `PARTITION_TABLE` is all that is required; they do not need to be valid, or correctly signed if required.

The two common cases where you would combine an `IMAGE_DEF` and a `PARTITION_TABLE` in the same Block Loop are:

1. You are only using the `PARTITION_TABLE` for flash permissions. You want to load it and boot them boot the associated `IMAGE_DEF`, rather than looking for an `IMAGE_DEF` within one of the partitions
2. The `IMAGE_DEF` is a bootloader stored in the Slot with the Partition Table. In this case, once again the Partition Table will be loaded, and the associated image entered, however the entered image will likely pick a partition from the Partition Table, and launch an image from there.

NOTE

When `PARTITION_TABLE`s are present, the Partition Table metadata determines which of Slot 0 and Slot 1 to use, and the `IMAGE_DEF` does not. Conversely, when there is no `PARTITION_TABLE` present, as mentioned above, there is considered only to be Slot 0, so there is no choice to make.

5.1.15. Flash Update / Version Downgrade

Normally, the choice of Slot 0 vs Slot 1 and Partition A vs Partition B, are made based on the version of the valid `PARTITION_TABLE` or `IMAGE_DEF` in those Slots or Partitions respectively, with the highest version winning.

It is however perfectly valid to wish to downgrade to a lower-versioned `IMAGE_DEF` when using A/B partitions, provided this does not violate anti-rollback rules on a secure RP2350.

Downloading the `IMAGE_DEF` partition into the non-currently-booting partition, and doing a normal reboot will obviously not work in this case, as the newly downloaded image has a lower version.

For this reason, a special type of flash boot called `FLASH_UPDATE` boot can be enabled by passing the correct flag through the watchdog scratch registers, along with a pointer to the start of the region of flash that has just been updated. It is used automatically when dropping UF2s on the USB Mass Storage Drive, but can also be invoked programmatically via the `reboot()` API (see [Section 5.4.10.1](#)).

The so-marked region is treated specially during this `FLASH_UPDATE` boot, and a `PARTITION_TABLE` in a Slot, or `IMAGE_DEF` in a Partition, will be chosen for boot irrespective of version, if the start of the region is the start of the respective Slot or Partition.

In order for the downgrade to persist, the first sector of the previously booting Slot or Partition is erased so that the newly installed `PARTITION_TABLE` or `IMAGE_DEF` will continue to be chosen on subsequent boots. This erase is performed as follows:

- When a `PARTITION_TABLE` is valid (and correctly signed if necessary), and its Slot is chosen for boot, the first sector of the other Slot is erased.
- When a valid (and correctly signed if necessary) `IMAGE_DEF` is launched, the first sector of the other Partition is erased.
- On explicit request by the launched `IMAGE_DEF`, the first sector of the other Partition is erased. This is an alternative to the standard behaviour in the previous bullet, and is selected by a flag in the `IMAGE_DEF`. This feature is described in [Section 5.1.16](#).

NOTE

Flash Update / Version Downgrade have no effect when using a single Slot, or standalone (non A/B) Partitions

5.1.16. Try Before You Buy

Try Before You Buy (TBYB) is an `IMAGE_DEF` only feature that allows for a completely safe cycle of version upgrade:

1. Executable image is running from say Partition B.
2. New image is downloaded into Partition A.
3. On download completion, a `FLASH_UPDATE` reboot is performed for Partition A.
4. If the new image fails validation/signature then the old image in Partition B will be used on next boot, recovering from the failed upgrade.
5. If the new image is valid (and correctly signed if necessary), it is entered under a watchdog timer, and has 16.7 seconds to mark itself OK via the `explicit_buy()` function.

- If the image calls back, it becomes the preferred image for subsequent boots.
- If it fails to call back within the allotted time, then the system reboots, and will continue to boot partition B (containing the original image) as partition A is still marked as a tentative TBYB image.

The watchdog timeout is fixed at 16.7 seconds (24-bit count on a 1-microsecond timebase) – this can be shortened after entering the target image, for example if it only needs a few hundred milliseconds for its self-test routine, or can be extended by reloading the watchdog counter, at the risk of getting stuck in the tentative image.

5.1.17. Address Translation

For simplicity, references to flash data are assumed to be compiled for an image that starts at `0x10000000`. Therefore, when launching an image from a Partition which does not start at `0x10000000`, address translation is performed to make the start of the Partition appear at `0x10000000`. The size of the region exposed between `0x10000000` and `0x11000000` defaults to the same as the size of the partition, and a bus fault will occur if the image tries to access beyond the bounds of the partition it booted from.

It may be however, that the image is compiled to run at some other address. The bootrom allows for runtime values of `0x10000000`, `0x10400000`, `0x10800000` and `0x10c00000` – which may be specified within the `IMAGE_DEF`.

Custom address translation is defined by the `ROLLING_WINDOW_DELTA` item (see [Section 5.10.3.5](#)). The above four values translate to a `ROLLING_WINDOW_DELTA` of `0`, `-0x400000`, `-0x800000`, `-0xc00000`, which indicate what how far into the image `0x10000000` is.

Positive `ROLLING_WINDOW_DELTA` values in multiples of 4 KiB are also allowed, which again indicate how far into the image `0x10000000` is; i.e. a `ROLLING_WINDOW_DELTA` of `0x1000` will set up address translation such that the image data starting at offset `0x1000` is mapped to `0x10000000` at runtime. The first 4 KiB of the image can no longer be accessed, except via the untranslated XIP window, which defaults to Secure access only.

NOTE

Because address translation of the `0x10000000 → 0x11000000` and `0x11000000 → 0x12000000` regions are independent, it is only possible to boot from partitions which are entirely contained within the first 16M of flash.

This translation is enabled by the address translation hardware in the QMI – see [Section 12.14.4](#)

5.1.18. Auto Architecture Switching

If a valid and correctly signed `IMAGE_DEF` is found for the non-current architecture (i.e. RISC-V when booted in Arm mode, or Arm when booted in RISC-V), and the other architecture is not disabled, then the bootrom will automatically reboot into that architecture, and the image should run.

This feature can be disabled by setting the `BOOT_FLAGS0.DISABLE_AUTO_SWITCH_ARCH`.

NOTE

Generally it is preferable to have an executable image for the current architecture if present boot instead of one for the other architecture. For this reason, images for different architectures are generally kept in different Partitions, with the Partitions marked as non-bootable for one architecture or the other, such that they are skipped-over during boot based on the current architecture.

See [Section 3.9](#) for details of hardware support for architecture switching.

5.2. Example Boot Scenarios

This section will contain detailed examples, of different Partition Table layouts, OTP settings etc.

5.2.1. Secure Booting

To enable secure booting on RP2350, you must do the following things:

1. Set the SHA256 hashes of the bootkeys you will be using in `BOOTKEY0_0` onwards
2. Set the bits in `BOOT_FLAGS1.KEY_VALID` for the keys you will be using

Then you can set `CRIT1.SECURE_BOOT_ENABLE` to turn on secure booting. It is also recommended to set the bits in `BOOT_FLAGS1.KEY_INVALID` for the keys that you will not be using, to prevent a malicious actor adding more `BOOTKEYS` and using them.

The typical layout for a signed image will be a Block within the first sector of the image which links to a second Block at the end of the image (see [Section 5.2.8](#)). The first Block can contain a single ignored item - it's only purpose is to create a Block Loop with the second Block. The second Block must be an `IMAGE_DEF`, with a `LOAD_MAP`, `HASH_DEF` and `SIGNATURE` (see [Section 5.1.10](#)). The `LOAD_MAP` should cover the entire image, up to the second Block, as any code not covered by the `LOAD_MAP` will not be signed.

The `SIGNATURE` can be computed by computing the SHA256 hash of all of the data specified in the `LOAD_MAP`, along with the block words specified in the `HASH_DEF`. This hash is then signed with an ECDSA secp256k1 private key, to produce a 64 byte signature.

For secure booting, it is recommended to use packaged SRAM binaries instead of flash binaries, as the signature check is only performed at boot, so a malicious actor with physical access could replace the data on the external flash after the signature check to run unsigned code.

5.2.2. Packaged SRAM Booting

To create a packaged SRAM binary, you can take a binary compiled to run in SRAM and add a relative `LOAD_MAP` into the `IMAGE_DEF` Block, with the runtime address set to SRAM. The subsequent binary will then run both when directly loaded into SRAM, and when loaded into Flash where the Bootrom will copy it to SRAM when booting.

Alternatively you can use an absolute `LOAD_MAP`, with the `storage_address` in Flash and the `runtime_address` in SRAM, but these binaries will only run if they are stored in Flash and cannot be booted directly in SRAM for debugging.

For example, if you have a binary compiled to run at `0x2000000` of length `0x8000`, and a metadata block at the end of the binary containing the `LOAD_MAP`, then the relative `LOAD_MAP` would be

```
storage_address_rel = -0x8000 = 0xFFFF8000
runtime_address_physical = 0x20000000
size = 0x8000
```

or the absolute `LOAD_MAP` would be

```
storage_address_physical = 0x10000000
runtime_address_physical = 0x20000000
runtime_end_address_physical = 0x20008000
```

5.2.3. A/B Booting

This is a common boot scenario, to be able to update the software without overwriting it. The partition layout required is

```

Partition 0
  Accepts Families: rp2350-arm-s, rp2350-riscv
Partition 1
  Accepts Families: rp2350-arm-s, rp2350-riscv
  Link Type: A
  Link Value: 0

```

It is recommended to have the same permissions for both partitions, and for both partitions to accept the same UF2 families, as the bootrom will only look at the families from the A partition when deciding if a given A/B pair accepts a particular family, and will skip an A partition if it's B partition isn't writable.

When uploading a new UF2 to the device, it will go into whichever partition is not currently booting (starting with partition 1). The bootrom will then perform a FLASH_UPDATE boot into the new binary (see [Section 5.1.15](#))

To create this partition table using `picotool partition create`, the following json could be used as an example

```

{
  "version": [1, 0],
  "unpartitioned": {
    "families": ["absolute"],
    "permissions": {
      "secure": "rw",
      "nonsecure": "rw",
      "bootloader": "rw"
    }
  },
  "partitions": [
    {
      "name": "A",
      "id": 0,
      "start": 2,
      "size": 20,
      "families": ["rp2350-arm-s", "rp2350-riscv"],
      "permissions": {
        "secure": "rw",
        "nonsecure": "rw",
        "bootloader": "rw"
      }
    },
    {
      "name": "B",
      "id": 1,
      "start": 30,
      "size": 20,
      "families": ["rp2350-arm-s", "rp2350-riscv"],
      "permissions": {
        "secure": "rw",
        "nonsecure": "rw",
        "bootloader": "rw"
      },
      "link": ["a", 0]
    }
  ]
}

```

5.2.4. A/B Booting with Owned Partitions

The concept of Owned Partitions is for when you require separate data partitions which generally won't have a Block Loop in them, but you would like these to be associated with a specific boot partition. An example partition table for this scenario would be

```

Partition 0
  Accepts Families: rp2350-arm-s, rp2350-riscv
Partition 1
  Accepts Families: rp2350-arm-s, rp2350-riscv
  Link Type: A
  Link Value: 0
Partition 2
  Accepts Families: data
  Link Type: Owner
  Link Value: 0
  IGNORED_DURING_ARM_BOOT: true
  IGNORED_DURING_RISCV_BOOT: true
Partition 3
  Accepts Families: data
  Link Type: A
  Link Value: 2
  IGNORED_DURING_ARM_BOOT: true
  IGNORED_DURING_RISCV_BOOT: true

```

When downloading a UF2 into an Owned Partition, the bootloader will select which partition out of 2/3 it goes to based on which partition out of 0/1 is currently booting. For example, if partition 1 is currently booting (due to having a higher version than partition 0), then any UF2 downloads with the data family will go to partition 3.

NOTE

Only the `get_uf2_target_partition()` bootrom function will consider owner partitions - the `pick_ab_partition()` function will always pick solely based on the A/B partition it is passed (ie partitions 2/3 in this example)

To create this partition table using `picotool partition create`, the following json could be used as an example

```
{
  "version": [1, 0],
  "unpartitioned": {
    "families": ["absolute"],
    "permissions": {
      "secure": "rw",
      "nonsecure": "rw",
      "bootloader": "rw"
    }
  },
  "partitions": [
    {
      "name": "A",
      "id": 0,
      "start": 2,
      "size": 20,
      "families": ["rp2350-arm-s", "rp2350-riscv"],
      "permissions": {
        "secure": "rw",
        "nonsecure": "rw",
        "bootloader": "rw"
      }
    }
  ]
}
```

```

},
{
  "name": "B",
  "id": 1,
  "start": 30,
  "size": 20,
  "families": [ "rp2350-arm-s", "rp2350-riscv" ],
  "permissions": {
    "secure": "rw",
    "nonsecure": "rw",
    "bootloader": "rw"
  },
  "link": [ "a", 0 ]
},
{
  "name": "a",
  "id": 2,
  "start": 50,
  "size": 5,
  "families": [ "data" ],
  "permissions": {
    "secure": "rw",
    "nonsecure": "rw",
    "bootloader": "rw"
  },
  "link": [ "owner", 0 ],
  "not_bootable_on_arm": true,
  "not_bootable_on_riscv": true
},
{
  "name": "b",
  "id": 3,
  "start": 55,
  "size": 5,
  "families": [ "data" ],
  "permissions": {
    "secure": "rw",
    "nonsecure": "rw",
    "bootloader": "rw"
  },
  "link": [ "a", 2 ],
  "not_bootable_on_arm": true,
  "not_bootable_on_riscv": true
}
]
}

```

5.2.5. Custom Bootloader

This is for the scenario where a bootloader must be run before booting into an image. This could perform additional validation, or set up peripherals for use by the image. For this to work, there must be an [IMAGE_DEF](#) for the bootloader and a [PARTITION_TABLE](#) to define the flash layout in the same Block Loop. This is scenario 2 in [Section 5.1.14](#).

The [PARTITION_TABLE](#) can have an A/B layout, to provide A/B booting functionality for the images. Slot 0/1 booting can then be used to provide A/B booting functionality for the bootloader based on the [PARTITION_TABLE](#) version. See [Section 5.1.12](#) for more details of this, as you will likely need to increase the size of Slot 0 in order to fit the bootloader. An example flash layout would then be as follows:

```

Slot 0 (0x00000000-0x00008000)
Bootloader Image 0
Partition Table 0
Slot 1 (0x00008000-0x00010000)
Bootloader Image 1
Partition Table 1
Partition 0 (0x00010000-0x00020000)
Binary A
Partition 1 (0x00020000-0x00030000)
Link Type: A
Link Value: 0
Binary B

```

The bootloader image would have the Block Loop shown in [Section 5.2.8](#) (assuming a signed/hashed image):

1. Block in first 4 KiB (contents doesn't matter, as it will be superceded by the later `IMAGE_DEF`)
2. `PARTITION_TABLE` at end of binary
3. `IMAGE_DEF` with a `LOAD_MAP` that covers the `PARTITION_TABLE` as well, for efficient signing/hashing

For the bootloader to replicate the functionality of the bootrom, it will need to call the following functions to determine which partition to boot an image from, and then boot the chosen image:

1. `get_sys_info()` with `BOOT_INFO`, to get the `flash_update_boot_window_base`

```

uint32_t sys_info[5];
get_sys_info(sys_info, 5*4, SYS_INFO_BOOT_INFO);
uint32_t flash_update_boot_window_base = sys_info[3];

```

2. `pick_ab_partition()` with `flash_update_boot_window_base`, to pick the boot partition

```

uint8_t boot_partition = pick_ab_partition(workarea, 0xC00, 0,
flash_update_boot_window_base);

```

3. `get_partition_table_info()` with `SINGLE_PARTITION`, the chosen partition number, and `PARTITION_LOCATION_AND_FLAGS`, to get the address of the boot partition

```

uint32_t partition_info[8];
get_partition_table_info(partition_info, 8, PT_INFO_PARTITION_LOCATION_AND_FLAGS
| PT_INFO_SINGLE_PARTITION | (boot_partition << 24));
uint16_t first_sector_number = (partition_info[1]
& PICOBIN_PARTITION_LOCATION_FIRST_SECTOR_BITS)
>> PICOBIN_PARTITION_LOCATION_FIRST_SECTOR_LSB;
uint16_t last_sector_number = (partition_info[1]
& PICOBIN_PARTITION_LOCATION_LAST_SECTOR_BITS)
>> PICOBIN_PARTITION_LOCATION_LAST_SECTOR_LSB;
uint32_t data_start_addr = first_sector_number * 0x1000;
uint32_t data_end_addr = (last_sector_number + 1) * 0x1000;
uint32_t data_size = data_end_addr - data_start_addr;

```

4. `chain_image()` with `data_start_addr` and `data_size`, to boot the chosen image. The `workarea` used must not be overwritten by the image being chained into - this is mainly relevant for a Packaged Binary, as there is the possibility of the Packaged Binary being copied into SRAM and overwriting the `workarea`, which will cause undefined behaviour.

```

chain_image(
    workarea,
    0xC00,
    (XIP_BASE + data_start_addr) *
        (info.boot_type == BOOT_TYPE_FLASH_UPDATE ? -1 : 1),
    data_size
);

```

If the bootloader determines that it should not boot the image chosen by the bootrom, it can use the `get_b_partition()` function to find the other partition and boot that instead.

5.2.6. OTP Bootloader

This is an extension of the Custom Bootloader, but it will be stored in the OTP and will run in SRAM. The entire bootloader will need to fit in the OTP rows from `0x0C0` to `0xF48` to avoid interfering with other OTP functionality, giving a maximum size of 7440 bytes (2 bytes per ECC row). If some bootkeys and OTP keys are not going to be used then this region can be extended slightly on either end if required.

The OTP bootloader itself should be stored in ECC format, starting from the row set in `OTPBOOT_SRC` with size set in `OTPBOOT_LEN`. When booting it will be loaded into the address specified in `OTPBOOT_DST0` and `OTPBOOT_DST1`, which must be in the main SRAM. The bootloader must fulfil the same criteria as a standard image, with an `IMAGE_DEF` within the first sector, and must be signed if secure boot is enabled.

Once the OTP bootloader has been written, and the `OTPBOOT_SRC`, `OTPBOOT_LEN`, `OTPBOOT_DST0` and `OTPBOOT_DST1` set, OTP booting can be enabled by setting `BOOT_FLAGS0.ENABLE OTP_BOOT`. To prevent booting from flash without going through the OTP bootloader, `BOOT_FLAGS0.DISABLE_FLASH_BOOT` can also be set.

Extra care should be taken when writing an OTP bootloader, as once the ECC rows are written they cannot be modified.

5.2.7. Rollback Versions with Bootloaders

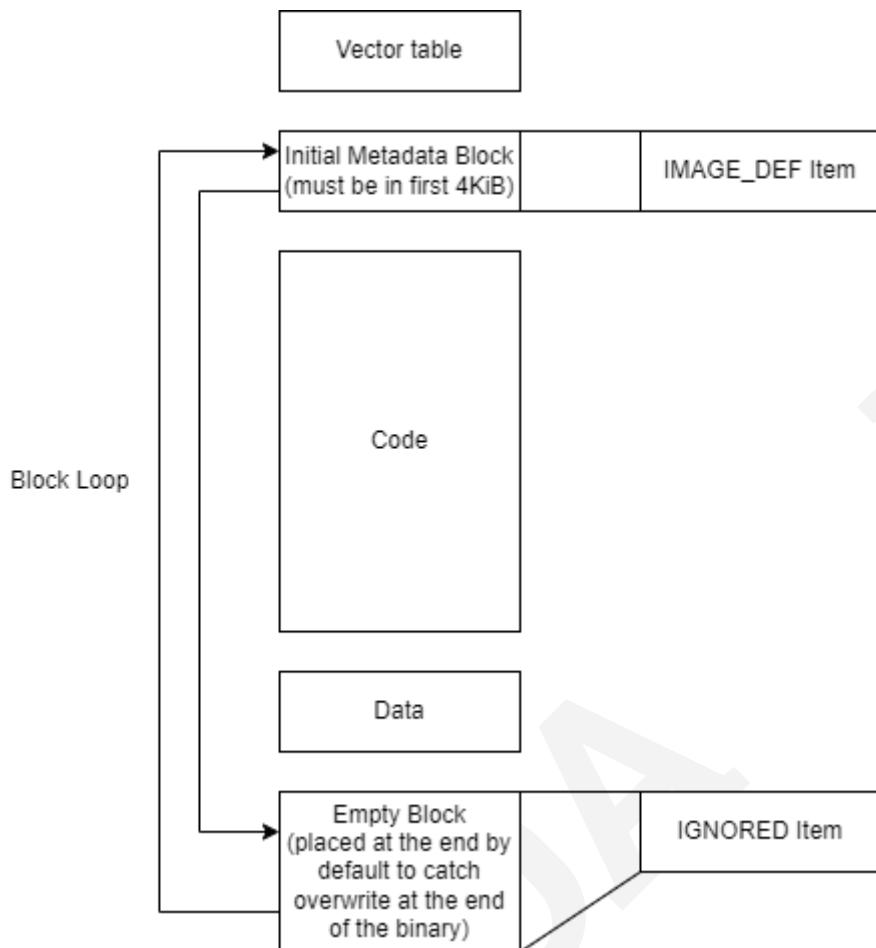
For Bootloaders that need to boot Binaries with rollback versions, you will need to have separate OTP rows for use by the Bootloader Rollback Version and the Binary Rollback Version. It is recommended to use the `DEFAULT_BOOT_VERSION0` & 1 rows for the Binary Rollback Version, and then select some other unused rows in the OTP to use for the Bootloader Rollback Version.

This allows for the Bootloader Rollback Version to be set when booting the Bootloader, which will correctly set the `BOOT_FLAGS0.ROLLBACK_REQUIRED` flag. This flag will not be set if the Bootloader does not have a rollback version (or has a rollback version of 0), as it is not set when chaining into an image.

5.2.8. Example Block Loops

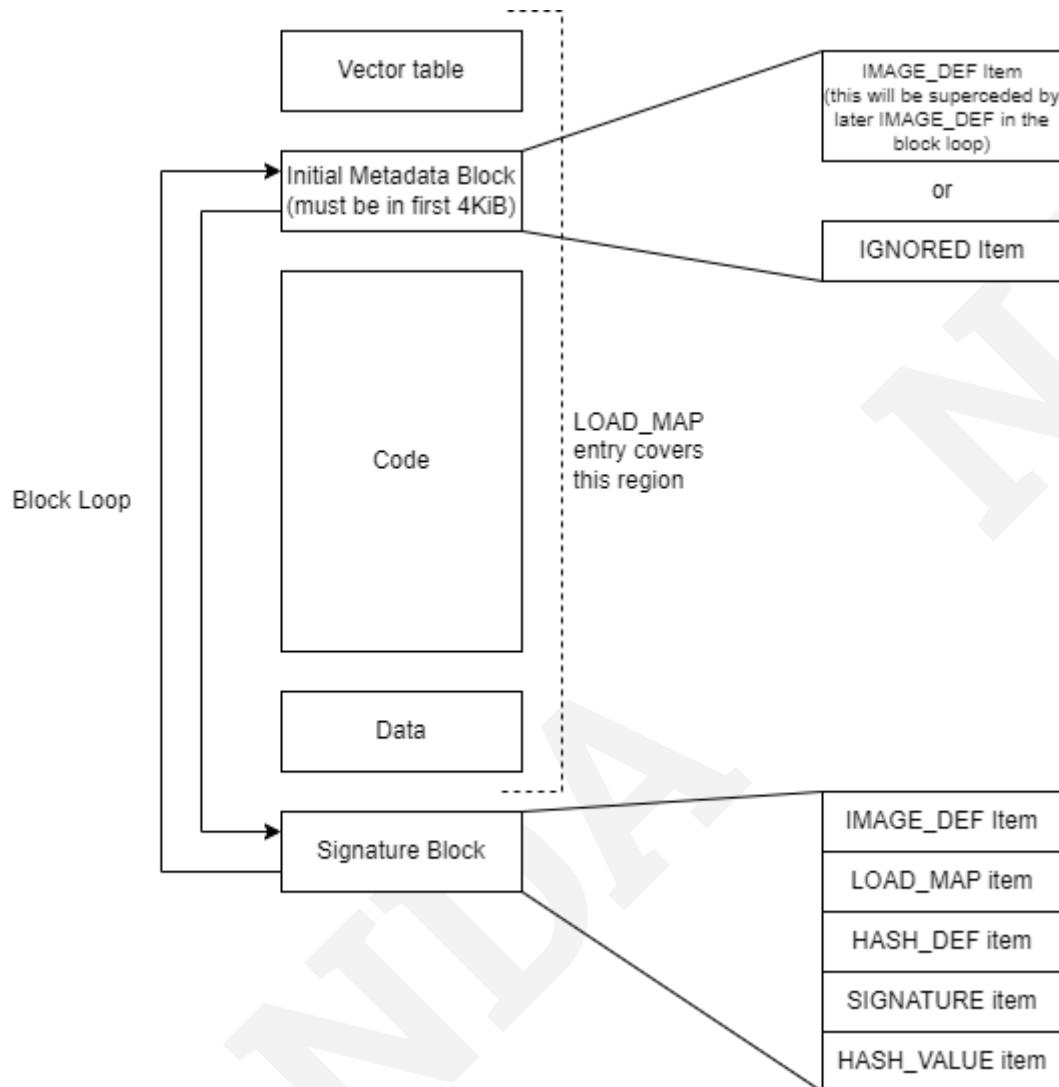
An example of the default Block Loop produced by the SDK is shown below. The empty Block at the end is included to prevent booting of an image that has only been partially written, for example if it's being written into a partition that is too small.

The reason having an empty Block prevents this, is that it will invalidate the Block Loop if it is not written. Therefore a partially written image will not boot, as there is no Block at the address pointed to by the first Block, so the Block Loop is invalid.

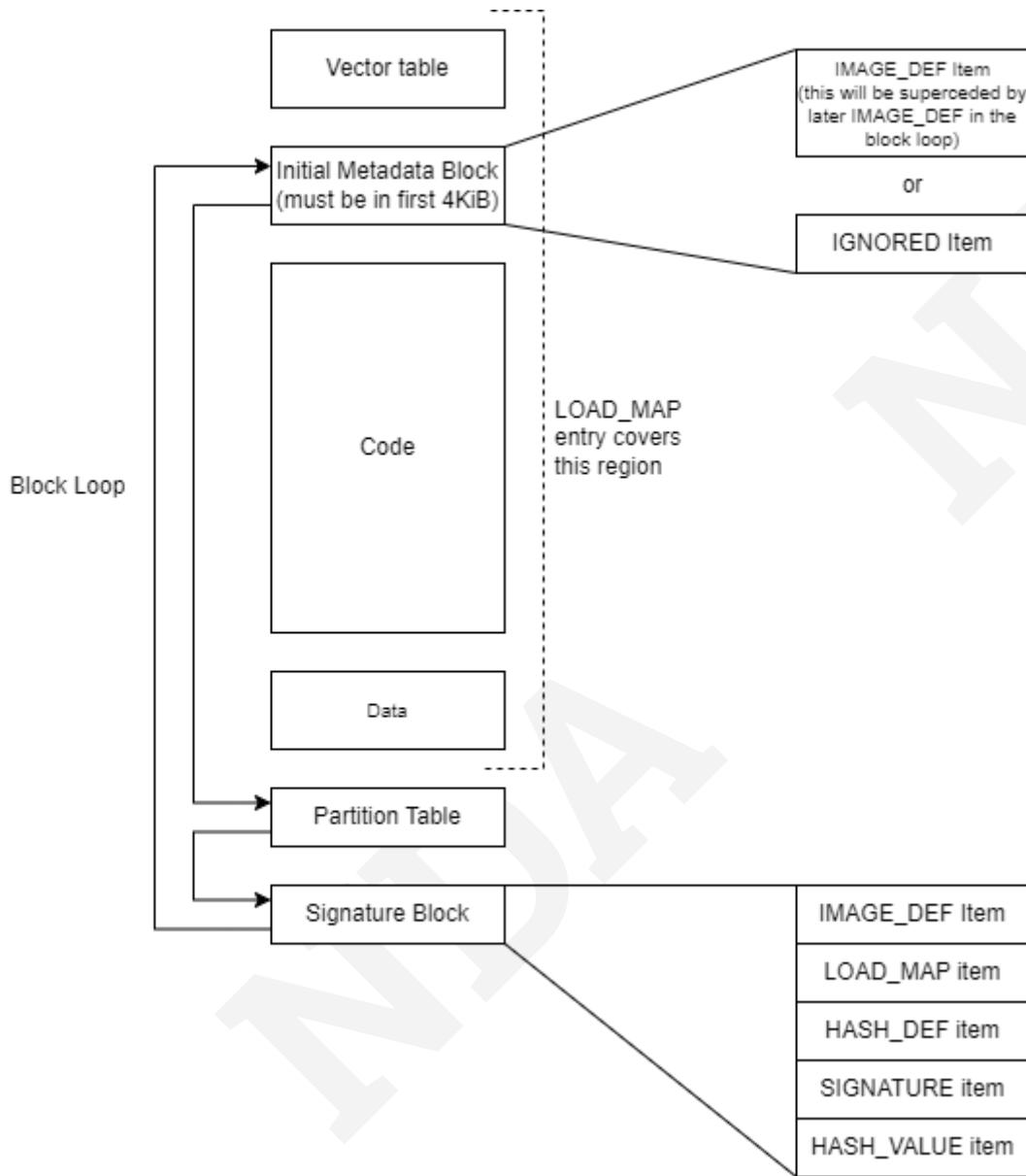


For signed or hashed images produced by the SDK, the Block at the end should now contain the signature or hash value, along with a LOAD_MAP entry indicating the region that is being signed or hashed.

The reason for having the signature or hash Block at the end of the loop, is so that it can be outside the region that is covered by the LOAD_MAP entry. If the hash was included in the initial metadata block, then there would need to be an explicit carve out around the block in the LOAD_MAP, as the LOAD_MAP should not cover the block containing it when being used for hashing or signing.



When using a custom bootloader, a Partition Table can also be embedded in the Block Loop, as shown below and described in [Section 5.2.5](#).



5.3. Processor-Controlled Boot Sequence

```

if (powman_vector_pcsp) {
    // note this call may return
    if (correct_arch) pownam_vector_pc() else hang();
}
if (watchdog_vector_pcsp) {
    // note this call may return
    if (correct_arch) pownam_vector_pc() else hang();
}
entering_ns_boot = BOOTSEL_PRESSED |
    reboot_mode == BOOTSEL |
    (double_tap_enabled_in_otp() && DOUBLE_TAP_DETECTED)
if (!entering_nsboot) {
    // ram_image_window specified by watchdog_scratch
    if (ram_image_window) {
        if (!ram_boot_disabled_in_otp) {

```

```

        // this only returns if there is no valid (signed if necessary) ram image to enter
        try_boot_ram_image(ram_image_window);
    }
} else {
    if (otp_boot_enabled_in_otp && !otp_boot_disabled_in_otp) {
        // this only returns if there is no valid (signed if necessary) OTP image to enter
        try_otp_boot();
    }
    if (!flash_boot_disabled_in_otp) {
        // this only returns if there is no valid (signed if necessary) flash image to enter
        try_flash_boot();
    }
}
}

if (entering_ns_boot) {
    if (bootstrap_select_uart && !nsboot_uart_disabled) {
        // does not return
        nsboot(uart);
    } else if (bootstrap_select_usb && !nsboot_usb_disabled) {
        // does not return
        nsboot(usb);
    }
}
loop_forever();

```

5.3.1. POWMAN Boot Vector

POWMAN contains scratch registers similar to the watchdog scratch registers, which persist over power-down of the switched core power domain, in addition to most system resets. These registers allow users to install their own boot handler, and divert control away from the main boot sequence on non-POR/BOR resets. It recognises the following values written to **BOOT0** through **BOOT3**:

- **BOOT0**: magic number **0xb007c0d3**
- **BOOT1**: Entry point XORed with magic **-0xb007c0d3 (0x4ff83f2d)**
- **BOOT2**: Stack pointer
- **BOOT3**: Entry point

If either of the magic numbers mismatch, POWMAN boot does not take place. If the numbers match, the bootrom zeroes **BOOT0** before entering the vector, so that the behaviour does not persist over subsequent reboots.

The POWMAN boot vector is permitted to return.

5.3.2. Watchdog Boot Vector

Watchdog boot allows users to install their own boot handler, and divert control away from the main boot sequence on non-POR/BOR resets. It recognises the following values written to the watchdog's upper scratch registers:

- Scratch 4: magic number **0xb007c0d3**
- Scratch 5: entry point XORed with magic **-0xb007c0d3 (0x4ff83f2d)**
- Scratch 6: stack pointer
- Scratch 7: entry point

If either of the magic numbers mismatch, watchdog boot does not take place. If the numbers match, the Bootrom zeroes scratch 4 before transferring control, so that the behaviour does not persist over subsequent reboots.

Watchdog boot can also be used to select the bootrom's special boot modes:

- **BOOTSEL** (see [Section 5.3.6](#))
- Flash Update (see [Section 5.1.15](#))
- RAM Image (see [Section 5.3.3](#))

The watchdog boot vector is permitted to return.

5.3.2.1. Special Watchdog Boot

The magic entry point `0xb007c0d3` is used to indicate a special boot type, identified by the stack pointer value:

- **BOOTSEL** (stack pointer = 2) - Boot into **BOOTSEL** mode.
- **RAM_IMAGE** (stack pointer = 3) - Boot into an image stored in SRAM or XIP SRAM.
- **FLASH_UPDATE** (stack pointer = 4) - Normal boot after updating flash.

Parameters to the boot are storing in:

- Scratch 2: Parameter 0
- Scratch 3: Parameter 1

5.3.3. RAM Image Boot

The bootrom is directed (via values in the watchdog registers) to boot into an image in SRAM or XIP SRAM. The two parameters indicate the start and size of the region to search for a Block Loop containing a valid (and correctly signed if necessary) **IMAGE_DEF**. These are passed as parameter 0/1, in watchdog scratch 2/3.

If the image to be booted in is contained in XIP SRAM, then the XIP SRAM must be pinned in place by the bootrom prior to launch. For this reason, if you are using XIP SRAM for your binaries, you must add a special entry to the **LOAD_MAP** item (see [Section 5.10.3.2](#)) to indicate this.

5.3.4. OTP Boot

If OTP boot is enabled, then code from OTP is executed in preference to code from flash. Note that the OTP code is free to "chain" into an executable stored in flash.

Code from OTP is copied into SRAM at the specified location, then execution proceeds similarly to RAM Image Boot. The SRAM with the data copied from OTP is searched for a valid (and correctly signed if necessary) **IMAGE_DEF**. If found it is booted; otherwise OTP boot falls through to Flash Boot (if enabled).

OTP boot could for example be used to execute some hidden decryption code to decode a flash image on startup. The OTP boot code can hide itself (in OTP) even from secure code, once it is done.

5.3.5. Flash Boot

This section will describe the flash boot in detail;

the TLDR is that up to 16 scans are made with the following flash settings in order *until* a valid **IMAGE_DEF** or **PARTITION_TABLE** is found. At this point the flash settings are considered valid, and the flash boot will proceed if a valid bootable **IMAGE_DEF** is found with these settings.

Note therefore, that all the settings will be tried when the flash is empty, as the bootrom has no way to distinguish spotty communication with the flash from garbage flash contents.

Mode	Clock Divider
EBh quad	3
BBh dual	3
0Bh serial	3
03h serial	3
EBh quad	6
BBh dual	6
0Bh serial	6
03h serial	6
EBh quad	12
BBh dual	12
0Bh serial	12
03h serial	12
EBh quad	24
BBh dual	24
0Bh serial	24
03h serial	24

5.3.6. BOOTSEL (USB/UART) Boot

The bootrom samples the state of QSPI CSn shortly after reset, and decides based on the result whether to enter BOOTSEL mode, which refers collectively to the USB and UART bootloaders.

The bootrom initialises the chip select to the following state:

- Output disabled
- Pulled high (note CSn is an active-low signal, so this deselects the external QSPI device if there is one)

If the chip select remains high, the bootrom continues with its normal, non-BOOTSEL sequence. By default on a blank device this means driving the chip select low and attempting to boot from an external flash or PSRAM device.

If chip select is driven low externally, the bootrom enters BOOTSEL mode. You must drive the chip select low with a sufficiently low impedance to overcome the internal pull-up. A 4.7 kΩ resistance to ground is a good intermediate value which reliably creates a low input logic level, but will not affect the output levels when RP2350 drives the chip select.

The QSPI SD1 line, which RP2350 initially pulls low, selects which bootloader to enter:

- SD1 remains pulled low: enter USB bootloader
- SD1 driven high: enter UART bootloader

USB boot is a low-friction method for programming an RP2350 from a sophisticated host like a Linux PC. It also directly exposes more advanced options like OTP programming. See [Section 5.5](#) for the drag-and-drop mass storage interface, or [Section 5.6](#) for the PICOBLOCK vendor interface.

UART boot is a minimal interface for bootstrapping a flashless RP2350 from another microcontroller. UART boot uses QSPI SD2 for UART TX, and QSPI SD3 for UART RX, at a fixed baud rate of 1 Mbaud. See [Section 5.8](#) for more details on UART boot.

5.3.7. Boot Configuration (OTP)

User configuration stored in OTP can be found in [Section 13.9](#), starting at `CRIT1`.

5.4. Bootrom APIs

5.4.1. Locating The API Functions

Some of the bootrom is dedicated to the implementation of the boot sequence and USB boot interfaces. There is also code in the bootrom useful to user programs. [Table 452](#) shows the fixed memory layout of certain words in the Bootrom which are instrumental in locating other content within the bootrom when using the ARM architecture. [Table 453](#) shows the additional entries for use when using the RISC-V architecture.

NOTE

These offsets are correct for A1 not A2

Table 452. Bootrom contents at fixed (well known) addresses for Arm code

Address	Contents	Description
<code>0x00000000</code>	32-bit pointer	Initial boot stack pointer
<code>0x00000004</code>	32-bit pointer	Pointer to boot reset handler function
<code>0x00000008</code>	32-bit pointer	Pointer to boot NMI handler function
<code>0x0000000c</code>	32-bit pointer	Pointer to boot Hard fault handler function
<code>0x00000010</code>	'M', 'u', <code>0x02</code>	Magic
<code>0x00000013</code>	byte	Bootrom version
<code>0x00000014</code>	32-bit pointer	Pointer to rom entry table (<code>BOOTROM_ROMTABLE_START</code>)
<code>0x00000018</code>	32-bit pointer	Pointer to a helper function (<code>rom_table_lookup_val()</code>)
<code>0x0000001c</code>	32-bit pointer	Pointer to a helper function (<code>rom_table_lookup_entry()</code>)

Table 453. Bootrom contents at fixed (well known) addresses for RISC-V code

Address	Contents	Description
<code>0x00007ff0</code>	32-bit pointer	Pointer to rom entry table (<code>BOOTROM_ROMTABLE_START</code>)
<code>0x00007ff4</code>	32-bit pointer	Pointer to a helper function (<code>rom_table_lookup_val()</code>)
<code>0x00007ff8</code>	32-bit pointer	Pointer to a helper function (<code>rom_table_lookup_entry()</code>)
<code>0x00007ffc</code>	32-bit instruction	RISC-V Entry Point

The Bootrom contains a number of public functions that provide useful RP2350 functionality that might be needed in the absence of any other code on the device.

These functions are normally made available to the user by the SDK, however a lower level method is provided to locate them (their locations may change with each Bootrom release) and call them directly.

Assuming the three bytes starting at address `0x00000010` are ('M', 'u', `0x02`) then the other fixed location fields can be assumed to be valid, and used to lookup ROM functionality.

The version byte at offset `0x00000013` is informational, and should not be used to infer the exact location of any functions.

The following code from the SDK shows how to look up a ROM function.

```

static __force_inline void *rom_func_lookup_inline(uint32_t code) {
#ifndef __riscv
    // on RISC-V the code (a jmp) is actually embedded in the table
    rom_table_lookup_fn rom_table_lookup =
        (rom_table_lookup_fn) (uintptr_t)*(uint16_t*)(BOOTROM_TABLE_LOOKUP_ENTRY_OFFSET
        + rom_offset_adjust);
    return rom_table_lookup(code, RT_FLAG_FUNC_RISCV);
#else
    // on ARM the function pointer is stored in the table, so we dereference it
    // via lookup() rather than lookup_entry()
    rom_table_lookup_fn rom_table_lookup =
        (rom_table_lookup_fn) (uintptr_t)*(uint16_t*)(BOOTROM_TABLE_LOOKUP_OFFSET);
    if (pico_processor_state_is_nonsecure()) {
        return rom_table_lookup(code, RT_FLAG_FUNC_ARM_NONSEC);
    } else {
        return rom_table_lookup(code, RT_FLAG_FUNC_ARM_SEC);
    }
#endif
}

```

To lookup a piece of ROM data, the following code is used:

```

void *rom_data_lookup(uint32_t code) {
    rom_table_lookup_fn rom_table_lookup =
        (rom_table_lookup_fn) (uintptr_t)*(uint16_t*)(BOOTROM_TABLE_LOOKUP_OFFSET);
    return rom_table_lookup(code, RT_FLAG_DATA);
}

```

The `code` parameter correspond to the `CODE` values in the tables below, and is calculated as follows:

```

uint32_t rom_table_code(char c1, char c2) {
    return (c2 << 8) | c1;
}

```

These codes are also available in https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/pico_bootrom/include/pico/bootrom.h as `#defines`.

5.4.2. API Function Availability

Not all functions are available under all architectures or security levels; the availability is as follows:

- **ARM-S** - The function is available to secure Arm code. The majority of functions are available for **ARM-S** unless they deal specifically with RISC-V or Non-secure functionality.
- **RISC-V** - The function is available to RISC-V code. Most of the functions that are available under **ARM-S** are also exposed under RISC-V unless they deal specifically with Arm security handling
- **ARM_NS** - The function is available to Non-secure Arm code. The function in this case will perform additional permission and argument checks to prevent secure data leaking or being corrupted. Note that each individual **ARM_NS** must be explicitly enabled by the secure code before use (via `set_ns_api_permission()`), and will return `BOOTROM_ERROR_NOT_PERMITTED` if they are not allowed.

5.4.3. API Function Return Codes

Some functions do not support returning any error, and are marked `void`; the remainder return either 0 (`BOOTROM_OK`) or a positive value (if data needs to be returned) for success. These bootrom error codes are identical to the error codes used by the SDK so can be used interchangeably, hence the gaps in the numbering for SDK error codes that aren't used by the bootrom:

Name	Value	Description
value	> 0	The function succeeded and returned the value
<code>BOOTROM_OK</code>	0	The function executed successfully
<code>BOOTROM_ERROR_NOT_PERMITTED</code>	-4	The operation was disallowed by a security constraint
<code>BOOTROM_ERROR_INVALID_ARG</code>	-5	One or more parameters passed to the function is outside the range of supported values; <code>BOOTROM_ERROR_INVALID_ADDRESS</code> and <code>BOOTROM_ERROR_BAD_ALIGNMENT</code> are more specific errors.
<code>BOOTROM_ERROR_INVALID_ADDRESS</code>	-10	An address argument was out-of-bounds or was determined to be an address that the caller may not access.
<code>BOOTROM_ERROR_BAD_ALIGNMENT</code>	-11	An address passed to the function was not correctly aligned.
<code>BOOTROM_ERROR_INVALID_STATE</code>	-12	Something happened or failed to happen in the past, and consequently the request cannot currently be serviced.
<code>BOOTROM_ERROR_BUFFER_TOO_SMALL</code>	-13	A user-allocated buffer was too small to hold the result or working state of the function.
<code>BOOTROM_ERROR_PRECONDITION_NOT_MET</code>	-14	The call failed because another ROM function must be called first.
<code>BOOTROM_ERROR_MODIFIED_DATA</code>	-15	Cached data was determined to be inconsistent with the full version of the data it was copied from.
<code>BOOTROM_ERROR_INVALID_DATA</code>	-16	A data structure failed validation.
<code>BOOTROM_ERROR_NOT_FOUND</code>	-17	An attempt was made to access something that does not exist; or, a search failed.
<code>BOOTROM_ERROR_UNSUPPORTED_MODIFICATION</code>	-18	Modification is impossible based on current state; e.g. attempted to clear an OTP bit.
<code>BOOTROM_ERROR_LOCK_REQUIRED</code>	-19	A required lock is not owned. See Section 5.4.4 .

5.4.4. API Functions And Exclusive Access

Various bootrom functions require access to parts of the system which cannot be safely accessed by both cores at once, or which prohibit the normal functioning of the part of the system for other uses. Examples include:

- Serial access to OTP; It is not possible to read from the memory mapped OTP regions at the same time as accessing via the serial interface
- Use of the SHA-256 block; Only one SHA-256 sum can be in progress at a time.
- Leaving the XIP access mode to perform flash programming makes all XIP access hard fault.

It is beyond the purview of the bootrom to implement a locking strategy, as the style and scope of the locking required is entirely up to how the application uses these resources.

At the same time however, it is important that, say, an **ARM-NS** user of a flash programming API can't cause a hard fault in secure code running from flash.

The solution the bootrom provides, is to use the BOOTRAM spin locks to inform the bootrom about what resources are currently owned. Applications that could suffer concurrent access problems, must implement their own locking mechanisms, and take ownership of the specified spin locks to signify ownership of a particular resource. The application could use the spin lock in question to implement locking for the resource, but that is its decision!

If the **LOCK_ENABLE** spin lock is owned, then attempts to use APIs requiring resource ownership when the spin lock for the resource is itself not owned, will cause the bootrom API to return **BOOTROM_ERROR_NOT_PERMITTED**.

The following bootrom spin locks (BOOTLOCKS) are used:

- **0x0 : LOCK_SHA_256** - if owned, then a bootrom API is allowed to use the SHA-256 block
- **0x1 : LOCK_FLASH_OP** - if owned, then a bootrom API is allowed to send serial commands to the flash
- **0x2 : LOCK OTP** - if owned, then a bootrom API is allowed to access OTP via the serial interface
- **0x7 : LOCK_ENABLE** - if owned, then bootrom API resource ownership checking is enabled. This is off by default, since the bootrom APIs should be usable by default without any additional setup.

5.4.5. SDK Access To The API

Bootrom functions are exposed in the SDK via the *pico_bootrom* library (see [pico_bootrom](#)).

5.4.6. Alphabetical List Of API Functions / Data

- **chain_image()** ARM-S RISC-V. See [Section 5.4.12.2](#).
- **connect_internal_flash()** ARM-S RISC-V. See [Section 5.4.7.1](#).
- **explicit_buy()** ARM-S RISC-V. See [Section 5.4.12.3](#).
- **flash_devinfo16_ptr** ARM-S RISC-V. See [Section 5.4.15.3](#).
- **flash_enter_cmd_xip()** ARM-S RISC-V. See [Section 5.4.7.6](#).
- **flash_exit_xip()** ARM-S RISC-V. See [Section 5.4.7.2](#).
- **flash_flush_cache()** ARM-S RISC-V. See [Section 5.4.7.5](#).
- **flash_op()** ARM-S ARM-NS 'RISC-V. See [Section 5.4.8.2](#).
- **flash_range_erase()** ARM-S RISC-V. See [Section 5.4.7.3](#).
- **flash_range_program()** ARM-S RISC-V. See [Section 5.4.7.4](#).
- **flash_reset_address_trans()** ARM-S RISC-V. See [Section 5.4.7.8](#).
- **flash_runtime_to_storage_addr()** ARM-S ARM-NS RISC-V. See [Section 5.4.8.1](#).
- **flash_select_xip_read_mode()** ARM-S RISC-V. See [Section 5.4.7.7](#).
- **get_b_partition()** ARM-S RISC-V. See [Section 5.4.12.5](#).
- **get_partition_table_info()** ARM-S ARM-NS RISC-V. See [Section 5.4.11.2](#).
- **get_sys_info()** ARM-S ARM-NS RISC-V. See [Section 5.4.11.1](#).
- **get_uf2_target_partition()** ARM-S RISC-V. See [Section 5.4.12.4](#).
- **api-git_revision** ARM-S ARM-NS RISC-V. See [Section 5.4.15.1](#).
- **load_partition_table()** ARM-S RISC-V. See [Section 5.4.11.3](#).
- **otp_access()** ARM-S ARM-NS RISC-V. See [Section 5.4.11.4](#).

- `partition_table_ptr` ARM-S RISC-V. See [Section 5.4.15.2](#).
- `pick_ab_partition()` ARM-S RISC-V. See [Section 5.4.12.1](#).
- `reboot()` ARM-S ARM-NS RISC-V. See [Section 5.4.10.1](#).
- `secure_call()` ARM-NS. See [Section 5.4.13.1](#).
- `set_bootrom_stack()` RISC-V. See [Section 5.4.14.1](#).
- `set_ns_api_permission()` ARM-S. See [Section 5.4.9.1](#).
- `set_rom_callback()` ARM-S RISC-V. See [Section 5.4.10.3](#).
- `validate_ns_buffer()` ARM-S RISC-V. See [Section 5.4.9.2](#).
- `xip_setup_func_ptr` ARM-S RISC-V. See [Section 5.4.15.4](#).

5.4.7. Low-Level Flash Access Functions

These low-level flash access functions are similar to the ones on RP2040.

5.4.7.1. `connect_internal_flash`

Code: 'I','F'

Signature: `void connect_internal_flash(void)`

Supported architectures: [ARM-S, RISC-V](#)

Restore all QSPI pad controls to their default state, and connect the QMI peripheral to the QSPI pads.

If a secondary flash chip select GPIO has been configured via OTP [FLASH_DEVINFO](#), or by writing to the runtime copy of `FLASH_DEVINFO` in bootrom, then this bank 0 GPIO is also initialised and the QMI peripheral is connected. Otherwise, bank 0 IOs are untouched.

5.4.7.2. `flash_exit_xip`

Code: 'E','X'

Signature: `void flash_exit_xip(void)`

Supported architectures: [ARM-S, RISC-V](#)

Initialise the QMI for serial operations (direct mode), and also initialise a basic XIP mode, where the QMI will perform 03h serial read commands at low speed (CLKDIV=12) in response to XIP reads.

Then, issue a sequence to the QSPI device on chip select 0, designed to return it from continuous read mode ("XIP mode") and/or QPI mode to a state where it will accept serial commands. This is necessary after system reset to restore the QSPI device to a known state, because resetting RP2350 does not reset attached QSPI devices. It is also necessary when user code, having already performed some continuous-read-mode or QPI-mode accesses, wishes to return the QSPI device to a state where it will accept the serial erase and programming commands issued by the bootrom's flash access functions.

If a GPIO for the secondary chip select is configured via `FLASH_DEVINFO`, then the XIP exit sequence is also issued to chip select 1.

The QSPI device should be accessible for XIP reads after calling this function; the name `flash_exit_xip` refers to returning the *QSPI device* from its XIP state to a serial command state.

5.4.7.3. `flash_range_erase`

Code: 'R','E'

Signature: `void flash_range_erase(uint32_t addr, size_t count, uint32_t block_size, uint8_t block_cmd)`

Supported architectures: [ARM-S](#), [RISC-V](#)

Erase `count` bytes, starting at `addr` (offset from start of flash). Optionally, pass a block erase command e.g. [D8h block erase](#), and the size of the block erased by this command – this function will use the larger block erase where possible, for much higher erase speed. `addr` must be aligned to a 4096-byte sector, and `count` must be a multiple of 4096 bytes.

This is a low-level flash API, and no validation of the arguments is performed. See [flash_op\(\)](#) for a higher-level API which checks alignment, flash bounds and partition permissions, and can transparently apply a runtime-to-storage address translation.

The QSPI device must be in a serial command state before calling this API, which can be achieved by calling `connect_internal_flash()` followed by `flash_exit_xip()`. After the erase, the flash cache should be flushed via `flash_flush_cache()` to ensure the modified flash data is visible to cached XIP accesses.

Finally, the original XIP mode should be restored by copying the saved XIP setup function from bootrom into SRAM, and executing it: the bootrom provides a default function which restores the flash mode/clkdiv discovered during flash scanning, and user programs can override this with their own XIP setup function.

For the duration of the erase operation, QMI is in direct mode ([Section 12.14.5](#)) and attempting to access XIP from DMA, the debugger or the other core will return a bus fault. XIP becomes accessible again once the function returns.

5.4.7.4. `flash_range_program`

Code: 'R','P'

Signature: `void flash_range_program(uint32_t addr, const uint8_t *data, size_t count)`

Supported architectures: [ARM-S](#), [RISC-V](#)

Program `data` to a range of flash storage addresses starting at `addr` (offset from the start of flash) and `count` bytes in size. `addr` must be aligned to a 256-byte boundary, and `count` must be a multiple of 256.

This is a low-level flash API, and no validation of the arguments is performed. See [flash_op\(\)](#) for a higher-level API which checks alignment, flash bounds and partition permissions, and can transparently apply a runtime-to-storage address translation.

The QSPI device must be in a serial command state before calling this API – see notes on [flash_range_erase\(\)](#).

5.4.7.5. `flash_flush_cache`

Code: 'F','C'

Signature: `void flash_flush_cache(void)`

Supported architectures: [ARM-S](#), [RISC-V](#)

Flush the entire XIP cache, by issuing an invalidate by set/way maintenance operation to every cache line ([Section 4.4.1](#)). This ensures that flash program/erase operations are visible to subsequent cached XIP reads.

Note that this unpins pinned cache lines, which may interfere with cache-as-SRAM use of the XIP cache.

No other operations are performed.

5.4.7.6. `flash_enter_cmd_xip`

Code: 'C', 'X'

Signature: `void flash_enter_cmd_xip(void)`

Supported architectures: [ARM-S](#), [RISC-V](#)

Alias for `flash_select_xip_read_mode(0, 12);`.

Configure the QMI to generate a standard `03h` serial read command, with 24 address bits, upon each XIP access. This is a slow XIP configuration, but is widely supported. CLKDIV is set to 12. The debugger may call this function to ensure that flash is readable following a program/erase operation.

Note that the same setup is performed by `flash_exit_xip()`, and the RP2350 flash program/erase functions do not leave XIP in an inaccessible state, so calls to this function are largely redundant. It is provided for compatibility with RP2040.

5.4.7.7. `flash_select_xip_read_mode`

Code: 'X', 'M'

Signature: `void flash_select_xip_read_mode(bootrom_xip_mode_t mode, uint8_t clkdiv)`

Supported architectures: [ARM-S](#), [RISC-V](#)

Configure QMI for one of a small menu of XIP read modes supported by the bootrom. This mode is configured for both memory windows (both chip selects), and the clock divisor is also applied to direct mode.

The available modes are:

- `0`: `03h` serial read: serial address, serial data, no wait cycles
- `1`: `0Bh` serial read: serial address, serial data, 8 wait cycles
- `2`: `BBh` dual-IO read: dual address, dual data, 4 wait cycles (including MODE bits, which are driven to 0)
- `3`: `EBh` quad-IO read: quad address, quad data, 6 wait cycles (including MODE bits, which are driven to 0)

The XIP write command/format are not configured by this function.

When booting from flash, the bootrom tries each of these modes in turn, from 3 down to 0. The first mode that is found to work is remembered, and a default XIP setup function is written into bootrom that calls *this function* (`flash_select_xip_read_mode`) with the parameters discovered during flash scanning. This can be called at any time to restore the flash parameters discovered during flash boot.

All XIP modes configured by the bootrom have an 8-bit serial command prefix, so that the flash can remain in a serial command state, meaning XIP accesses can be mixed more freely with program/erase serial operations. This has a performance penalty, so users can perform their own flash setup *after flash boot* using continuous read mode or QPI mode to avoid or alleviate the command prefix cost.

5.4.7.8. `flash_reset_address_trans`

Code: 'R', 'A'

Signature: `void flash_reset_address_trans(void)`

Supported architectures: [ARM-S](#), [RISC-V](#)

Restore the QMI address translation registers, `ATRANS0` through `ATRANS7`, to their reset state. This makes the runtime-to-storage address map an identity map, i.e. the mapped and unmapped address are equal, and the entire space is fully mapped. See [Section 12.14.4](#).

5.4.8. High-level flash access functions

The higher level access functions, provide functionality that is safe to expose (with permissions) to Non-secure code as well.

5.4.8.1. `flash_runtime_to_storage_addr`

Code: 'F','A'

Signature: `int flash_runtime_to_storage_addr(uint32_t addr)`

Supported architectures: [ARM-S](#), [ARM-NS](#), [RISC-V](#)

Applies the address translation currently configured by QMI address translation registers, [ATRANS0](#) through [ATRANS7](#). See [Section 12.14.4](#).

Translating an address outside of the XIP runtime address window, or beyond the bounds of an [ATRANSx_SIZE](#) field, returns [BOOTROM_ERROR_INVALID_ADDRESS](#), which is not a valid flash storage address. Otherwise, return the storage address which QMI would access when presented with the runtime address `addr`. This is effectively a virtual-to-physical address translation for QMI.

5.4.8.2. `flash_op`

Code: 'F','0'

Signature: `int flash_op(uint32_t flags, uint32_t addr, uint32_t size_bytes, uint8_t *buf)`

Supported architectures: [ARM-S](#), [ARM-NS](#), [RISC-V](#)

Perform a flash read, erase, or program operation. Erase operations must be sector-aligned (4096 bytes) and sector-multiple-sized, and program operations must be page-aligned (256 bytes) and page-multiple-sized; misaligned erase and program operations will return [BOOTROM_ERROR_BAD_ALIGNMENT](#). The operation – erase, read, program – is selected by the [CFLASH_OP_BITS](#) bitfield of the `flags` argument.

`addr` is the address of the first flash byte to be accessed, ranging from [XIP_BASE](#) to [XIP_BASE + 0xffffffff](#) inclusive. This may be a runtime or storage address. `buf` contains data to be written to flash, for program operations, and data read back from flash, for read operations. `buf` is never written by program operations, and is completely ignored for erase operations.

The flash operation is bounds-checked against the known flash devices specified by the runtime value of [FLASH_DEVINFO](#), stored in boottrom. This is initialised by the boottrom to the OTP value [FLASH_DEVINFO](#), if [BOOT_FLAGS0.FLASH_DEVINFO_ENABLE](#) is set; otherwise it is initialised to 16 MiB for chip select 0 and 0 bytes for chip select 1. [FLASH_DEVINFO](#) can be updated at runtime by writing to its location in boottrom, the pointer to which can be looked up in the ROM table.

If a resident partition table is in effect, then the flash operation is also checked against the partition permissions. The Secure version of this function can specify the caller's effective security level (Secure, Non-secure, bootloader) using the [CFLASH_SECLEVEL_BITS](#) bitfield of the `flags` argument, whereas the Non-secure function is always checked against the Non-secure permissions for the partition. Flash operations which span two partitions are not allowed, and will fail address validation.

If [FLASH_DEVINFO.D8H_ERASE_SUPPORTED](#) is set, erase operations will use a D8h 64 kB block erase command where possible (without erasing outside the specified region), for faster erase time. Otherwise, only 20h 4 kB sector erase commands are used.

Optionally, this API can translate `addr` from flash runtime addresses to flash storage addresses, according to the translation currently configured by QMI address translation registers, [ATRANS0](#) through [ATRANS7](#). For example, an image stored at a +2 MiB offset in flash (but mapped at XIP address 0 at runtime), writing to an offset of +1 MiB into the image, will write to a physical flash storage address of 3 MiB. Translation is enabled by setting the [CFLASH_ASPACE_BITS](#) bitfield in the `flags` argument.

When translation is enabled, flash operations which cross address holes in the XIP runtime address space (created by non-maximum `ATRANSx_SIZE`) will return an error response. This check may tear: the transfer may be partially performed before encountering an address hole and ultimately returning failure.

When translation is enabled, flash operations are permitted to cross chip select boundaries, provided this does not span an ATRANS address hole. When translation is disabled, the entire operation must target a single flash chip select (as determined by bits 24 and upward of the address), else address validation will fail.

A typical call sequence for erasing a flash sector in the runtime address space from Secure code would be:

- `connect_internal_flash();`
- `flash_exit_xip();`
- `flash_op((CFLASH_OP_VALUE_ERASE << CFLASH_OP_LSB) | (CFLASH_SECLEVEL_VALUE_SECURE << CFLASH_SECLEVEL_LSB) | (CFLASH_ASPACE_VALUE_RUNTIME << CFLASH_ASPACE_LSB), addr, 4096, NULL);`
- `flash_flush_cache();`
- Copy the XIP setup function from bootrom to SRAM and execute it, to restore the original XIP mode
 - The bootrom will have written a default setup function which restores the mode/clkdiv parameters found during flash search; user code can overwrite this with its own custom setup function.

A similar sequence is required for program operations. Read operations can leave the current XIP mode in effect, so only the `flash_op(...);` call is required.

Note that the RP2350 ROM leaves the flash in a basic XIP state in between program/erase operations. However, during a program/erase operation, the QMI is in direct mode ([Section 12.14.5](#)) and any attempted XIP access will return a bus error response.

5.4.9. Security Related Functions

5.4.9.1. `set_ns_api_permission`

Code: 'S','P'

Signature: `int set_ns_api_permission(uint ns_api_num, bool allowed)`

Supported architectures: [ARM-S](#)

Allow or disallow the specific NS API (note all NS APIs default to disabled).

`ns_api_num` is one of the following, configuring [ARM-NS](#) access to the given API. When an NS API is disabled, calling it will return `BOOTROM_ERROR_NOT_PERMITTED`.

- `0x0: get_sys_info`
- `0x1: flash_op`
- `0x2: flash_runtime_to_storage_addr`
- `0x3: get_partition_table_info`
- `0x4: secure_call`
- `0x5: otp_access`
- `0x6: reboot`
- `0x7: get_b_partition`

NOTE

All permissions default to disallowed after a reset (see also [bootrom_state_reset\(\)](#)).

5.4.9.2. validate_ns_buffer

Code: 'V','B'

Signature: `void validate_ns_buffer(const void *addr, uint32_t size, uint32_t write, uint32_t *ok)`

Supported architectures: [ARM-S](#)

Utility method that can be used by secure ARM code to validate a buffer passed to it from Non-secure code.

Both the `write` parameter and the (out) result parameter `ok` are RCP booleans, so `0xa500a500` for true, and `0x00c300c3` for false. This enables hardening of this function, and indeed the `write` parameter must be one of these values or the RCP will hang the system.

For success, the entire buffer must fit in range `XIP_BASE` → `SRAM_END`, and must be accessible by the Non-secure caller according to SAU + NS MPU (privileged or not based on current processor IPSR and NS CONTROL flag). Buffers in USB RAM are also allowed if access is granted to NS via ACCESSCTRL.

5.4.10. Miscellaneous Functions**5.4.10.1. reboot**

Code: 'R','B'

Signature: `int reboot(uint32_t flags, uint32_t delay_ms, uint32_t p0, uint32_t p1)`

Supported architectures: [ARM-S](#), [ARM-NS](#), [RISC-V](#)

Resets the RP2350 and uses the watchdog facility to restart.

The `delay_ms` is the millisecond delay before the reboot occurs. Note: by default this method is asynchronous (unless `NO_RETURN_ON_SUCCESS` is set - see below), so the method will return and the reboot will happen this many milliseconds later.

The `flags` field contains one of the following values:

- `0x0000` : [REBOOT_TYPE_NORMAL](#) - reboot into the normal boot path.
- `0x0002` : [REBOOT_TYPE_BOOTSEL](#) - reboot into BOOTSEL mode.
 - `p0` - the GPIO number to use as an activity indicator (enabled by flag in `p1`).
 - `p1` - a set of flags:
 - `0x01` : [DISABLE_MSD_INTERFACE](#) - Disable the BOOTSEL USB drive (see [Section 5.5](#)).
 - `0x02` : [DISABLE_PICOBOOT_INTERFACE](#) - Disable the PICOBLOCK interface (see [Section 5.6](#)).
 - `0x10` : [GPIO_PIN_ACTIVE_LOW](#) - The GPIO in `p0` is active low.
 - `0x20` : [GPIO_PIN_ENABLED](#) - Enable the activity indicator on the specified GPIO.
- `0x0003` : [REBOOT_TYPE_RAM_IMAGE](#) - reboot into an image in RAM. The region of RAM or XIP RAM is searched for an image to run. This is the type of reboot used when a RAM UF2 is dragged onto the BOOTSEL USB drive.
 - `p0` - the region start address (word-aligned).
 - `p1` - the region size (word-aligned).

- **0x0004 : REBOOT_TYPE_FLASH_UPDATE** - variant of **REBOOT_TYPE_NORMAL** to use when flash has been updated. This is the type of reboot used after dragging a flash UF2 onto the BOOTSEL USB drive.
 - **p0** - the address of the start of the region of flash that was updated. If this address matches the start address of a partition or slot, then that partition or slot is treated preferentially during boot (when there is a choice). This type of boot facilitates TBYB ([Section 5.1.16](#)) and version downgrades.
- **0x000d : REBOOT_TYPE_PC_SP** - reboot to a specific PC and SP. Note: this is not allowed in the **ARM-NS** variant.
 - **p0** - the initial program counter (PC) to start executing at. This must have the lowest bit set for Arm and clear for RISC-V
 - **p1** - the initial stack pointer (SP).

All of the above, can have optional flags ORed in:

- **0x0010 : REBOOT_TO_ARM** - switch both cores to the Arm architecture (rather than leaving them as is). The call will fail with **BOOTROM_ERROR_INVALID_STATE** if the Arm architecture is not supported.
- **0x0020 : REBOOT_TO_RISCV** - switch both cores to the RISC-V architecture (rather than leaving them as is). The call will fail with **BOOTROM_ERROR_INVALID_STATE** if the RISC-V architecture is not supported.
- **0x0100 : NO_RETURN_ON_SUCCESS** - the watchdog h/w is asynchronous. Setting this bit forces this method not to return *if* the reboot is successfully initiated.

5.4.10.2. `bootrom_state_reset`

Code: 'S','R'

Signature: `void bootrom_state_reset(uint32_t flags)`

Supported architectures: **ARM-S, RISC-V**

Resets internal bootrom state, based on the following flags:

- **0x0001 : STATE_RESET_CURRENT_CORE** - Resets any internal bootrom state for the current core into a clean state. This method should be called prior to calling any other bootrom APIs on the current core, and is called automatically by the bootrom during normal boot of core 0 and launch of code on core 1.
- **0x0002 : STATE_RESET_OTHER_CORE** - Resets any internal bootrom state for the other core into a clean state. This is generally called by a debugger when resetting the state of one core via code running on the other.
- **0x0004 : STATE_RESET_GLOBAL_STATE** - Resets all non core-specific state, including:
 - Disables access to bootrom APIs from **ARM-NS** (see also [set_ns_api_permission\(\)](#)).
 - Unlocks all BOOT spinlocks.
 - Clears any secure code callbacks. (see also [set_rom_callback\(\)](#))

Note: the SDK calls this method on runtime initialisation to put the bootrom into a known state. This allows the program to function correctly if it is entered (e.g. from a debugger) without taking the usual boot path (which resets the state appropriately itself).

5.4.10.3. `set_rom_callback`

Code: 'R','C'

Signature: `int set_rom_callback(uint callback_number, int (*callback)(…))`

Supported architectures: **ARM-S, RISC-V**

The only supported `callback_number` is **0** which sets the callback used for the **secure_call** API.

A callback value of **0** deletes the callback function, a positive value (all valid function pointers are on RP2350) sets the callback function, and a negative value does not set a new value.

If successful, returns ≥ 0 (the existing value of the function pointer on entry to the function).

5.4.11. System Information Functions

5.4.11.1. `get_sys_info`

Code: 'G', 'S'

Signature: `int get_sys_info(uint32_t *out_buffer, uint32_t out_buffer_word_size, uint32_t flags)`

Supported architectures: ARM-S, ARM-NS, RISC-V

Fills a buffer with various system information. Note that this API is also used to return information over the PICOBOOT interface.

On success, the buffer is filled, and the number of words filled in the buffer is returned.

The information returned is chosen by the flags parameter; the first word in the returned buffer, is the (sub)set of those flags that the API supports. You should always check this value before interpreting the buffer.

Following the first word, returns words of data for each present flag in order:

- **0x0001 : CHIP_INFO** - unique identifier for the chip (3 words)
 - Word 0 : Value of the `CHIP_INFO_PACKAGE_SEL` register
 - Word 1 : RP2350 device id
 - Word 2 : RP2350 wafer id
- **0x0002 : CRITICAL** (1 word)
 - Word 0 : Value of the OTP `CRITICAL` register, containing critical boot flags read out on last OTP reset event
- **0x0004 : CPU_INFO** (1 word)
 - Word 0 : Current CPU architecture
 - 0 - Arm
 - 1 - RISC-V
- **0x0008 : FLASH_DEV_INFO** (1 word)
 - Word 0 : Flash device info in the format of OTP `FLASH_DEVINFO`
- **0x0010 : BOOT_RANDOM** - a 128-bit random number generated on each boot (4 words)
 - Word 0 : Per boot random number 0
 - Word 1 : Per boot random number 1
 - Word 2 : Per boot random number 2
 - Word 3 : Per boot random number 3
- **0x0040 : BOOT_INFO** (4 words)
 - Word 0 : `0xttppbbdd`
 - **tt** - recent boot TBYB and update info (updated on regular non BOOTSEL boots)
 - **pp** - recent boot partition (updated on regular not BOOTSEL boots)
 - **bb** - boot type of the most recent boot
 - **dd** - recent boot diagnostic "partition"

- o Word 1 : Recent boot diagnostic. Diagnostic information from a recent boot (with information from the partition (or slot) indicated by `dd` above. "partition" numbers here are:
 - 0-15 : a partition number
 - -1 : none
 - -2 : slot 0
 - -3 : slot 1
 - -4 : image (the diagnostic came from the launch of a RAM image, OTP boot image or user `chain_image()` call).
- o Word 2 : Last reboot param 0
- o Word 3 : Last reboot param 1

"Boot Diagnostic" information is intended to help identify the cause of a failed boot, or booting into an unexpected binary. This information can be retrieved via PICOBLOCK after a watchdog reboot, however it will not survive a reset via the RUN pin or POWMAN reset.

There is only one word of diagnostic information. What it records is based on the `pp` selection above, which is itself set as a parameter when rebooting programmatically into a normal boot.

To get diagnostic info, `pp` must refer to a slot or an "A" partition; image diagnostics are automatically selected on boot from OTP or RAM image, or when `chain_image()` is called.)

The diagnostic word thus contains data for either slot 0 and slot 1, or the "A" partition (and its "B" partition if it has one). The low half word of the diagnostic word contains information from slot 0 or partition A; the high half word contains information from slot 1 or partition B.

The format of each half-word is as follows (using the word *region* to refer to slot or partition)

- `0x0001 : REGION_SEARCHED` - The region was searched for a Block Loop.
- `0x0002 : INVALID_BLOCK_LOOP` - A Block Loop was found but it was invalid
- `0x0004 : VALID_BLOCK_LOOP` - A valid Block Loop was found (Blocks from a loop wholly contained within the region, and the Blocks have the correct structure. Each Block consists of items whose sizes sum to the size of the Block)
- `0x0008 : VALID_IMAGE_DEF` - A valid `IMAGE_DEF` was found in the region. A valid `IMAGE_DEF` must parse correctly and must be executable.
- `0x0010 : HAS_PARTITION_TABLE` - Whether a partition table is present. This partition table must have a correct structure formed if `VALID_BLOCK_LOOP` is set. If the partition table turns out to be invalid, then `INVALID_BLOCK_LOOP` is set too (thus both `VALID_BLOCK_LOOP` and `INVALID_BLOCK_LOOP` will both be set).
- `0x0020 : CONSIDERED` - There was a choice of partition/slot and this one was considered. The first slot/partition is chosen based on a number of factors. If the first choice fails verification, then the other choice will be considered.
 - o the version of the `PARTITION_TABLE/IMAGE_DEF` present in the slot/partition respectively.
 - o whether the slot/partition is the "update region" as per a `FLASH_UPDATE` reboot.
 - o whether an `IMAGE_DEF` is marked as "explicit buy"
- `0x0040 : CHOSEN` - This slot/partition was chosen (or was the only choice)
- `0x0080 : PARTITION_TABLE_MATCHING_KEY_FOR_VERIFY` - if a signature is required for the `PARTITION_TABLE` (via OTP setting), then whether the `PARTITION_TABLE` is signed with a key matching one of the four stored in OTP
- `0x0100 : PARTITION_TABLE_HASH_FOR_VERIFY` - set if a hash value check could be performed. In the case a signature is required, this value is identical to `PARTITION_TABLE_MATCHING_KEY_FOR_VERIFY`
- `0x0200 : PARTITION_TABLE_VERIFIED_OK` - whether the `PARTITION_TABLE` passed verification (signature/hash if present/required)
- `0x0400 : IMAGE_DEF_MATCHING_KEY_FOR_VERIFY` - if a signature is required for the `IMAGE_DEF` due to secure boot, then

whether the `IMAGE_DEF` is signed with a key matching one of the four stored in OTP.

- `0x0800 : IMAGE_DEF_HASH_FOR_VERIFY` - set if a hash value check could be performed. In the case a signature is required, this value is identical to `IMAGE_DEF_MATCHING_KEY_FOR_VERIFY`
- `0x1000 : IMAGE_DEF_VERIFIED_OK` - whether the `PARTITION_TABLE` passed verification (signature/hash if present/required) and any `LOAD_MAP` is valid
- `0x2000 : LOAD_MAP_ENTRIES_LOADED` - whether any code was copied into RAM due to a `LOAD_MAP`
- `0x4000 : IMAGE_LAUNCHED` - whether an `IMAGE_DEF` from this region was launched
- `0x8000 : IMAGE_CONDITION_FAILURE` - whether the `IMAGE_DEF` failed final checks before launching; these checks include:
 - verification failed (if it hasn't been verified earlier in the `CONSIDERED` phase).
 - a problem occurred setting up any rolling window.
 - the rollback version could not be set in OTP (if required in secure mode)
 - the image was marked as Non-secure
 - the image was marked as "explicit buy", and this was a flash boot, but then region was not the "flash update" region
 - the image has the wrong architecture, but architecture auto-switch is disabled (or the correct architecture is disabled)

To get a full picture of a failed boot involving slots and multiple partitions, the device can be rebooted multiple times to gather the information.

5.4.11.2. `get_partition_table_info`

Code: `'G','P'`

Signature: `int get_partition_table_info(uint32_t *out_buffer, uint32_t out_buffer_word_size, uint32_t flags_and_partition)`

Supported architectures: `ARM-S, ARM-NS, RISC-V`

Fills a buffer with information from the partition table. Note that this API is also used to return information over the PICOBOOT interface.

On success, the buffer is filled, and the number of words filled in the buffer is returned. If the partition table has not been loaded (e.g. from a watchdog or RAM boot), then this method will return `BOOTROM_ERROR_NO_DATA`, and you should load the partition table via `load_partition_table()` first.

Note that not all data from the partition table is kept resident in memory by the bootrom due to size constraints. To protect against changes being made in flash after the bootrom has loaded the resident portion, the bootrom keeps a hash of the partition table as of the time it loaded it. If the hash has changed by the time this method is called, then it will return `BOOTROM_ERROR_INVALID_STATE`.

The information returned is chosen by the `flags_and_partition` parameter; the first word in the returned buffer, is the (sub)set of those flags that the API supports. You should always check this value before interpreting the buffer.

Following the first word, returns words of data for each present flag in order. With the exception of `PT_INFO`, all the flags select "per partition" information, so each field is returned in flag order for one partition after the next. The special `SINGLE_PARTITION` flag indicates that data for only a single partition is required. Flags include:

- `0x0001 - PT_INFO` : information about the partition table as a whole. The second two words for unpartitioned space in the same form described in [Section 5.10.4.2](#).
 - Word 0 : `partition_count` (low 8 bits), `partition_table_present` (bit 8)
 - Word 1 : `unpartitioned_space_permissions_and_location`
 - Word 2 : `unpartitioned_space_permissions_and_flags`

- **0x8000 - SINGLE_PARTITION** : only return data for a single partition; the partition number is stored in the top 8 bits of `flags_and_partition`

Per-partition fields:

- **0x0010 - PARTITION_LOCATION_AND_FLAGS** : the core information about a partition. The format of these fields is described in [Section 5.10.4.2](#).
 - Word 0 - `permissions_and_location`
 - Word 1 - `permissions_and_flags`
- **0x0020 - PARTITION_ID** : the optional 64-bit identifier for the partition. If the HAS_ID bit is set in the partition flags, then the 64 bit ID is returned:
 - Word 0 - first 32 bits
 - Word 1 - second 32 bits
- **0x0040 - PARTITION_FAMILY_IDS** : Any additional UF2 family-ids that the partition supports being downloaded into it via the MSD bootloader beyond the standard ones flagged in the `permissions_and_flags` field (see [Section 5.10.4.2](#)).
- **0x0080 - PARTITION_NAME** : The optional name for the partition. If the HAS_NAME field bit in `permissions_and_flags` is not set, then no data is returned for this partition; otherwise the format is as follows:
 - Byte 0 : 7 bit length of the name (LEN); top bit reserved
 - Byte 1 : first character of name
 - ...
 - Byte LEN : last character of name
 - ... (*padded up to the next word boundary*)

5.4.11.3. `load_partition_table`

Code: 'L','P'

Signature: `int load_partition_table(uint8_t *workarea_base, uint32_t workarea_size, bool force_reload)`

Supported architectures: [ARM-S](#), [RISC-V](#)

Loads the current partition table from flash, if present.

This method potentially requires similar complexity to the boot path in terms of picking amongst versions, checking signatures etc. As a result it requires a user provided memory buffer as a work area. The work area should be word-aligned and of sufficient size or `BOOTROM_ERROR_INSUFFICIENT_RESOURCES` will be returned. The work area size currently required is 3064, so 3K is a good choice.

If `force_reload` is false, then this method will return `BOOTROM_OK` immediately if the bootrom is loaded, otherwise it will reload the partition table if it has been loaded already, allowing for the partition table to be updated in a running program.

5.4.11.4. `otp_access`

Code: 'O','A'

Signature: `int otp_access(uint8_t *buf, uint32_t buf_len, uint32_t row_and_flags)`

Supported architectures: [ARM-S](#), [ARM-NS](#), [RISC-V](#)

Writes data from a buffer into OTP, or reads data from OTP into a buffer.

- **0x0000ffff - ROW_NUMBER**: 16 low bits are row number (0-4095)

- **0x00010000** - **IS_WRITE**: if set, do a write (not a read)
- **0x00020000** - **IS_ECC**: if this bit is set, each value in the buffer is 2 bytes and ECC is used when read/writing from 24 bit value in OTP. If this bit is not set, each value in the buffer is 4 bytes, the low 24-bits of which are written to or read from OTP.

The buffer must be aligned to 2 bytes or 4 bytes according to the **IS_ECC** flag.

This method will read and write rows until the first row it encounters that fails a key or permission check at which it will return **BOOTROM_ERROR_NOT_PERMITTED**.

Writing will also stop at the first row where an attempt is made to set an OTP bit from a 1 to a 0, and **BOOTROM_ERROR_INVALID_STATE** will be returned.

If all rows are read/written successfully, then **BOOTROM_OK** will be returned.

5.4.12. Boot Related Functions

5.4.12.1. `pick_ab_partition`

Code: 'A', 'B'

Signature: `int pick_ab_partition(uint8_t *workarea_base, uint32_t workarea_size, uint partition_a_num)`

Supported architectures: **ARM-S, RISC-V**

Determines which of the partitions has the "better" **IMAGE_DEF**. In the case of executable images, this is the one that would be booted

This method potentially requires similar complexity to the boot path in terms of picking amongst versions, checking signatures etc. As a result it requires a user provided memory buffer as a work area. The work area should be word aligned, and of sufficient size or **BOOTROM_ERROR_INSUFFICIENT_RESOURCES** will be returned. The work area size currently required is 3064, so 3K is a good choice.

The passed partition number can be any valid partition number other than the "B" partition of an A/B pair.

This method returns a negative error code, or the partition number of the picked partition if (i.e. `partition_a_num` or the number of its "B" partition if any).

NOTE

This method does not look at owner partitions, only the A partition passed and its corresponding B partition.

5.4.12.2. `chain_image`

Code: 'C', 'I'

Signature: `int chain_image(uint8_t *workarea_base, uint32_t workarea_size, int32_t region_base, uint32_t region_size)`

Supported architectures: **ARM-S, RISC-V**

Searches a memory region for a launchable image, and executes it if possible.

The `region_base` and `region_size` specify a word-aligned, word-multiple-sized area of RAM, XIP RAM or flash to search. The first 4 kB of the region must contain the start of a Block Loop with an **IMAGE_DEF**. If the new image is launched, the call does not return otherwise an error is returned.

The `region_base` is signed, as a negative value can be passed, which indicates that the (negated back to positive value) is both the `region_base` and the base of the "flash update" region.

This method potentially requires similar complexity to the boot path in terms of picking amongst versions, checking

signatures etc. As a result it requires a user provided memory buffer as a work area. The work area should be word aligned, and of sufficient size or `BOOTROM_ERROR_INSUFFICIENT_RESOURCES` will be returned. The work area size currently required is 3064, so 3K is a good choice.

ⓘ NOTE

This method is primarily expected to be used when implementing bootloaders.

ⓘ NOTE

When chaining into an image, the `BOOT_FLAGS0.ROLLBACK_REQUIRED` flag will not be set, to prevent invalidating a bootloader without a rollback version by booting a binary which has one (see [Section 5.2.7](#)).

5.4.12.3. `explicit_buy`

Code: `'E','B'`

Signature: `int explicit_buy(uint8_t *buffer, uint32_t buffer_size)`

Supported architectures: `ARM-S RISC-V`

Perform an "explicit" buy of an executable launched via an `IMAGE_DEF` which was "explicit buy" flagged. A "flash update" boot of such an image is a way to have the image execute once, but only become the "current" image if it calls back into the bootrom via this call.

This call may perform the following:

- Erase and rewrite the part of flash containing the "explicit buy" flag in order to clear said flag.
- Erase the first sector of the other partition in an A/B partition scenario, if this new `IMAGE_DEF` is a version downgrade (so this image will boot again when not doing a "flash update" boot)
- Update the rollback version in OTP if the chip is secure, and a rollback version is present in the image.

ⓘ NOTE

The device may reboot while updating the rollback version, if multiple rollback rows need to be written - this occurs when the version crosses a multiple of 24 (for example upgrading from version 23 to 25 requires a reboot, but 23 to 24 or 24 to 25 doesn't). The application should therefore be prepared to reboot when calling this function, if rollback versions are in use.

Note that the first of the above requires 4 kiB of scratch space, so you should pass a word aligned buffer of at least 4 kiB to this method, or it will return `BOOTROM_ERROR_INSUFFICIENT_RESOURCES` if the "explicit buy" flag needs to be cleared.

5.4.12.4. `get_uf2_target_partition`

Code: `'G','U'`

Signature: `int get_uf2_target_partition(uint8_t *workarea_base, uint32_t workarea_size, uint32_t family_id, resident_partition_t *partition_out)`

Supported architectures: `ARM-S RISC-V`

5.4.12.5. `get_b_partition`

Code: `'G','B'`

Signature: `int get_b_partition(uint partition_a)`

Supported architectures: [ARM-S](#) [RISC-V](#)

Returns: The index of the B partition of partition A if a partition table is present and loaded, and there is a partition A with a B partition; otherwise returns `BOOTROM_ERROR_NOT_FOUND`.

5.4.13. Non-secure-specific Functions

5.4.13.1. `secure_call`

Code: `'S','C'`

Signature: `int secure_call(...)`

Supported architectures: [ARM-NS](#)

Call a secure method from Non-secure code, passing the method to be called in the register `r4` (other arguments passed as normal).

This method provides the ability to decouple the Non-secure code from the secure code, allowing the former to call methods in the latter without needing to know the location of the methods.

This call will always return `BOOTROM_ERROR_INVALID_STATE` unless secure Arm code has provided a handler function via `set_rom_callback()`; if there is a handler function, this method will return the return code that the handler returns, with the convention that `BOOTROM_ERROR_INVALID_ARG` if the "function selector" (in `r4`) is not supported.

Certain "function selectors" are pre-defined to facilitate interaction between secure and Non-secure SDK code, or indeed with other environments.

5.4.14. RISC-V specific functions

5.4.14.1. `set_bootrom_stack`

Code: `'S','S'`

Signature: `int set_bootrom_stack(uint32_t base_size[2])`

Supported architectures: [RISC-V](#)

Most bootrom functions are written just once, in Arm code, to save space. As a result these functions are emulated when running under the RISC-V architecture. This is largely transparent to the user, however the stack used by the Arm emulation is separate from the calling user's stack, and is stored in boot RAM but is of quite limited size. When using certain of the more complex APIs or if nesting bootrom calls from within IRQs, you may need to provide a large stack.

This method allows the caller to specify a region of RAM to use as the stack for the current core by passing a pointer to two values: the word aligned base address, and the size in bytes (multiple of 4).

The method fills in the previous base/size values into the passed array before returning.

5.4.15. Data Values

The Bootrom table contains the following data entries.

5.4.15.1. `git_revision`

Code: `'G','R'`

Type: `const uint32_t git_revision`

The 8 most significant hex digits of the bootrom git revision. Uniquely identifies this version of the bootrom.

NOTE

This is the git revision as built at tapeout: the git hash in the public repository will be different due to squashed history, even though the contents is the same. The bootrom contents can be verified by building the public bootrom source and comparing to a binary dumped from the chip.

5.4.15.2. `partition_table_ptr`

Code: `'P','T'`

Type: `uint32_t *partition_table_ptr`

A pointer to the resident partition table info. The resident partition table is the subset of the full partition table that is kept in memory, and used for flash permissions.

The public part of the resident partition table info is of the form:

```
uint8_t          partition_count;
// If >= partition_count includes any regions added at runtime
uint8_t          permission_partition_count;
bool            loaded;
bool            _pad;
uint32_t         unpartitioned_space_permissions_and_flags;
resident_partition_t partitions[16];
```

`resident_partition_t` is of the form:

```
uint32_t  location_and_permissions;
uint32_t  flags_and_permissions
```

Details of the fields can be found in [Section 5.10.4](#).

5.4.15.3. `flash_devinfo16_ptr`

Code: `'F','D'`

Type: `uint16_t *flash_devinfo16_ptr`

Pointer to the flash device info used by the flash APIs, e.g. for bounds checking against size of flash devices, and configuring the GPIO used for secondary QSPI chip select.

If `BOOT_FLAGS0.FLASH_DEVINFO_ENABLE` is set, this bootrom location is initialised from `FLASH_DEVINFO` at startup, otherwise it is initialised to:

- Chip select 0 size: 16 MiB
- Chip select 1 size: 0 bytes
- No chip select 1 GPIO

- No D8h erase command support

The flash APIs use this bootram copy of FLASH_DEVINFO, so flash device info can be updated at runtime by writing through this pointer.

5.4.15.4. `xip_setup_func_ptr`

Code: 'X','F'

Type: `void *(xip_setup_func_ptr)(void)`

5.5. USB Mass Storage Interface

The Bootrom provides a standard USB bootloader that makes a writeable drive available for copying code to the RP2350 using UF2 files (see [Section 5.5.2](#)).

A suitable UF2 file copied to the drive is downloaded and written to Flash or RAM, and the device is automatically rebooted, making it trivial to download and run code on the RP2350 using only a USB connection.

5.5.1. The RP2350 Drive

The RP2350 appears as a standard 128MB flash drive named [RP2350](#) formatted as a single partition with FAT16. There are only ever two actual files visible on the drive specified.

- [INFO_UF2.TXT](#) - contains a string description of the UF2 bootloader and version.
- [INDEX.HTM](#) - redirects to information about the RP2350 device.

The contents of these files and the name of the drive may be customized, see [Section 5.7](#).

NOTE

By default, the [INDEX.HTM](#) file redirects to <https://www.raspberrypi.com/documentation/microcontrollers/>.

Any type of files may be written to the USB drive from the host, however in general these are not stored, and only appear to be so because of caching on the host side.

When a suitable UF2 file is written to the device however, the special contents are recognised and data is written to specified locations in RAM or Flash.

Where flash targeted UF2s are written on RP2350 is determined by the *family id* of the UF2 contents and the partition table.

If there is no partition table, then UF2s are stored at the address they specify; otherwise they (with the exception of the special *ABSOLUTE* family id) are stored into a single partition, with UF2 flash address `0x10000000` mapping to the start of the partition.

It is possible, based on the partition table or family id, that the UF2 is not downloadable anywhere in flash, in which case it will be ignored. Further detail can be discovered via [GET_INFO - UF2_STATUS](#) On the completed download of an entire valid UF2 file, the RP2350 automatically reboots to run the newly downloaded code.

5.5.2. UF2 Format Details

💡 TIP

To generate UF2 files, use the UF2 convert functionality in [picotool](#).

ℹ️ NOTE

Invalid UF2 files may not write at all or only write partially to RP2350 before failing. Not all operating systems notify you of disk write errors after a failed write. You can use [picotool](#) to verify that a UF2 file wrote correctly to RP2350.

All data destined for the device must be in UF2 blocks with:

- `familyID` present, with a value in the range `0xe48bff58` through `0xe48bff5b` (see table in [Section 5.5.3](#))
- A `payload_size` of `256`

All data must be destined for (and fit entirely within) the following memory ranges (depending on the type of binary being downloaded which is determined by the address of the first UF2 block encountered):

- A regular flash binary
 - `0x10000000-0x11000000` Flash: All blocks must be targeted at 256 byte alignments. Writes beyond the end of physical flash will wrap back to the beginning of flash.
- A RAM *only* binary
 - `0x20000000-0x20082000` Main RAM: Blocks can be positioned with byte alignment.
 - `0x13ffc000-0x14000000` XIP RAM: (since flash is not being targeted, the Flash Cache is available for use as RAM with same properties as *Main RAM*).

ℹ️ NOTE

Traditionally UF2 has only been used to write to Flash, but this is more a limitation of using the metadata-free `.BIN` file as the source to generate the UF2 file. RP2350 takes full advantage of the inherent flexibility of UF2 to support the full range of binaries in the richer `.ELF` format produced by the build to be used as the source for the UF2 file.

- The `numBlocks` must specify a total size of the binary that fits in the regions specified above
- A change of `numBlocks` or the binary type (determined by *UF2* block target address) will discard the current transfer in progress.
- All data must be in blocks without the `UF2_FLAG_NOT_MAIN_FLASH` marking which relates to content to be ignored rather than Flash vs RAM.

ℹ️ NOTE

When targeting flash, the *UF2* block target addresses are interpreted to be in the content of a flash binary that *starts at* `0x10000000`. The *UF2* image may be downloaded into a partition that starts somewhere else in flash, so the actual storage address is `uf2_image_target_base + uf2_block_target_addr - 0x10000000`.

The flash is always erased a 4 kiB sector at a time, so including data for only a subset of the 256-byte pages within a sector in a flash-binary UF2 will leave the remaining 256-byte pages of the sector erased but undefined. The RP2350 bootrom will accept UF2 binaries with such partially-filled sectors, however due to a bug (erratum RP2040-E14 in the RP2040 datasheet) such binaries may not be written correctly if there is any partially-filled sector other than at the end. Most flash binaries are 4-kiB-aligned and contiguous, and therefore it *is* usually only the last sector that is partially-filled. If you need to write non-aligned or non-contiguous UF2s to flash, then you should make sure to include a full 4 kiB worth of data for every sector in flash that will be written other than the last. This is handled for you automatically by the `elf2uf2` tool in the SDK version 1.3.1 onwards, which explicitly adds zero-filled pages to the appropriate *partially-filled* sectors.

A binary is considered "downloaded" when each of the `numBlocks` blocks has been seen at least once in the course of a single valid transfer. The data for a block is only written the first time in case of the host resending duplicate blocks.

After downloading a regular flash binary, a reset is performed after which the flash binary second stage (at address `0x10000000` - the start of flash) will be entered (if valid) via the bootrom.

A downloaded *RAM only* binary is entered by watchdog reset into the start of the binary, which is calculated as the lowest address of a downloaded block (with Main RAM considered lower than Flash Cache if both are present).

It is possible for host software to temporarily disable UF2 writes via the PICOBLOCK interface to prevent interference with operations being performed via that interface (see below), in which case any UF2 file write in progress will be aborted.

NOTE

If a problem is encountered downloading the UF2, then it will appear as if *nothing has happened* since the device will not reboot. The `picotool` command `uf2info` can be used to determine the status of the last download in this case (see also [GET_INFO - UF2_STATUS](#)).

5.5.3. UF2 Targeting Rules

When the first block of a UF2 is downloaded, a choice is made where to store the UF2 in flash based on the *family ID* of the UF2. This choice is performed by the same code as the `get_uf2_target_partition()` API (see [Section 5.4.12.4](#)).

The following family IDs are defined by the bootrom, however the user is free to use their own for more specific targeting:

Name	Value	Description
<code>absolute</code>	<code>0xe48bff57u</code>	Special family ID for content intended to be written directly to flash, ignoring partitions
<code>rp2040</code>	<code>0xe48bff56u</code>	RP2040 executable image
<code>data</code>	<code>0xe48bff58u</code>	Generic catch all for data UF2s
<code>rp2350_arm_s</code>	<code>0xe48bff59u</code>	RP2350 ARM Secure image (i.e. one intended to be booted by the bootrom)
<code>rp2350_riscv</code>	<code>0xe48bff5au</code>	RP2350 RISC-V image
<code>rp2350_arm_ns</code>	<code>0xe48bff5bu</code>	RP2350 ARM Non-Secure image. This is not directly bootable by the bootrom, however secure user code is likely to want to be able to locate binaries of this type

NOTE

the only information available to the algorithm that makes the choice of where to store the UF2, is the UF2 family ID; the algorithm cannot look inside at the UF2 contents as UF2 data sectors may appear at the device in any order.

A UF2 with the `absolute` family ID is downloaded without regard to partition boundaries. A partition table (if present) or OTP configuration can define whether `absolute` family ID downloads are allowed, and download to the start of flash. The default factory settings allow for `absolute` family ID downloads

If there is a partition table present, any other family IDs download to a single partition; if there is no partition table present then the `data`, `rp2350-arm-s` (if Arm architecture is enabled) and `rp2350-riscv` (if RISC-V architecture is enabled) family IDs are allowed by default, and the UF2 is always downloaded to the start of flash.

If a partition table is present, then up to four passes are made over the partition table (from first to last partition encountered) until a matching partition is found; Each pass has different selection criteria:

1. Look for an (unowned) A partition, ignoring those marked `NOT_BOOTABLE` for the current CPU architecture

Use of the `NOT_BOOTABLE` flags allows you to have separate boot partitions for each CPU architecture (ARM / RISC-V); were you not to use `NOT_BOOTABLE` flags in this scenario, and say the first encountered partition has an ARM `IMAGE_DEF`, then, when booting under the RISC-V architecture with auto architecture switching enabled, the bootrom would just switch back into the ARM architecture to boot the ARM binary. Marking the first partition as `NOT_BOOTABLE_RISCV` in the partition table solves this problem.

The correct CPU architecture refers to a match between the architecture of the UF2 (determined by family ID of `rp2350_arm_s` or `rp2350_riscv`) and the current CPU architecture.

This pass allows the user to drop either ARM or RISC-V UF2s, and have them stored as you'd want for the `NOT_BOOTABLE` flag scenario.

2. If auto architecture switching is enabled and the other architecture is available, look for an (unowned) A partition, ignoring those marked `NOT_BOOTABLE` for the that CPU architecture

This pass is designed to match the boot use case of booting images from the other architecture as a fallback. If there is a partition that would be booted as a result auto architecture switching then this a reasonable place to store this UF2 for the alternative architecture.

3. Look for any unowned A partition that accepts the family ID

This pass provides a way to target any UF2s to a partitions based on family ID, but assumes that you'd prefer a UF2 to go into a matching top-level partition vs an owned partition.

4. Finally, look for any A partition that accepts the family ID

This pass implicitly only looks at owned partitions, since unowned partitions would have been matched in the previous pass.

If none of the passes find a match, then the UF2 contents will not be downloaded. The `picotool` command `uf2info` can be used to determine the status of the last download in this case (see also [GET_INFO - UF2_STATUS](#)).

5.5.3.1. A / B Partitions And Ownership

You will note that each of the above passes refers to finding an A partition (remember, any partition that isn't a B partition is an A partition; i.e. an unpaired partition is classed as an A partition).

If the found A partition does not have a B partition paired with it, then the A partition is the UF2 target partition.

If however, the A partition has a B partition, then a further choice must be made as to which of the A / B partitions should be targeted.

1. If the A partition is unowned, then the partition choice is made based on any current valid `IMAGE_DEF` in those partitions. The valid partition with the higher version number is not chosen; in the case of executable `IMAGE_DEFs`, this is the opposite of what would happen during boot; this makes sense as you want to drop the UF2 on the partition which isn't currently booting.
2. If the A partition is marked owned, then the contents of the A partition and B partition are assumed not to contain `IMAGE_DEFs` which can be used to make a version based choice. Therefore, the owner of the A partition (A_{owner}) and its B partition (B_{owner}) are used to make the choice

It is however dependent on the use case whether you would want a UF2 that is destined for partition A / partition B to go into partition A when partition A_{owner} has an `IMAGE_DEF` with the higher version (i.e. would boot if the `IMAGE_DEF` was executable) or when B_{owner} has an `IMAGE_DEF` with the higher version. By default, the bootrom picks partition A when partition A_{owner} has the higher versioned `IMAGE_DEF`, however this can be changed by setting the `UF2_DOWNLOAD_AB_NON_BOOTABLE_OWNER_AFFINITY` flag in partition A.

5.5.3.2. Multiple UF2 families

It is possible to include sectors targeting different family IDs in the same UF2 file. The intention in the UF2 specification is to allow one file to be shipped for multiple different devices, but the expectation is that each device only accepts one UF2 family ID.

Similarly on RP2350, it is only supported to download a UF2 file containing multiple family IDs if *only one* of those family IDs is acceptable for download to the device according to the above rules.

5.6. USB PICOBLOCK Interface

The PICOBLOCK interface is a low level USB protocol for interacting with the RP2350 while it is in BOOTSEL mode. This interface may be used concurrently with the USB Mass Storage Interface.

It provides for flexible reading from and writing to RAM or Flash, rebooting, executing code on the device and a handful of other management functions.

Constants and structures related to the interface can be found in the SDK header https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/common/boot_picoblock_headers/include/boot/picoblock.h

5.6.1. Identifying The Device

A RP2350 device can be recognised by the Vendor ID and Product ID in its device descriptor (shown in [Table 454](#)), unless different values have been set in OTP.

Table 454. RP2350 Boot Device Descriptor

Field	Value
bLength	18
bDescriptorType	1
bcdUSB	2.10
bDeviceClass	0
bDeviceSubClass	0
bDeviceProtocol	0
bMaxPacketSize0	64
idVendor	0x2e8a - this value may be overridden in OTP
idProduct	0x000f - this value may be overridden in OTP
bcdDevice	1.00 - this value may be overridden in OTP
iManufacturer	1
iProduct	2
iSerial	3
bNumConfigurations	1

5.6.2. Identifying The Interface

The PICOBLOCK interface is recognised by the vendor-specific Interface Class, the zero Interface Sub Class and Interface Protocol (shown in [Table 455](#)). Note that you should not rely on the interface number, as that is dependent on whether the device is also exposing the Mass Storage Interface. Note also that the device equally may not be exposing the PICOBLOCK interface at all, so you should not assume it is present.

Table 455. PICOBLOCK Interface Descriptor

Field	Value
bLength	9

Field	Value
bDescriptorType	4
bInterfaceNumber	varies
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	0xff (vendor specific)
bInterfaceSubClass	0
bInterfaceProtocol	0
iInterface	0

5.6.3. Identifying The Endpoints

The PICOBOOT interface provides a single `BULK_OUT` and a single `BULK_IN` endpoint. These can be identified by their direction and type. You should not rely on endpoint numbers.

5.6.4. PICOBOOT Commands

The two bulk endpoints are used for sending commands and retrieving successful command results. All commands are exactly 32 bytes (see [Table 456](#)) and sent to the `BULK_OUT` endpoint.

Table 456. PICOBOOT Command Definition

Offset	Name	Description
0x00	dMagic	The value 0x431fd10b
0x04	dToken	A user provided token to identify this request by
0x08	bCmdId	The ID of the command. Note that the top bit indicates data transfer direction (0x80 = IN)
0x09	bCmdSize	Number of bytes of valid data in the args field
0x0a	reserved	0x0000
0x0c	dTransferLength	The number of bytes the host expects to send or receive over the bulk channel
0x10	args	16 bytes of command-specific data padded with zeros

If a command sent is invalid or not recognised, the bulk endpoints will be stalled. Further information will be available via the `GET_COMMAND_STATUS` request (see [Section 5.6.5.2](#)).

Following the initial 32 byte packet, if `dTransferLength` is non-zero, then that many bytes are transferred over the bulk pipe and the command is completed with an empty packet in the opposite direction. If `dTransferLength` is zero then command success is indicated by an empty IN packet.

The following commands are supported (note common fields `dMagic`, `dToken`, and `reserved` are omitted for clarity)

5.6.4.1. EXCLUSIVE_ACCESS (0x01)

Claim or release exclusive access for writing to the RP2350 over USB (versus the Mass Storage Interface)

Table 457. PICOBLOCK
EXCLUSIVE_ACCESS
command structure

Offset	Name	Value / Description	
0x08	bCmdId	0x01 (EXCLUSIVE_ACCESS)	
0x09	bCmdSize	0x01	
0x0c	dTransferLength	0x00000000	
0x10	bExclusive	NOT_EXCLUSIVE (0)	No restriction on USB Mass Storage operation
		EXCLUSIVE (1)	Disable USB Mass Storage writes (the host should see them as write protect failures, but in any case any active UF2 download will be aborted)
		EXCLUSIVE_AND_EJECT (2)	Lock the USB Mass Storage Interface out by marking the drive media as not present (ejecting the drive)

5.6.4.2. REBOOT (0x02)

Not supported on RP2350.

Use [Section 5.6.4.10](#) instead.

5.6.4.3. FLASH_ERASE (0x03)

Erases a contiguous range of flash sectors.

Table 458. PICOBLOCK
FLASH_ERASE
command structure

Offset	Name	Value / Description
0x08	bCmdId	0x03 (FLASH_ERASE)
0x09	bCmdSize	0x08
0x0c	dTransferLength	0x00000000
0x10	dAddr	The address in flash to erase, starting at this location. This must be sector (4 kiB) aligned
0x14	dSize	The number of bytes to erase. This must an exact multiple number of sectors (4 kiB)

5.6.4.4. READ (0x84)

Read a contiguous memory (Flash or RAM or ROM) range from the RP2350

Table 459. PICOBLOCK
Read memory
command (Flash,
RAM, ROM) structure

Offset	Name	Value / Description
0x08	bCmdId	0x84 (READ)
0x09	bCmdSize	0x08
0x0c	dTransferLength	Must be the same as dSize
0x10	dAddr	The address to read from. May be in Flash or RAM or ROM
0x14	dSize	The number of bytes to read

5.6.4.5. WRITE (0x05)

Writes a contiguous memory range of memory (Flash or RAM) on the RP2350.

Table 460. PICOBOOT Write memory command (Flash, RAM) structure

Offset	Name	Value / Description
0x08	bCmdId	0x05 (WRITE)
0x09	bCmdSize	0x08
0x0c	dTransferLength	Must be the same as dSize
0x10	dAddr	The address to write from. May be in Flash or RAM, however must be page (256 byte) aligned if in Flash. Note the flash must be erased first or the results are undefined.
0x14	dSize	The number of bytes to write. If writing to flash and the size is not an exact multiple of pages (256 bytes) then the last page is zero-filled to the end.

5.6.4.6. EXIT_XIP (0x06)

A no-op provided for compatibility with RP2040. An XIP exit sequence ([flash_exit_xip\(\)](#)) is issued once before entering the USB bootloader, which returns the external QSPI device from whatever XIP state it was in to a serial command state, and the external QSPI device then remains in this state until reboot.

Table 461. PICOBOOT EXIT_XIP command structure

Offset	Name	Value / Description
0x08	bCmdId	0x06 (EXIT_XIP)
0x09	bCmdSize	0x00
0x0c	dTransferLength	0x00000000

5.6.4.7. ENTER_XIP (0x07)

A no-op provided for compatibility with RP2040. Note that, unlike RP2040, the low-level bootrom flash operations do not leave the QSPI interface in a state where XIP is inaccessible, therefore there is no need to reinitialise the interface each time. XIP setup is performed once before entering the USB bootloader, using an 03h command with a fixed clock divisor of 6.

Table 462. PICOBOOT Enter Execute in place (XIP) command

Offset	Name	Value / Description
0x08	bCmdId	0x07 (ENTER_XIP)
0x09	bCmdSize	0x00
0x0c	dTransferLength	0x00000000

5.6.4.8. EXEC (0x08)

Not supported on RP2350.

5.6.4.9. VECTORIZE_FLASH (0x09)

Not supported on RP2350.

5.6.4.10. REBOOT2 (0x0a)

Reboots the RP2350 out of BOOTSEL mode. Note that BOOTSEL mode may be re-entered if no valid bootable image is found.

The parameters *flags*, *delay_ms*, *p0*, *p1* are the same as for [api_reboot\(\)](#)

Table 463. PICOBOOT REBOOT2 command structure

Offset	Name	Value / Description
0x08	bCmdId	0x0a (REBOOT2)
0x09	bCmdSize	0x10
0x0c	dTransferLength	0x00000000
0x10	dAddr	flags
0x14	dSize	delay_ms
0x18	dSize	p0
0x1c	dSize	p1

5.6.4.11. GET_INFO (0x0b)

Generic conduit for retrieving information from the device.

The transfer length indicates the maximum number of bytes to be retrieved. The first word returned indicates the number of significant words of data that follow. A full "transfer length" is always returned, padding with zeroes as necessary.

Note "Word 0" below refers to the first word of the actual response (i.e. the word after the count word).

Table 464. PICOBOOT GET_INFO command structure

Offset	Name	Value / Description
0x08	bCmdId	0x0b (GET_INFO)
0x09	bCmdSize	0x10
0x0c	dTransferLength	the size of data to be received. Note this must be a multiple of 4, and less than 256

Offset	Name	Value / Description
0x10	bType	<p>the type of information being retrieved:</p> <ul style="list-style-type: none"> • 0x1 - INFO_SYS : Retrieves information from <code>get_sys_info()</code>; the <code>flag</code> parameter for that function comes from <code>dParam0</code>. • 0x2 - PARTITION : Retrieves information from <code>get_partition_table_info()</code>; the <code>flags_and_partition</code> parameter for that function comes from <code>dParam0</code>. • 0x3 - UF2_TARGET_PARTITION : Retrieves the partition that a given UF2 family_id would be downloaded into (if it were dragged on the USB drive in BOOTSEL mode). The family id is passed in <code>dParam0</code>. <ul style="list-style-type: none"> ◦ Word 0 : Target partition number: <ul style="list-style-type: none"> ▪ 0-15 : the partition number the family would be downloaded to ▪ 0xff : if the family would be downloaded at an absolute location ▪ -1 : if there is nowhere to download the family ◦ Word 1 : Target partition Section 5.10.4.2 if the partition number is not -1 ◦ Word 2 : Target partition Section 5.10.4.2 if the partition number is not -1 • 0x4 - UF2_STATUS : Retrieves information about the current/recent UF2 download <ul style="list-style-type: none"> ◦ Word 0 - 0nnrr00af <ul style="list-style-type: none"> ▪ 'n' - no reboot flag; if 0x01, there is no reboot when the UF2 download completes ▪ 'r' - if 0x01, the UF2 being download is a RAM UF2 ▪ 'a' - UF2 download abort reason flags <ul style="list-style-type: none"> ▪ 0x1 EXCLUSIVELY_LOCKED ▪ 0x2 BAD_ADDRESS ▪ 0x4 WRITE_ERROR ▪ 0x8 REBOOT_FAILURE // if the UF2 targeted a disabled architecture ▪ 'f' - UF2 download status flags <ul style="list-style-type: none"> ▪ 0x1 IGNORED_FAMILY ◦ Word 1 - the current family id ◦ Word 2 - the number of 256 byte blocks successfully downloaded ◦ Word 3 - the total number of 256 byte blocks in the UF2 to download

5.6.4.12. OTP_READ (0x8c)

Reads data out of OTP. (see also `otp_access()` which provides the data). Note that data returned is subject to the "BL" OTP permissions which define "boot loader" OTP access permissions.

Table 465. PICOBOOT
OTP_READ command
structure

Offset	Name	Value / Description
0x08	bCmdId	0x8c (OTP_READ)
0x09	bCmdSize	0x05
0x0c	dTransferLength	$wRowCount \times (bEcc ? 2 : 4)$
0x10	wRow	the first row number to read
0x12	wRowCount	the number of rows to read
0x14	bEcc	<ul style="list-style-type: none"> 0 - if reading raw rows (32 bits are returned per row, the top 8 of which are zero) 1 - if reading rows as ECC rows (16 bits per row are returned)

5.6.4.13. OTP_WRITE (0x0d)

Reads data out of OTP. (see also [otp_access\(\)](#) which performs the operation). Note that writing is subject to the "BL" OTP permissions which define "boot loader" OTP access permissions.

Table 466. PICOBOOT
OTP_WRITE command
structure

Offset	Name	Value / Description
0x08	bCmdId	0x0d (OTP_WRITE)
0x09	bCmdSize	0x05
0x0c	dTransferLength	$wRowCount \times (bEcc ? 2 : 4)$
0x10	wRow	the first row number to read
0x12	wRowCount	the number of rows to read
0x14	bEcc	<ul style="list-style-type: none"> 0 - if writing raw rows (32 bits are provided per row, the top 8 of which are ignored) 1 - if writing ECC rows (16 bits are provided per row, and are written with error correcting information to the OTP)

5.6.5. Control Requests

The following requests are sent to the interface via the default control pipe.

5.6.5.1. INTERFACE_RESET (0x41)

The host sends this control request to reset the PICOBOOT interface. This command:

- Clears the HALT condition (if set) on each of the bulk endpoints
- Aborts any in-process PICOBOOT or Mass Storage transfer and any flash write (this method is the only way to kill a stuck flash transfer).
- Clears the previous command result
- Removes [EXCLUSIVE_ACCESS](#) and remounts the Mass Storage drive if it was ejected due to exclusivity.

Table 467. PICOBOOT
Reset PICOBOOT
interface control

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000001b	01000001b	0000h	Interface	0000h	none

This command responds with an empty packet on success.

5.6.5.2. GET_COMMAND_STATUS (0x42)

Retrieve the status of the last command (which may be a command still in progress). Successful completion of a PICOBOOT Protocol Command is acknowledged over the bulk pipe, however if the operation is still in progress or has failed (stalling the bulk pipe), then this method can be used to determine the operation's status.

Table 468. PICOBOOT
Get last command
status control

bmRequestType	bRequest	wValue	wIndex	wLength	Data
11000001b	01000010b	0000h	Interface	0000h	none

The command responds with the following 16 byte response

Table 469. PICOBOOT
Get last command
status control
response

Offset	Name	Description
0x00	dToken	The user token specified with the command

Offset	Name	Description	
0x04	dStatusCode	OK (0)	The command completed successfully (or is still in progress)
		UNKNOWN_CMD (1)	The ID of the command was not recognised
		INVALID_CMD_LENGTH (2)	The length of the command request was incorrect
		INVALID_TRANSFER_LENGTH (3)	The data transfer length was incorrect given the command
		INVALID_ADDRESS (4)	The address specified was invalid for the command type; i.e. did not match the type Flash/RAM that the command was expecting
		BAD_ALIGNMENT (5)	The address specified was not correctly aligned according to the requirements of the command
		INTERLEAVED_WRITE (6)	A Mass Storage Interface UF2 write has interfered with the current operation. The command was abandoned with unknown status. Note this will not happen if you have <i>exclusive</i> access.
		REBOOTING (7)	The device is in the process of rebooting, so the command has been ignored.
		UNKNOWN_ERROR (8)	Some non-specific error occurred.
		INVALID_STATE (9)	Something happened or failed to happen in the past, and consequently the request can't (currently) be serviced.
		NOT_PERMITTED (10)	Permission violation e.g. write to read-only flash partition.
		INVALID_ARG (11)	Argument is outside of range of supported values.
		BUFFER_TOO_SMALL (12)	The provided buffer was too small to hold the result.
		PRECONDITION_NOT_MET (13)	The operation failed because another ROM function must be called first.
		MODIFIED_DATA (14)	Cached data was determined to be inconsistent with the full version of the data it was calculated from.
		INVALID_DATA (15)	A data structure failed to validate.
		NOT_FOUND (16)	Attempted to access something that does not exist; or, a search failed.
		UNSUPPORTED_MODIFICATION (17)	Write is impossible based on previous writes; e.g. attempted to clear an OTP bit.
0x08	bCmdId	The ID of the command	
0x09	bInProgress	1 if the command is still in progress	0 otherwise
0x0a	reserved	(6 zero bytes)	

5.7. USB White-Labeling

To brand RP2350-based products, customers must replace identifying information exposed by USB interfaces. We call this **white-labeling**, and you can accomplish it in RP2350 by specifying values in OTP.

Begin by writing the OTP location of the white-label data to `USB_WHITE_LABEL_ADDR`. Then, set `USB_BOOT_FLAGS.WHILE_LABEL_ADDR_VALID` to mark the white-labelling as valid. Do not set the valid flag without first programming valid white-label data.

Using this process, you can modify the value of the following fields:

5.7.1. USB Device Descriptor

The USB device descriptor includes the following 16-bit values:

- `VID` (default `0x2e8a`)
- `PID` (default `0x000f`)
- `BCD_DEVICE` (default `0x0100`)
- `LANG_ID` (default `0x0409`)

5.7.2. USB Device Strings

- `MANUFACTURER` (default "Raspberry Pi", max-length 30 UTF-16 or ASCII chars)
- `PRODUCT` (default "RP2350 Boot", max-length 30 UTF-16 or ASCII chars)
- `SERIAL_NUMBER` (default hex string of `device_id_lo, device_id_hi, wafer_id_lo, wafer_id_hi` - i.e. first 4 rows of OTP, max-length 30 UTF-16 or ASCII chars)

5.7.3. USB Configuration Descriptor

The USB Configuration Description is not strictly white-labelling, but is still helpful for users:

- `ATTRIBUTES_MAX_POWER_VALUES` (default `0xfa80` i.e. `bMaxPower` of `0xfa`, `bmAttributes=0x80`)

5.7.4. MSD Drive

- `VOLUME_LABEL` (default "RP2350", max-length 11 ASCII chars)

5.7.5. UF2 INDEX.HTM File

This is of the form:

```
<html>
  <head>
    <meta http-equiv="refresh" content="0;URL='*REDIRECT_URL*' />
  </head>
  <body>Redirecting to <a href='`*REDIRECT_URL*`'*REDIRECT_NAME*</a></body>
</html>
```

- `REDIRECT_URL` (default "https://raspberrypi.com/device/RP2?version=5A09D5466F90", note the 12 hex digits are the first 6 of the `SYSINFO_GITREF_RP2350` and the first 6 of the bootrom `gitref`, max-length 127 ASCII chars)
- `REDIRECT_NAME` (default "raspberrypi.com", max-length 127 ASCII chars)

5.7.6. UF2 INFO_UF2.TXT File

This is of the form:

```
UF2 Bootloader v1.0
Model: MODEL
Board-ID: BOARD_ID
```

- **MODEL** (default "Raspberry Pi RP2350", max-length 127 ASCII chars)
- **BOARD_ID** (default "RP2350", max-length 127 ASCII chars)

5.7.7. SCSI Inquiry

Returned via the SCSI Inquiry command:

- **VENDOR** (default "RPI", max-length 8 ASCII chars)
- **PRODUCT** (default "RP2350", max-length 16 ASCII chars)
- **VERSION** (default "1", max-length 4 ASCII chars)

5.7.8. Volume Label Example

The following example demonstrates how to change the volume label using **picotool** to the value "SPOON":

NOTE

Newer versions of **picotool** can load white-label data from a JSON file using the **picotool otp white-label -s <start row> <JSON filename>** command.

1. First, define the row of white label structure to be **0x400**:

```
$ picotool otp set -e OTP_DATA_USB_WHITE_LABEL_ADDR 0x400
```

2. Next, because the volume label is located at index **0x8** within **OTP_DATA_USB_WHITE_LABEL_ADDR**, write to **0x408**. Define the location of the volume label string to be offset from **OTP_DATA_USB_WHITE_LABEL_ADDR** by **0x30**. For this example, "SPOON" has 5 characters, so we write **0x3005** to **0x408**:

```
$ picotool otp set -e 0x408 0x3005
```

3. Then, write the "S" and "P" characters:

```
$ picotool otp set -e 0x430 0x5053
```

4. Then, write the "O" and "O" characters:

```
$ picotool otp set -e 0x431 0x4f4f
```

5. Then, write the "N" character:

```
$ picotool otp set -e 0x432 0x4e
```

6. Finally, enable the valid override to use the new values (bit 8 marks the `VOLUME_LABEL` override as valid, and bit 22 marks the `OTP_DATA_USB_WHITE_LABEL_ADDR` override as valid):

```
$ picotool otp set -r OTP_DATA_USB_BOOT_FLAGS 0x400100
```

7. To put your changes into effect, reboot the device:

```
$ picotool reboot -u
```

5.8. UART Boot

UART boot is a minimal interface for bootstrapping a flashless RP2350 from a simple host, such as another microcontroller. It is available by default on a blank device, so it allows RP2350 to be deployed into the field on multi-device boards without loading firmware or programming OTP bits in advance.

To select UART boot, drive QSPI `CSn` low (BOOTSEL mode) and drive QSPI `SD1` high. The bootrom checks these signals shortly after device reset is released. UART `TX` appears on QSPI `SD2`, and UART `RX` appears on QSPI `SD3`.

The UART mode is `8n1`: one start bit, eight data bits, no parity, one stop bit. Data within each UART frame is sent and received LSB-first. The baud rate is fixed at 1 Mbaud.

5.8.1. Baud Rate and Clock Requirements

The nominal baud rate for UART boot is 1 Mbaud, divided from a nominal 48 MHz system clock frequency. UART boot uses the USB PLL to derive the system clock and UART baud clock, so you must either provide a crystal or drive a stable clock into the crystal oscillator `XIN` pad. The host baud rate must match the RP2350 baud rate within 3%.

By default the crystal is assumed to be 12 MHz, but the `BOOTSEL_PLL_CFG` and `BOOTSEL_XOSC_CFG` OTP locations override this to achieve a nominal 48 MHz system clock from any supported crystal. The same OTP configuration is used for both USB and UART boot.

 **TIP**

You may drive a somewhat faster or slower clock into `XIN` without any OTP configuration, if you scale your UART baud rate appropriately. The permissible range is 7.5 to 16 MHz on `XIN`, limited by the PLL VCO frequency range.

5.8.2. UART Boot Shell Protocol

After the bootrom samples QSPI `CSn` and `SD1`, there will be a delay of several milliseconds as the bootrom goes through some necessary steps such as switching from the ring oscillator to the PLL, and erasing SRAM before releasing it to the Non-secure UART bootloader.

The UART bootloader signals it is ready to begin by printing the ASCII splash string `RP2350`. In bytes, this is `0x52, 0x50, 0x32, 0x33, 0x35, 0x30`.

Before sending any commands, you must send a special knock sequence to unlock the interface. This is a measure to

avoid transient effects due to noise on GPIOs and ensure the host and device are initially well-synchronised. The sequence is: **0x56, 0xff, 0x8b, 0xe4**. This is the RP2040 UF2 family ID, chosen as a well-known magic number. Any sequence of bytes ending with this four-byte sequence is detected.

A UART boot shell command is always in the host-to-device direction (RP2350 receives), and consists of a single command byte, optionally followed by a 32-byte write payload. RP2350 responds with an optional 32-byte read payload followed by an echo of the command byte. You should wait for the command echo before sending the next command.

The supported commands are:

Command (ASCII)	Command (hex)	Description
n	0x6e	No-op. Do nothing, and report back when you've done it. Used to ping the interface when recovering lost synchronisation. Echoes the command byte, ' n '.
w	0x77	Write a 32-byte payload to the current value of the read/write pointer. Increment the address pointer by 32. Echoes the command byte, ' w ', once all 32 bytes are written to memory.
r	0x72	Read a 32-byte payload from the current value of the read/write pointer. Increment the address pointer by 32. Echoes the command byte, ' r ', after transmitting the 32-byte read payload.
c	0x63	Clear the read/write pointer. The pointer resets to the first location in SRAM 0x20000000 , and you can begin a new read or write sequence from there. Echoes the command byte, ' c '.
x	0x78	Execute the payload that has been written to memory. Echoes the command byte, ' x ', and then reboots, passing a RAM boot search window spanning all of main SRAM. If a valid binary was successfully written into SRAM before sending this command, it will execute.

Unrecognised commands are echoed with no other effect. More commands may be added in future versions.

5.8.3. UART Boot Programming Flow

1. Reset or power down the RP2350 device.
2. Drive **CSn** low to select BOOTSEL, and **SD1** high to select UART.
3. Release the reset or power up the device.
4. Wait for the splash string to be transmitted on QSPI **SD2 (TX)**.
5. Transmit the knock sequence **0x56, 0xff, 0x8b, 0xe4** on QSPI **SD3 (RX)**
6. Send a '**n**' nop command to ensure the interface is awake; if there is no reply, send the knock sequence again.
7. Send '**w**' commands until your entire write payload transfers.
8. (Optional) Send a '**c**' clear command to reset the address pointer, and then send '**r**' read commands to read back and verify the payload.
9. Send an '**x**' execute command to attempt to run the payload.

There is no feedback from UART boot after echoing the final '**x**' command. At this point the device reboots to attempt a RAM image boot on the data loaded by the Non-secure UART bootloader. If the RAM image boot fails, the bootrom falls through to the next boot source, continuing the normal boot flow. Maintaining **CSn** driven low and **SD1** driven high will cause the bootrom to fall through back to UART boot a second time, re-sending the UART splash screen: this indicates the bootrom failed to recognise the UART boot binary.

5.8.4. Recovering from Stuck Interface

Noise on the GPIOs may cause the UART boot shell to stop replying to commands, for example because it thinks the host is part way through a write payload, and the host thinks that it is not. To resynchronise to the start of the next command:

1. Wait 1 ms for the link to quiesce
2. Send 33 '`n`' NOP commands (size of longest command)
3. Wait 1 ms and flush your receive data
4. Send 1 '`n`' NOP command and confirm the device responds with an echoed NOP

5.8.5. Requirements for UART Boot Binaries

A UART boot binary is a normal RAM binary. It must have a valid `IMAGE_DEF` in order for the boot path to recognise it as a bootable binary. The search window for the `IMAGE_DEF` is the whole of SRAM, but it's recommended to place it close to the beginning, because the bootrom searches linearly forward for the beginning of the `IMAGE_DEF`.

The maximum size for a UART boot binary is the entirety of main SRAM: 520 kiB, or 532 480 bytes.

UART boot only supports loading to the start of SRAM, so your binary must be linked to run at address `0x20000000`. Sparse loading is not supported: your program must load as a single flat binary image.

All security requirements relating to RAM image boot apply to UART boot too. If secure boot is enabled, your binary must be signed. Likewise, if OTP anti-rollback versioning is in effect, your binary's rollback version must be no lower than the version number stored in OTP.

5.9. Launching Code On Processor Core 1

As described in the introduction to [Section 5.3](#), after reset, processor core 1 "sleeps (WFE with `SCR.SLEEPDEEP` enabled) and remains asleep until woken by user code, via the mailbox".

If you are using the SDK then you can simply use the `multicore_launch_core1` function to launch code on processor core 1. However this section describes the procedure to launch code on processor core 1 yourself.

The procedure to start running on processor core 1 involves both cores moving in lockstep through a state machine coordinated by passing messages over the inter-processor FIFOs. This state machine is designed to be robust enough to cope with a recently reset processor core 1 which may be anywhere in its boot code, up to and including going to sleep. As result, the procedure may be performed at any point after processor core 1 has been reset (either by system reset, or explicitly resetting just processor core 1).

The following C code is the simplest way to describe the procedure:

```
// values to be sent in order over the FIFO from core 0 to core 1
//
// vector_table is value for VTOR register
// sp is initial stack pointer (SP)
// entry is the initial program counter (PC) (don't forget to set the thumb bit!)
const uint32_t cmd_sequence[] =
    {0, 0, 1, (uintptr_t) vector_table, (uintptr_t) sp, (uintptr_t) entry};

uint seq = 0;
do {
    uint cmd = cmd_sequence[seq];
    // always drain the READ FIFO (from core 1) before sending a 0
    if (!cmd) {
        // discard data from read FIFO until empty
```

```

        multicore_fifo_drain();
        // execute a SEV as core 1 may be waiting for FIFO space
        __sev();
    }
    // write 32 bit value to write FIFO
    multicore_fifo_push_blocking(cmd);
    // read 32 bit value from read FIFO once available
    uint32_t response = multicore_fifo_pop_blocking();
    // move to next state on correct response (echo-d value) otherwise start over
    seq = cmd == response ? seq + 1 : 0;
} while (seq < count_of(cmd_sequence));

```

5.10. Boot Metadata Details

5.10.1. Blocks And Block Loops

Blocks consist of a fixed 32-bit header, one or more *items*, a 32-bit relative offset to the next block, and a fixed 32-bit footer. All multi-byte values within a block are little-endian. Blocks must start on a word-aligned boundary, and the total size is always an exact number of words (a multiple of four bytes).

The final item in a block must be of type `PICOBIN_BLOCK_ITEM_2BS_LAST`, which encodes the total word count of the block's items: that is, the size of the block's contents in bytes – excluding the header, footer, the `LAST` item itself, and relative link – divided by four. For example, in a minimal `IMAGE_DEF` with only an `IMAGE_TYPE`, the `LAST` item would encode a length of 1 word, which includes the `IMAGE_TYPE` item and excludes everything else.

The 32-bit relative link allows multiple blocks to form a linked list, for example, to append a new block with a signature to an image which already contains an unsigned `IMAGE_DEF` block. To be valid, this linked list must eventually link back to the first block in the list, forming a closed *block loop*; failure to close the loop results in the entire linked list being ignored. The loop rule is used to detect orphaned blocks which are the tail of a list that was partially erased.

Due to RAM restrictions in the boot path, size of blocks is limited to 640 bytes for `PARTITION_TABLEs` and 384 bytes for `IMAGE_DEFs`. Blocks larger than this will be ignored.

The format is little-endian, and has sizes and values as follows:

Word	Size	Value
0	0x4	0xffffded3 (<code>MAGIC_HEADER</code>)
1	0x1	<p><code>size_flag</code>:1 item type:7 (<code>size_flag</code> = 0 means 1 byte size, 1 means 2 byte size)</p> <ul style="list-style-type: none"> • 0x1 : size low byte • 0x1 : size hi byte/item type specific data • 0x1 : item type specific data
1 + s0	0x1	<p><code>size_flag</code>:1 item type:7 (<code>size_flag</code> = 0 means 1 byte size, 1 means 2 byte size)</p> <ul style="list-style-type: none"> • 0x1 : size low byte • 0x1 : size hi byte/item type specific data • 0x1 : item type specific data
1 + s0 + s1	0x1	<p>0xff (size_flag = 1, item type = <code>BLOCK_ITEM_LAST</code>)</p> <ul style="list-style-type: none"> • 0x2 : other items' size (s1 + s2) • 0x1 : 0

Word	Size	Value
2 + s0 + s1	0x4	relative position in bytes of next block MAGIC_HEADER relative to this block's MAGIC_HEADER . this forms a loop, so a single block loop has 0 here.
3 + s0 + s1	0x4	0xab123579 (MAGIC_FOOTER)

IMAGE_DEF and **PARTITION_TABLE** blocks are recognised by their first item being an **IMAGE_DEF** or **PARTITION_TABLE** item.

Constants describing blocks can be found in the SDK in https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/common/boot_picobin_headers/include/boot/picobin.h.

5.10.2. Common Block Items

The following items may appear in a **IMAGE_DEF** or a **PARTITION_TABLE** Block.

5.10.2.1. VERSION item

A major/minor version number for the binary, 32 bits total, plus optionally a 16-bit rollback version and a list of OTP rows which can be read to determine the (thermometer-coded) minimum major rollback version which this device will allow to be installed. The major and minor are always present, whereas the rollback version and OTP row list are generally only included if rollback protection is required.

Note: The rollback version and OTP row list are only valid for **IMAGE_DEF**s.

Each OTP row entry indicates the row number (1 through 4095 inclusive) of the first in a group of 3 OTP rows. The three OTP rows are read through the raw read alias, combined with a bitwise majority vote, and then the index of the most-significant 1 bit determines the version number. So, a single group of three rows can encode rollback versions from 0 to 23 inclusive, or, when all 24 bits are set, an indeterminate version of at least 24. Each additional entry adds a further group of 3 rows which increases the maximum version by 24.

There is no requirement for different OTP row entries to be contiguous in OTP. They should not overlap, though the bootrom does not need to check this (the boot signing tool may).

For this entry to be considered valid, the number of available bits in the indicated OTP rows must be *strictly greater than* the rollback version. This means that it is always possible to determine that the device's minimum rollback version is greater than the rollback version indicated in this block, even if we don't know the full list of OTP rows used by later major versions.

If the number of OTP row entries is zero, there is no rollback version for this block.

The major/minor version are used to disambiguate which is newer out of two binaries with the same major rollback version. For example, to select which A/B image to boot from. when no major rollback version is specified, A/B comparisons will treat the missing major version as zero, but no rollback check will be performed.

```
.byte PICOBIN_BLOCK_ITEM_1BS_VERSION // item type
.byte 2 + ((num_row_entries != 0) + num_row_entries + 1) / 2
.byte 0
.byte num_otp_row_entries
.hword minor
.hword major
.hword rollback      // optional, present if num_otp_row_entries != 0
.hword otp_row        // optional
```

5.10.2.2. HASH_DEF item

Optional item with information about what how to hash:

```
.byte PICOBIN_BLOCK_ITEM_1BS_HASH_DEF // item type (with one byte size)
.byte 0x1                      // word size of this item
.byte 0                      // pad
.byte PICOBIN_HASH_SHA256 // hash type
.hword block_words_hashed // number of words of block (not including START marker) hashed
.hwrd 0                      // pad
```

`block_words_hashed` must include this item if using this item for a signature.

The most recent LOAD_MAP item (see [Section 5.10.3.2](#)) that defines *what* to hash.

5.10.2.3. HASH_VALUE item

Optional item with hash result (for use when not using signature):

```
.byte PICOBIN_BLOCK_ITEM_1BS_HASH_VALUE // item type (with two byte size)
.hword 0x1 + n          // word size of this item (note if the hash value is included, n > 0)
.byte 0                      // pad
.word [n]                  // hash value which can be used for checking the binary is AOK
```

Note whilst a SHA-256 is 16 words, you can include less (down to 1 word).

This HASH_VALUE item is paired with the most recent HASH_DEF item (see [Section 5.10.2.2](#)) which defines what is being hashed.

5.10.2.4. SIGNATURE item

Optional item with cryptographic signature:

```
.byte PICOBIN_BLOCK_ITEM_SIGNATURE // item type
.hword 0x22                      // word size of this item
.byte PICOBIN_SIGNATURE_SECP256K1 // signature type
.word [16]                      // secp256k1 public key
.word [16]                      // signature of hash
```

5.10.3. Image Definition

5.10.3.1. IMAGE_DEF item

The `IMAGE_DEF` item must be the first item within an Image Definition:

```
.byte PICOBIN_BLOCK_ITEM_1BS_IMAGE_TYPE // item type with 1-byte size
.byte 0x1                      // word size of this item
.hword                      // image type flags
```

The flags are defined in the SDK in https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/common/boot_picobin_headers/include/boot/picobin.h, but are summarized here:

Image Type

(bits 0-3)

IMAGE_TYPE_INVALID	0
IMAGE_TYPE_EXE	1 // Image is executable
IMAGE_TYPE_DATA	2 // Image is valid, but is not intended to be executed.

The remaining bits are specific to the Image Type; values are only currently defined for the EXE Image Type:

EXE Security

(bits 4-5)

EXE_SECURITY_UNSPECIFIED	0
EXE_SECURITY_NS	1
EXE_SECURITY_S	2

EXE CPU

(bits 8-10)

EXE_CPU_ARM	0
EXE_CPU_RISCV	1

EXE CHIP

(bits 12-14)

EXE_CHIP_RP2040	0
EXE_CHIP_RP2350	1

TBYB

(bit 15)

TBYB	1
------	---

Other bits are reserved for future use, and should be set to zero.

5.10.3.2. LOAD_MAP item

Optional item with a similar representation to the ELF program header. This is used both to define content to hash, and also to "load" data before image execution (e.g. a secure flash binary can be loaded into RAM prior to both sig check and execution)

The load map is a collection of runtime address, physical address, size and flags.

Note on terms:

physical address

this is where the data is stored in the logical address space (e.g. the start of a flash image even if stored in a partition could have a physical address of `0x10000000`). (The closest ELF concept is LMA)

runtime address

what the address of the data is at runtime (The closest ELF concept is VMA)

storage address

this is an absolute location where the data is stored (not necessarily the same as physical address for flash when partitions are in use)

Note that we use physical address here, not storage address, since this data is written by a tool working on the ELF which will not know where the binary will finally be stored in flash.

This serves several purposes:

1. For a `packaged` binary, this tells the bootrom where to load the code
2. For a signed binary, the runtime addresses and `size` indicate code/data that must be included in the hash to be verified.

NOTE

If the `runtime_address` is in equal to the `storage_address`, then data is never copied, it is just hashed in place.

```
.byte PICOBIN_BLOCK_ITEM_LOAD_MAP           // item type (1 or 2 byte size)
.hword 1 + num_entries * 3                 // word size of this item
.byte absolute | num_entries               // top bit == absolute flag
if (!absolute) {
    for each entry {
        .word storage_address_rel          // relative to this load map item
        .word runtime_address_physical
        .word size
    }
} else {
    for each entry {
        .word storage_address_physical
        .word runtime_address_physical
        .word runtime_end_address_physical
    }
}
```

If the `storage_address/storage_address_rel` is `0x00000000`, then zeros are copied into `runtime_address` → `runtime_address + size`

NOTE

For hashing purposes, the size of the zero range is hashed, not that many zeros.

5.10.3.2.1. XIP Pinning

A load-map entry (with `storage_address == runtime_address`) MUST be present for part of `XIP_RAM` if your binary has code in the `XIP_CACHE`; this causes the cache to be pinned before entering the binary. it is fine for this to be:

5.10.3.3. VECTOR_TABLE item

Optional item with location of the vector table. for Arm binaries, the entry_point/initial_sp will be taken from here if present (unless there is an `ENTRY_POINT`). Note if there is no `ENTRY_POINT` or `VECTOR_TABLE`, then a `VECTOR_TABLE` at the start of the image is assumed.

```
.byte PICOBIN_BLOCK_ITEM_VECTOR_TABLE // item type
.byte 0x2                         // word size of this item
.byte 0                           // pad
.byte 0                           // pad
.word vector_table                // location (runtime address)
```

 **NOTE**

This item applies to ARM binaries only. RISC-V binaries do not have a public vector table.

5.10.3.4. ENTRY_POINT item

Optional block with info on initial PC, SP

```
.byte PICOBIN_BLOCK_ITEM_ENTRY_POINT // item type
.byte 0x3/0x4                     // word size of this item
.byte 0                           // pad
.byte 0                           // pad
.word entry_point                // (runtime address)
.word initial_sp
.word initial_sp_lim             // (optional)
```

5.10.3.5. ROLLING_WINDOW_DELTA item

Optional item that allow for binaries that aren't intended to be run at `0x10000000`. Note that this delta is in addition to the roll resulting from the binary being stored in a different partition in flash.

```
.byte PICOBIN_BLOCK_ITEM_1BS_ROLLING_WINDOW_DELTA // item type with 1-byte size
.byte 0x2                         // word size of this item
.hword 0                           // pad
.word delta                        // delta
```

The delta is the number of bytes into the image that `0x10000000` should be mapped.

 **NOTE**

`ROLLING_WINDOW_DELTA` is ignored for non-flash binaries.

5.10.4. Partition Tables

Partition tables allow flash access permissions to be specified per partition (and for un-partitioned space). Access permissions are defined for secure code, Non-secure code, and "NSBoot" which refers to the boot loader (and PICOBOOT)

These permissions are of course advisory to secure code, however they are respected by `flash_op()`, the PICOBOOT flash access commands, and UF2 downloads.

5.10.4.1. PARTITION_TABLE item

```
.byte PICOBIN_BLOCK_ITEM_2BS_PARTITION_TABLE_TYPE // item type with 2-byte size
.hword // word size of this item
.byte // top bit set if singleton; low 4 is partition count
.word unpartitioned_space_permissions_and_flags

[for each partition]
    .word permissions_and_location
    .word permissions_and_flags
    [if has_id]
        .word id_lo
        .word id_hi
    [for 0..num_extra_families]
        .word family_id
    [if has_name]
        .byte top_bit_reserved; lower_7)_bits: n_name_len_bytes
        [for 1..(n_name_len_bytes + 4) & ~3]
            .byte name_char
```

5.10.4.2. Partition location, permissions and flags

Two common words are stored in the partition table for both un-partitioned space, and each partition. These common words describe the size/location, along with access permissions and various flags.

The permission fields are repeated in both words, hence the two words are `location_and_permissions` and `flags_and_permissions`.

Table 470. Permission Fields. 'P' means the field applies to partitions, 'U' means the field applies to un-partitioned space, however the word 'partition' is always used in the description

Mask	AppliesTo	Description
0x04000000u	'P' 'U'	<code>PERMISSION_S_R_BITS</code>
		If set, the partition is readable by secure code. See Section 5.1.3
0x08000000u	'P' 'U'	<code>PERMISSION_S_W_BITS</code>
		If set, the partition is writable by secure code. See Section 5.1.3
0x10000000u	'P' 'U'	<code>PERMISSION_NS_R_BITS</code>
		If set, the partition is readable by Non-secure code. See Section 5.1.3
0x20000000u	'P' 'U'	<code>PERMISSION_NS_W_BITS</code>
		If set, the partition is writable by Non-secure code. See Section 5.1.3
0x40000000u	'P' 'U'	<code>PERMISSION_NSBOOT_R_BITS</code>
		If set, the partition is readable by NSBOOT (i.e. boot loader). secure code. See Section 5.1.3
0x80000000u	'P' 'U'	<code>PERMISSION_NSBOOT_W_BITS</code>
		If set, the partition is writable by NSBOOT (i.e. boot loader). secure code. See Section 5.1.3

Table 471. Location Fields. 'P' means the field applies to partitions, 'U' means the field applies to un-

Mask	AppliesTo	Description
0x00001ffffu	'P' 'U'	LOCATION_FIRST_SECTOR_BITS The sector number (0-4095) of the first sector in the partition (a sector is 4 KiB)
0x03ffe000u	'P' 'U'	LOCATION_LAST_SECTOR_BITS The sector number (0-4095) of the last sector in the partition (a sector is 4 KiB)

Table 472. Flags Fields. 'P' means the field applies to partitions, 'U' means the field applies to unpartitioned space, however the word "partition" is always used in the description

Mask	AppliesTo	Description
0x00000001u	'P'	FLAGS_HAS_ID_BITS If set, the partition has a 64 bit identifier
0x00000006u	'P'	FLAGS_LINK_TYPE_BITS The type of link stored in the partition: <ul style="list-style-type: none"> 0x0 - None 0x1 - A_PARTITION : This is a "B" partition, and The LINK_VALUE field stores the partition number of the corresponding "A" partition 0x2 - OWNER : This is an "A" partition, and the LINK_VALUE field stores the partition number of the owning partition (which should also be an "A" partition).
0x00000078u	'P'	FLAGS_LINK_VALUE_BITS If LINK_TYPE is non zero, then this field holds the partition number of the linked partition.
0x00000180u	'P'	FLAGS_ACCEPTS_NUM_EXTRA_FAMILIES_BITS 0-3 the number of extra non-standard UF2 family ids the partition accepts.
0x00000200u	'P'	FLAGS_NOT_BOOTABLE_ARM_BITS If set then this partition is marked non-bootable on Arm, and will be ignored during Arm boot. Setting this for non Arm bootable partitions can improve boot performance.
0x00000400u	'P'	FLAGS_NOT_BOOTABLE_RISCV_BITS If set then this partition is marked non-bootable on RISC-V, and will be ignored during RISC-V boot. Setting this for non RISC-V bootable partitions can improve boot performance.
0x00000800u	'P'	FLAGS_UF2_DOWNLOAD_AB_NON_BOOTABLE_OWNER_AFFINITY
0x00001000u	'P'	FLAGS_HAS_NAME_BITS If set, the partition has a name.
0x00002000u	'P' 'U'	FLAGS_UF2_DOWNLOAD_NO_REBOOT_BITS If set, the RP2350 will not reboot after dragging a UF2 into this partition.
0x00004000u	'P' 'U'	FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2040_BITS If set, a UF2 with the RP2040 family id 0xe48bff56 may be downloaded into this partition.
0x00008000u	'U'	FLAGS_ACCEPTS_DEFAULT_FAMILY_ABSOLUTE_BITS If set for un-partitioned spaced, a UF2 with the ABSOLUTE family id 0xe48bff57 may be downloaded onto the RP2350 and will be written at the addresses specified in the UF2 without regard to partition locations. Partition-defined flash access permissions will however still be respected (i.e. the UF2 download will fail if it needs to write over a read-only region of flash).
0x00010000u	'P' 'U'	FLAGS_ACCEPTS_DEFAULT_FAMILY_DATA_BITS If set, a UF2 with the DATA family id 0xe48bff58 may be downloaded into this partition.

partitioned space, however the word "partition" is always used in the description

Mask	AppliesTo	Description
0x00020000u	'P' 'U'	FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2350_ARM_S_BITS If set, a UF2 with the RP2350_ARM_S family id 0xe48bff59 may be downloaded into this partition.
0x00040000u	'P' 'U'	FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2350_RISCV_BITS If set, a UF2 with the RP2350_RISC_V family id 0xe48bff5a may be downloaded into this partition.
0x00080000u	'P' 'U'	FLAGS_ACCEPTS_DEFAULT_FAMILY_RP2350_ARM_NS_BITS If set, a UF2 with the RP2350_ARM_NS family id 0xe48bff5b may be downloaded into this partition.
0x03f00000u	'P' 'U'	reserved; should be 0

5.10.5. Minimum Viable Metadata

A minimum amount of metadata (i.e. a valid **IMAGE_DEF**) must be embedded in any binary for the bootrom to recognise it as a valid program image, as opposed to, for example, blank flash contents or a disconnected flash device. This must appear within the first 4 kB of a flash image, or anywhere in a RAM or OTP image.

Unlike RP2040, there is no requirement for flash binaries to have a checksummed "boot2" flash setup function at flash address 0. The RP2350 bootrom performs a simple best-effort XIP setup during flash scanning, and a flash-resident program can continue executing in this state, or can choose to reconfigure the QSPI interface at a later time for best performance.

5.10.5.1. Minimum Arm IMAGE_DEF

Assuming **CRIT1.SECURE_BOOT_ENABLE** is clear, the minimum valid **IMAGE_DEF** is the following 20-byte sequence:

```

.word 0xfffffded3 // PICOBIN_BLOCK_MARKER_START
.word 0x10210142 // .byte PICOBIN_BLOCK_ITEM_1BS_IMAGE_TYPE
    // .byte 0x1 (1 word in size)
    // .hword PICOBIN_IMAGE_TYPE_IMAGE_TYPE_AS_BITS(EXE) |
    //      PICOBIN_IMAGE_TYPE_EXE_SECURITY_AS_BITS(S) |
    //      PICOBIN_IMAGE_TYPE_EXE_CPU_AS_BITS(ARM) |
    //      PICOBIN_IMAGE_TYPE_EXE_CHIP_AS_BITS(RP2350)
.word 0x000001ff // .byte PICOBIN_BLOCK_ITEM_2BS_LAST
    // .hword 1 (1 word of total contents)
    // .byte 0 (padding)
.word 0x00000000 // relative link to next in block loop -- 0 for 1-block loop
.word 0xab123579 // PICOBIN_BLOCK_MARKER_END

```

The **.word** directives indicate 32-bit little-endian values that should appear verbatim in your program image, for example, by assembling the above snippet into an object file. Since the above block does not specify an explicit entry point, the bootrom will assume the binary starts with a Cortex-M vector table, and enter via the reset handler and initial stack pointer specified in that table (offsets +4 and +0 bytes into the table). An explicit vector table pointer can be provided by a **PICOBIN_BLOCK_ITEM_1BS_VECTOR_TABLE** item, or the entry point can be specified directly by a **PICOBIN_BLOCK_ITEM_1BS_ENTRY_POINT** item.

5.10.5.2. Minimum RISC-V IMAGE_DEF

The minimum valid **IMAGE_DEF** is the following 20-byte sequence:

```
.word 0xffffded3 // PICOBIN_BLOCK_MARKER_START
.word 0x11010142 // .byte PICOBIN_BLOCK_ITEM_1BS_IMAGE_TYPE
    // .byte 0x1 (1 word in size)
    // .hword PICOBIN_IMAGE_TYPE_IMAGE_TYPE_AS_BITS(EXE) |
    //      PICOBIN_IMAGE_TYPE_EXE_CPU_AS_BITS(RISCV) |
    //      PICOBIN_IMAGE_TYPE_EXE_CHIP_AS_BITS(RP2350)
.word 0x000000ff // .byte PICOBIN_BLOCK_ITEM_2BS_LAST
    // .hword 1 (1 word of total contents)
    // .byte 0 (padding)
.word 0x00000000 // relative link to next in block loop -- 0 for 1-block loop
.word 0xab123579 // PICOBIN_BLOCK_MARKER_END
```

The `.word` directives indicate 32-bit little-endian values that should appear verbatim in your program image, for example, by assembling the above snippet into an object file. Since the above block does not specify an explicit entry point, the bootrom will enter the binary at its lowest address, which is the default behaviour on RISC-V. This default entry point can be overridden by a `PICOBIN_BLOCK_ITEM_1BS_ENTRY_POINT` item. Note that `PICOBIN_BLOCK_ITEM_1BS_VECTOR_TABLE` is not valid on RISC-V, as unlike Cortex-M the RISC-V vector table does not define the program entry point.

Chapter 6. Power

6.1. Power Supplies

6.2. Core Voltage Regulator

RP2350 provides an on-chip voltage regulator for its digital core supply (**DVDD**). The regulator requires a 2.7V to 5.5V input supply (**VREG_VIN**), allowing DVDD to be generated directly from a single lithium ion cell, or a USB power supply. A separate, nominally 3.3V, low noise supply (**VREG_AVDD**) is required for the regulator's analogue control circuits. The regulator supports both switching and linear modes of regulation, allowing efficient operation at both high and low loads.

To allow the chip to start up, the regulator is enabled by default, and will power-up as soon as its supplies are available. The regulator starts in switching mode, with a nominal 1.1V output, but its operating mode and output voltage can be changed once the chip is out of reset. The output voltage can be set in the range 0.55V to 3.30V, and the regulator can supply up to 200mA.

Although intended for the chip's digital core supply (**DVDD**), the regulator can be used for other purposes if DVDD is powered directly from an external power supply.

6.2.1. Operating Modes

The regulator has the following three modes of operation.

6.2.1.1. Normal Mode

In Normal mode, the regulator operates in a switching mode, and can supply up to 200mA. Normal mode is used for **P0.x** power states, when the chip's switched core is powered on. The regulator must be in Normal Mode *before* the core supply current is allowed to exceed 1mA. The regulator starts up in Normal mode when its input supplies are first applied.

6.2.1.2. Low Power Mode

In Low Power mode, the regulator operates in a linear mode, and can only supply up to 1mA. Low Power mode can be used for **P1.x** power states, where the chip's switched core is powered off. The core supply current must be less than 1mA *before* the regulator is moved to Low Power mode. The regulator's output voltage is limited to 1.3V in Low Power mode.

⚠ CAUTION

In Low Power mode, the output of the regulator is directly connected to **DVDD**. It is not possible to disconnect the regulator from **DVDD** in this mode. Do not put the regulator into Low Power mode if **DVDD** is being powered from an external supply.

6.2.1.3. High Impedance Mode

In High Impedance mode, the regulator is disabled, its power consumption is minimised, and its outputs are set to a high impedance state. This mode should only be used if the digital core supply (DVDD) is provided by an external regulator. If the on-chip regulator is supplying DVDD, entering high impedance mode causes a reset event, returning the on-chip regulator to Normal mode.

6.2.2. Software Control

The regulator can be directly controlled by software, but must first be unlocked by writing a 1 to the **UNLOCK** field in the **VREG_CTRL** register. This is a one time operation and software control can not be subsequently disabled by writing a 0 to the field. Once unlocked, the regulator can be controlled via the **VREG** register.

The regulator's operating mode defaults to Normal, at initial power-up or after a reset event, but can be switched to High Impedance by writing a 1 to the **VREG** register's **HIZ** field. The regulator's output voltage can be set by writing to the register's **VSEL** field, see the **VREG** register description for details on available settings. To prevent accidental overvoltage, the output voltage is limited to 1.3V unless the **DISABLE_VOLTAGE_LIMIT** field in the **VREG_CTRL** is set. The output voltage defaults to 1.1V at initial power-on or after a reset event.

The **UPDATE_IN_PROGRESS** field in the **VREG** register is set while the regulator's operating mode or output voltage are being updated. When **UPDATE_IN_PROGRESS** is set, writes to the register are ignored.

NOTE

It is not possible to place the regulator in Low Power mode under software control, as the load current will exceed 1mA when software is running.

CAUTION

The regulator's output voltage can be varied between 0.55V and 3.3V, but RP2350 may not operate reliably with its digital core supply (DVDD) at a voltage other than 1.1V.

6.2.3. Power Manager Control

The regulator's operating mode and output voltage can also be controlled by the Power Manager. Power Manager control is typically used when the chip enters or exits a low power (**P1.x**) state, when software may not be running.

In addition to Normal and High Impedance modes, Power Manager control allows the regulator to be placed in Low Power mode. By default, the regulator switches to Low Power mode when entering a low power (**P1.x**) state, and returns to Normal mode when returning to a normal (**P0.x**) state.

The operating mode and output voltage in the low power state are set by the values in the **VREG_LP_ENTRY** register. And the operating mode and output voltage to be used when the chip has returned to a normal state are set by values in the **VREG_LP_EXIT** register. The registers contain an additional **MODE** field that allows you to select Low Power mode, see [Table 473](#).

The values in the registers must be written by software *before* requesting a transition to a low power state, as software will not be running during or after the transition. The actual transitions to and from the low power state are handled by the Power Manager. Once the chip has returned to a normal state, software can be run and the regulator controlled directly. The values in the **VREG** register reflect the regulator's current operating mode and output voltage once the chip has returned to a normal state.

Table 473. Voltage Regulator Mode Select when under Software Control

Mode	HIZ	MODE
Normal ^a	0	0
Low Power	0	1

High Impedance	1	X
----------------	---	---

^a the regulator will be in normal mode at initial power-on or following a reset event

⚠ CAUTION

Low Power mode should only be used when the regulator is providing the chip's digital core supply (DVDD), as the regulator's low power output is directly connected to DVDD.

6.2.4. Status

To determine the status of the regulator, read the [VREG_STS](#) register, which contains two fields:

- [VOUT_OK](#) indicates whether the voltage regulator's output is being correctly regulated. At power-on, [VOUT_OK](#) remains low until the regulator has started up and the output voltage reaches the [VOUT_OK](#) assertion threshold ([VOUT_OK_{TH,ASSERT}](#)). It then remains high until the voltage drops below the [VOUT_OK](#) deassertion threshold ([VOUT_OK_{TH,DEASSERT}](#)), remaining low until the output voltage is above the assertion threshold again. [VOUT_OK_{TH,ASSERT}](#) is nominally 90% of the selected output voltage, 0.99V if the selected output voltage is 1.1V, and [VOUT_OK_{TH,DEASSERT}](#) is nominally 87% of the selected output voltage, 0.957V if the selected output voltage is 1.1V. See [\[section-core-voltage-regulator-detailed-specifications\]](#) for details.
- [STARTUP](#) is high when the regulator is starting up, and remains high until the regulator's operating mode or output voltage are changed, either by software or the Power Manager

ℹ NOTE

Adjusting the output voltage to a higher voltage will cause [VOUT_OK](#) to go low until the assertion threshold for the higher voltage is reached. [VOUT_OK](#) will also go low if the regulator is placed in High Impedance mode.

6.2.5. Current Limit

The voltage regulator includes a current limit to prevent the load current exceeding the maximum rated value. The output voltage will not be regulated and will drop below the selected value when the current limit is active. See [\[section-core-voltage-regulator-detailed-specifications\]](#) for details.

6.2.6. Over Temperature Protection

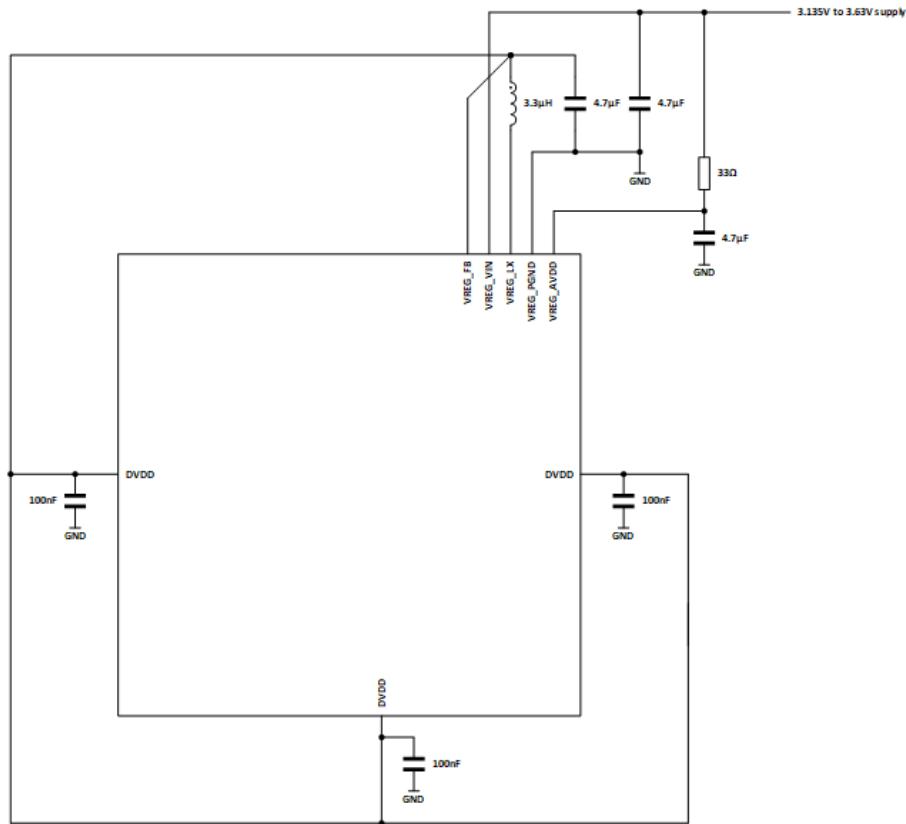
The voltage regulator will terminate regulation, and disable its power transistors, if the transistor junction temperature rises above a threshold set by the [HT_TH](#) field in the [VREG_CTRL](#) register. The regulator will restart regulation when the transistor junction temperature drops to approximately 20°C below the temperature threshold.

6.2.7. Application Circuit

The regulator requires two external power supplies, the input supply ([VREG_VIN](#)), and a separate low noise supply for its analogue control circuits ([VREG_AVDD](#)). [VREG_VIN](#) must be in the range 2.7V to 5.5V, and [VREG_AVDD](#) must be in the range 3.135V to 3.63V.

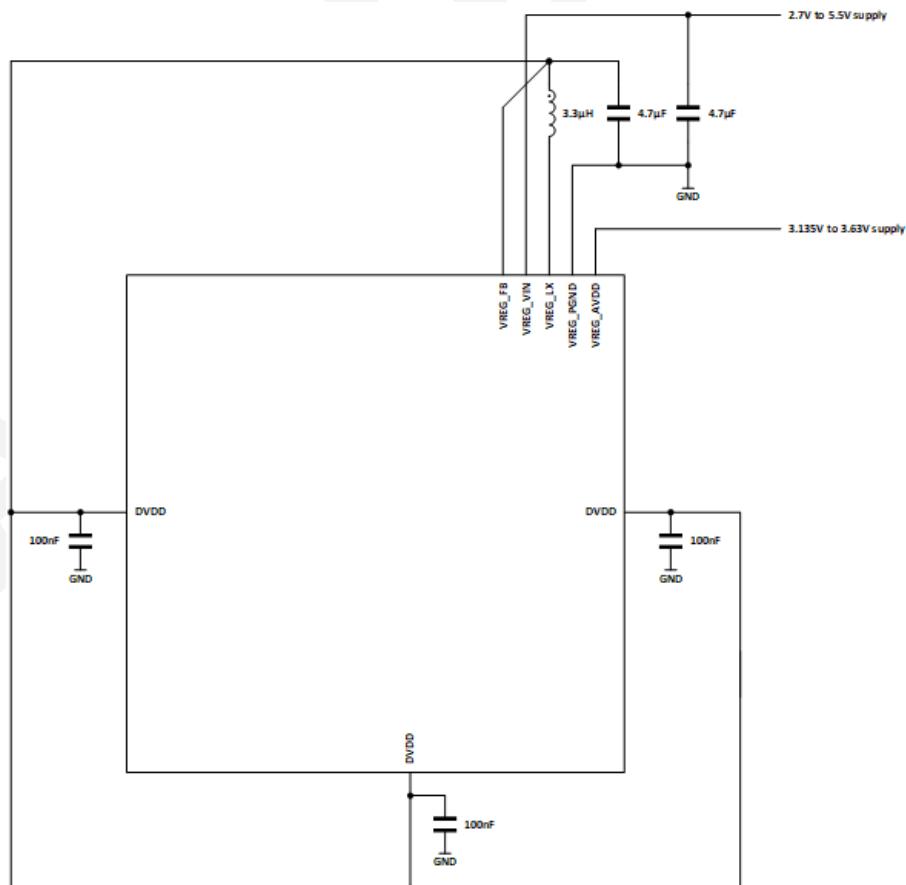
If [VREG_VIN](#) is limited to the range 3.135V to 3.63V, a single combined supply can be used for both [VREG_VIN](#) and [VREG_AVDD](#). This approach is shown in [Figure 17](#). Care must be taken to minimise noise on [VREG_AVDD](#).

Figure 17. core voltage regulator with combined supplies



Alternatively, to support input voltages above 3.63V, **VREG_VIN** and **VREG_AVDD** can be powered separately. This is shown in Figure 18.

Figure 18. core voltage regulator with separate supplies



6.2.8. External Components and PCB layout requirements

The most critical part of an RP2350 PCB layout is the core voltage regulator. This should be placed first on any board design and these **guidelines strictly followed**.

Figure 19. Regulator section of the Raspberry Pi Pico 2 schematic

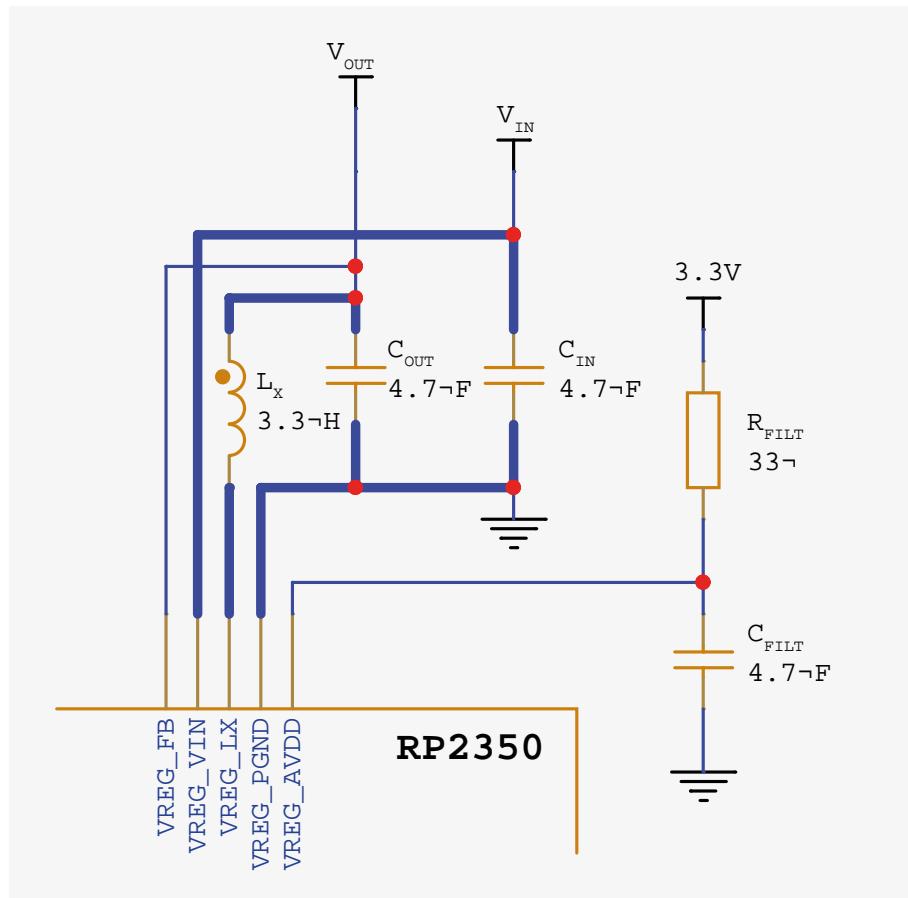
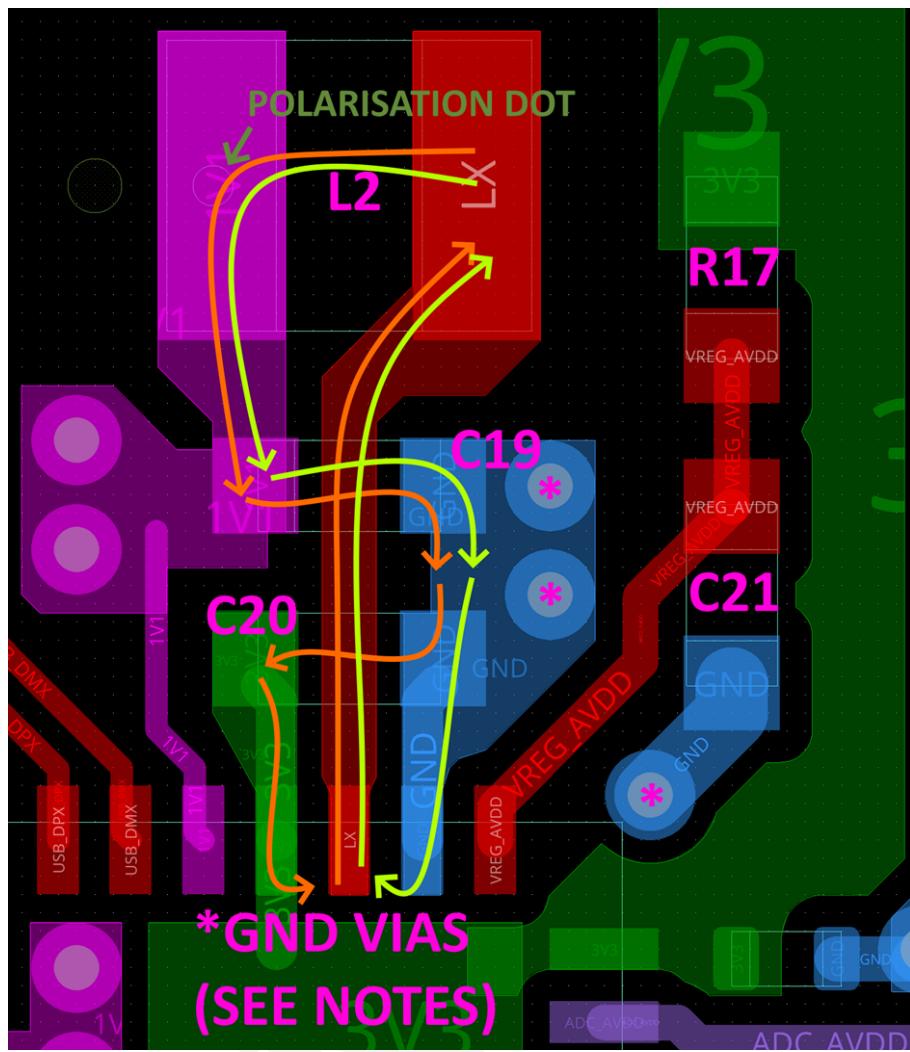


Figure 20. Regulator section of the Raspberry Pi Pico 2 PCB layout showing the high current paths for each of the regulator's switching phases



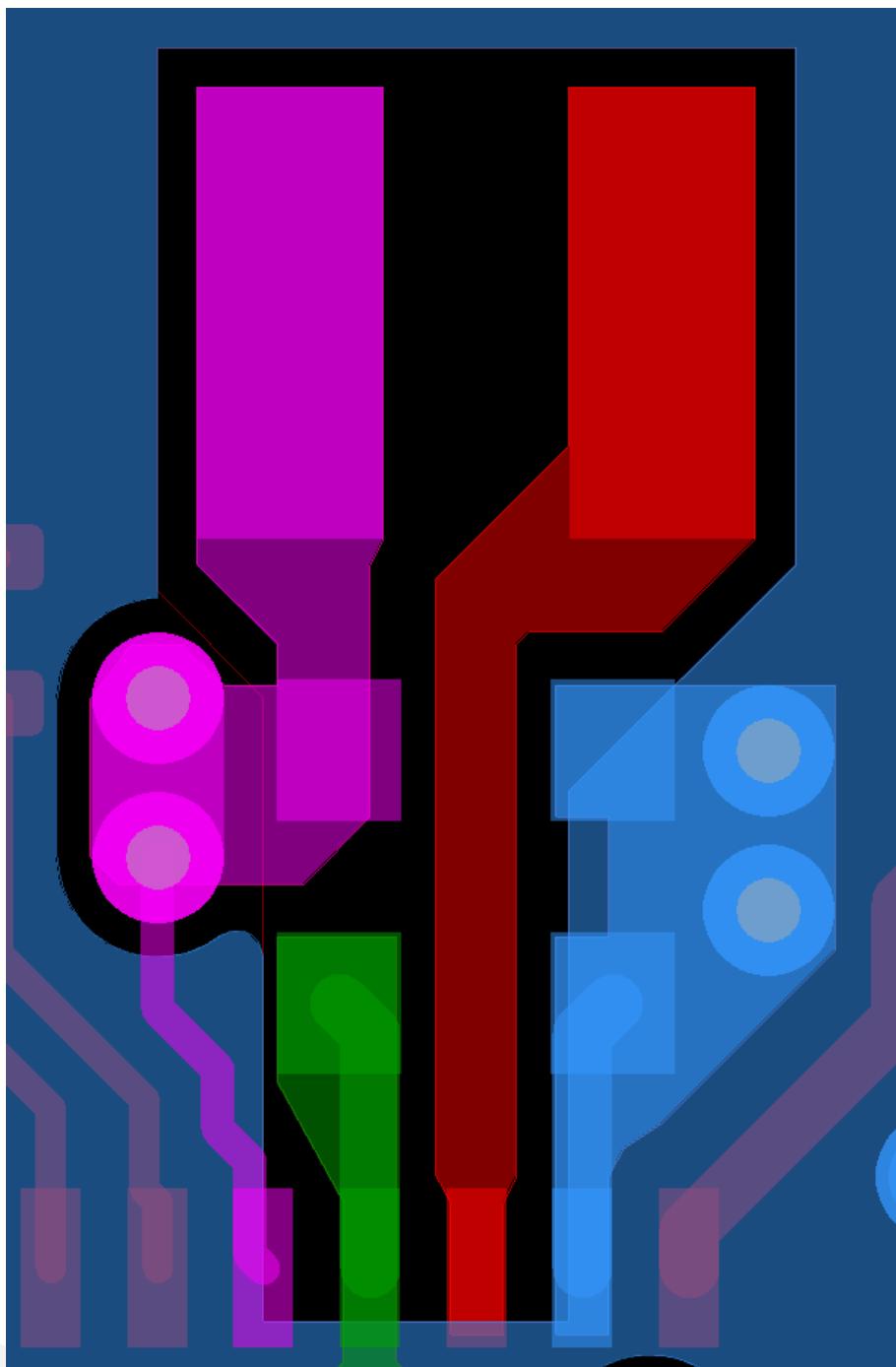
Designers should follow the above schematic Figure 19 and layout Figure 20 as closely as possible as this has had the most verification and is considered our 'best practice' layout. This circuit design is present on the Raspberry Pi Pico 2 and RP2350 reference design (see [Hardware design with RP2040, Minimal Design Example](#)) and both of these designs are made available in either Cadence Allegro or Kicad formats respectively. Figure 20 shows the regulator layout on the top layer of the Raspberry Pi Pico 2 PCB. The bottom layer under the regulator is a ground plane which connects to the QFN GND central pad.

6.2.8.1. Layout Recommendations

- VREG_AVDD is a noise sensitive signal and must be RC filtered as per Figure 19.
 - Avoid doing anything that may couple noise into VREG_AVDD.
 - C_{IN} needs its own separate GND via / low impedance path back to the RP2350 GND pad.
- The orange and lime green arrows in Figure 20 show the high current paths for each of the regulator's switching phases. It is critical to keep the loop area of these current paths as small and low-impedance as possible, while also keeping them isolated (i.e. only connect to main GND at one point).
 - Follow this layout as closely as possible.
 - Do not place any of $C_{IN}/L_x/C_{OUT}$ on the opposite side of the PCB.
- Reduce parasitics on the VREG_LX node.

- On the top layer make sure to cut away any extra copper underneath the inductor, cut back copper near the VREG_LX trace where possible.
 - For a multi-layer board (4 or more layers) please cut away any copper immediately underneath L_x /VREG_LX node. For example, [Figure 21](#) illustrates this.
- The GND via placement is critical.
 - There must be a short-as-possible, low impedance GND path back to the Raspberry Pi Pico 2} QFN GND pad from the high-current GND at one single point (using 2 adjacent vias to reduce the impedance).
 - C_{FILT} must also have a low impedance and short-as-possible path back to the QFN GND pad (Do not share any GND vias with the C_{IN}/C_{OUT} high current GND).
- The VREG_FB pin should be fed from the output of C_{OUT} , avoiding routing directly underneath L_x .
- C_{OUT} is critical for regulator performance and EMI. It must be placed between VREG_VIN and VREG_PGND as close to the pins as practically possible.
 - In addition to C_{OUT} , for best performance we recommend a second 4.7 μ F capacitor is used on the V_{OUT} net, located on the bottom edge of the package (DVDD pin 23 on the QFN-60). Do not place this near L_x/C_{OUT} .

Figure 21. Cut-out
beneath LX/VREG_LX
net on layer 2 of 4 (or
more) layer PCBs



6.2.8.2. Component Values

- C_{IN} should be at least $4.7\mu F$ and have a maximum parasitic resistance of $50m\Omega$.
- C_{OUT} must be $4.7\mu F \pm 20\%$ with a maximum parasitic resistance of $250m\Omega$ and a maximum inductance of $6nH$.
- L_x must be fully shielded, $3.3\mu H \pm 20\%$ and with a maximum DC resistance of $250m\Omega$. Saturation current should be at least $1.5A$. The inductor must be marked for polarity (see [Figure 22](#)) and placed on the layout as indicated in [Figure 20](#). As discussed below, we recommend the TBD.

6.2.8.3. Regulator Sensitivities

The RP2350 regulator has a few sensitivities:

1. The VREG_AVDD supply is noise sensitive.
2. Efficiency is quite sensitive to inductance roll-off with inductor current, so an inductor with low roll-off is required for best operation (generally the higher saturation current the better).
3. Even with nominally fully shielded inductors, leakage magnetic field coupling into the loop formed by the output VREG_LX node through the inductor and output capacitor (C_{OUT}) seems to affect the regulator control loop and output voltage. Field orientation (and hence inductor orientation) matters - the inductor has to be “the right way around” to make sure the regulator operates properly especially at higher output currents and for higher load transients. This necessitates an inductor with marked polarity.

To meet the above requirements, Raspberry Pi have worked with Abracon to create a custom 2.0x1.6mm 3.3 μ H polarity-marked inductor, part number TBD (see [Figure 22](#) and [Figure 22](#)). These will be available in general distribution in time, but for now please contact Raspberry Pi to request samples / production volumes. Please note that Raspberry Pi is still working with the regulator IP vendor to fully verify and qualify the regulator and custom inductor.

Figure 22. TBD inductor with orientation marking, showing current and magnetic field directions

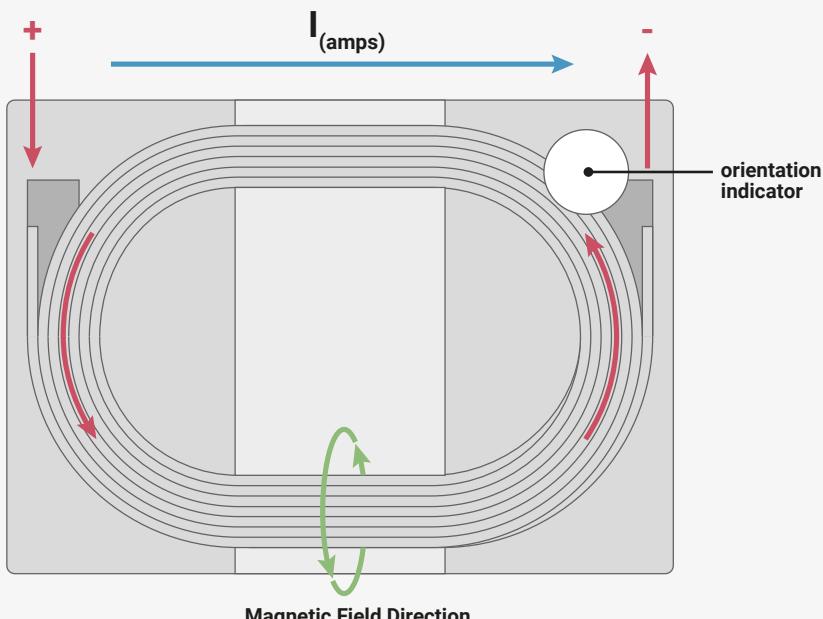


Figure 23. Dimensions of the TBD inductor



6.2.9. List of Registers

The voltage regulator shares a register address space with other power management subsystems in the always-on domain. This address space is referred to as **POWMAN** elsewhere in this document, and a complete list of **POWMAN** registers is provided in [Section 6.6](#). For reference information on **POWMAN** registers associated with the voltage regulator is repeated

here.

The **POWMAN** registers start at a base address of **0x40100000** (defined as **POWMAN_BASE** in the SDK).

- [VREG_CTRL](#)
- [VREG_STS](#)
- [VREG](#)
- [VREG_LP_ENTRY](#)
- [VREG_LP_EXIT](#)

6.3. Power Control

6.3.1. Changes from RP2040

RP2350 manages power consumption using SLEEP ([Section 6.3.4](#)) and DORMANT ([Section 6.3.5](#)) states which stop the clocks to various components. This eliminates switching current so only the leakage current remains.

RP2350 includes SLEEP and DORMANT modes for dynamic power control, and introduces power domains ([Section 6.4](#)), which allow power to be removed from various components on chip, virtually eliminating the leakage currents, and allowing lower power modes to be supported.

Power domains are controlled via the Power Manager ([Section 6.5](#)).

6.3.2. Dynamic Power Control

RP2350 provides a range of options for reducing dynamic power:

- Top-level clock gating of individual peripherals and functional blocks
- Automatic control of top-level clock gates based on processor sleep state
- On-the-fly changes to system clock frequency or system clock source (e.g. switch to internal ring oscillator, and disable PLLs and crystal oscillator)
- Zero-dynamic-power DORMANT state, waking on GPIO event or AON-Timer IRQ

All digital logic on RP2350 is in a single core power domain. The following options are available for static power reduction:

- Placing memories into state-retaining power down state
- Power gating on peripherals that support this, e.g. ADC, temperature sensor

6.3.3. Top-level Clock Gates

Each clock domain (for example, the system clock) may drive a large number of distinct hardware blocks, not all of which may be required at once. To avoid unnecessary power dissipation, each individual endpoint of each clock (for example, the UART system clock input) may be disabled at any time.

Enabling and disabling a clock gate is glitch-free. If a peripheral clock is temporarily disabled, and subsequently re-enabled, the peripheral will be in the same state as prior to the clock being disabled. No reset or reinitialisation should be required.

Clock gates are controlled by two sets of registers: the **WAKE_ENx** registers (starting at **WAKE_EN0**) and **SLEEP_ENx** registers (starting at **SLEEP_EN0**). These two sets of registers are identical at the bit level, each possessing a flag to control each clock endpoint. The **WAKE_EN** registers specify which clocks are enabled whilst the system is awake, and the **SLEEP_EN**

registers select which clocks are enabled while the processor is in the SLEEP state (Section 6.3.4).

The two Cortex-M0+ processors do not have externally-controllable clock gates. Instead, the processors gate the clocks of their subsystems autonomously, based on execution of `WFI`/`WFE` instructions, and external Event and IRQ signals.

6.3.4. SLEEP State

RP2350 enters the SLEEP state when all of the following are true:

- Both processors are asleep (e.g. in a `WFE` or `WFI` instruction)
- The system DMA has no outstanding transfers on any channel

RP2350 exits the SLEEP state when either processor is awoken by an interrupt.

When in the SLEEP state, the top-level clock gates are masked by the `SLEEP_ENx` registers (starting at `SLEEP_EN0`), rather than the `WAKE_ENx` registers (starting at `WAKE_EN0`). This permits more aggressive pruning of the clock tree when the processors are asleep.

 **NOTE**

Though it is possible for a clock to be enabled during SLEEP and disabled outside of SLEEP, this is generally not useful

For example, if the system is sleeping until a character interrupt from a UART, the entire system except for the UART can be clock-gated (`SLEEP_ENx` = all-zeroes except for `CLK_SYS_UART0` and `CLK_PERI_UART0`). This includes system infrastructure such as the bus fabric.

When the UART asserts its interrupt and wakes a processor, RP2350 leaves SLEEP mode and switches back to the `WAKE_ENx` clock mask. At the minimum, this should include the bus fabric and the memory devices containing the processor's stack and interrupt vectors.

A system-level clock request handshake holds the processors off the bus until the clocks are re-enabled.

6.3.5. DORMANT State

The DORMANT state is a true zero-dynamic-power sleep state, where all clocks (and all oscillators) are disabled. The system can awake from the DORMANT state upon a GPIO event (high/low level or rising/falling edge), or an AON-Timer interrupt: this restarts one of the oscillators (either ring oscillator or crystal oscillator) and ungates the oscillator output once it is stable. System state is retained, so code execution resumes immediately upon leaving the DORMANT state.

If relying on the AON-Timer (Section 12.10) to wake from the DORMANT state, the AON-Timer must have some external clock source. The AON-Timer accepts clock frequencies as low as 1Hz.

DORMANT does not halt PLLs. To avoid unnecessary power dissipation, software should power down PLLs before entering the DORMANT state, and power up and reconfigure the PLLs again after exiting.

The DORMANT state is entered by writing a keyword to the DORMANT register in whichever oscillator is active: ring oscillator (Section 8.3) or crystal oscillator (Section 8.2). If both are active, the one providing the processor clock must be stopped last because it will stop software from executing.

6.3.6. Memory Power Down

The main system memories (`SRAM0` → `SRAM9`, mapped to bus addresses `0x20000000` to `0x20081fff`), as well as the USB DPRAM, can be powered down via the `MEMPOWERDOWN` register in the SYSCFG registers (see Section 12.15.2). When powered down, memories retain their current contents, but cannot be accessed. Static power is reduced.

NOTE

RP2350 supports additional memory power-down modes that remove all power to memory; contents are not retained. For more information, see (Section 6.4) and (Section 6.5).

CAUTION

Memories must not be accessed when powered down. Doing so can corrupt memory contents.

When powering a memory back up, a 20ns delay is required before accessing the memory again.

The XIP cache (see Section 4.4) can also be powered down, with `CTRL.POWER_DOWN`. The XIP hardware will not generate cache accesses whilst the cache is powered down. Note that this is unlikely to produce a net power savings if code continues to execute from XIP, due to the comparatively high voltages and switching capacitances of the external QSPI bus.

6.3.7. Programmer's Model

6.3.7.1. Sleep

The `hello_sleep` example, https://github.com/raspberrypi/pico-playground/blob/master/sleep/hello_sleep/hello_sleep.c, demonstrates sleep mode. The `hello_sleep` application (and underlying functions) takes the following steps:

1. Switches all clocks in the system to run from XOSC.
2. Configures an alarm in the AON-Timer for 10 seconds in the future.
3. Sets the AON-timer clock as the only clock running in sleep mode using the `SLEEP_ENx` registers (see `SLEEP_EN0`).
4. Enables deep sleep in the processor.
5. Calls `_wfi` on processor, which will put the processor into deep sleep until woken by the AON-Timer interrupt.
6. After 10 seconds, the AON-Timer interrupt clears the alarm and then calls a user supplied callback function.
7. The callback function ends the example application.

NOTE

To enter sleep mode, you must enable deep sleep on both proc0 and proc1, call `_wfi`, and ensure the DMA is stopped.

`hello_sleep` makes use of functions in `pico_sleep` of the `Pico Extras`. In particular, `sleep_goto_sleep_until` puts the processor to sleep until woken up by an AON-Timer time assumed to be in the future.

6.3.7.2. Dormant

The `hello_dormant` example, https://github.com/raspberrypi/pico-playground/blob/master/sleep/hello_dormant/hello_dormant.c, demonstrates dormant mode. The example takes the following steps:

1. Switches all clocks in the system to run from XOSC.
2. Configures a GPIO interrupt for the `dormant_wake` hardware, which can wake both the ROSC and XOSC from dormant mode.
3. Puts the XOSC into dormant mode, which stops all processor execution (and all other clocked logic on the chip) immediately.

4. When GPIO 10 goes high, the XOSC restarts and program execution continues.

hello_dormant uses `sleep_goto_dormant_until_pin` under the hood:

Pico Extras: https://github.com/raspberrypi/pico-extras/blob/master/src/p2_common/pico_sleep/sleep.c Lines 134 - 155

```

134 void sleep_goto_dormant_until_pin(uint gpio_pin, bool edge, bool high) {
135     bool low = !high;
136     bool level = !edge;
137
138     // Configure the appropriate IRQ at IO bank 0
139     assert(gpio_pin < NUM_BANK0_GPIOS);
140
141     uint32_t event = 0;
142
143     if (level && low) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_LEVEL_LOW_BITS;
144     if (level && high) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_LEVEL_HIGH_BITS;
145     if (edge && high) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_EDGE_HIGH_BITS;
146     if (edge && low) event = IO_BANK0_DORMANT_WAKE_INTE0_GPIO0_EDGE_LOW_BITS;
147
148     gpio_set_dormant_irq_enabled(gpio_pin, event, true);
149
150     _go_dormant();
151     // Execution stops here until woken up
152
153     // Clear the irq so we can go back to dormant mode again if we want
154     gpio_acknowledge_irq(gpio_pin, event);
155 }
```

6.4. Power Domains

6.4.1. Isolation

GPIOs use latched isolation gates to retain the state of outputs when the switched-core is powered down as described in (Section 9.7). GPIO isolation is not removed as soon as the domain is powered because the IO logic is reset and therefore the GPIOs would return to their default states which may be undesirable. So, GPIO isolation is removed in software after the IO logic has been configured. Two GPIO isolation controls are provided to allow isolation to be removed in two stages. GPIOs are allocated to either of the isolation controls. This also allows GPIOs to be individually unisolated.

Normal operation:

- assign GPIOs to the isolation controls by writing to the POWMAN_GPIO_ISO_ASSIGN0/1/2 registers
- on switched-core power-down, all GPIOs are clamped to their current state
- after switched-core power-up the two isolation controls are released at appropriate times by software writes to POWMAN_GPIO_ISO

Individual GPIO unisolation:

- assign all GPIOs to the GROUP0 isolation control
- on switched-core power-down, all GPIOs are clamped to their current state
- after switched-core power-up release the POWMAN_GPIO_ISO_GROUP1 control
- then GPIOs can be unisolated individually by assigning them to GROUP1 using the POWMAN_GPIO_ISO_ASSIGN0/1/2 registers

6.5. Power Manager

6.5.1. Core Power Domains

To support low power modes, RP2350 is split into five core power domains.

- AON - always on power domain - a small amount of logic that is always powered on when chip's core supply (DVDD) is supplied with power
- SWCORE - switched core logic - the remaining core logic functions, including processors, bus fabric, peripherals etc.
- SRAM0 - SRAM power domain 0 - the lower half of the large SRAM banks
- SRAM1 - SRAM power domain 1 - the upper half of the large SRAM banks, and the scratch SRAMs
- XIP - XIP cache SRAM and Boot RAM

Logic in the AON domain controls the power state of the other power domains, as it will always be powered on. The other power domains can be powered on or off independently, apart from the XIP domain which must always be powered when the SWCORE domain is powered. SRAM's that are powered on, retain their contents when the switched core is powered off

[\[diagram-power-domain-overview\]](#) gives an overview of the power domains.

6.5.2. Power States

RP2350 can operate in a number of power states, depending on which domains are powered on or off. Power states have names in the form **Ps.m** where:

- s indicates the state of the switched core (SWCORE) domain
- m is a 3 bit binary representation of the memory power domains, in the order XIP, SRAM0, SRAM1
- 0 = On / 1 = Off

P0.m states, where the switched core is powered on, are *Normal Operation* states. **P1.m** states, where the switched core is powered off, are *Low Power* states

[Table 474](#) shows the available power states.

Table 474. supported power states

Power State	Description	AON	SWCORE	XIP	SRAM0	SRAM1
P0.0	Normal Operation	on	on	on	on	on
P0.1	Normal Operation (SRAM1 off)	on	on	on	on	off
P0.2	Normal Operation (SRAM0 off)	on	on	on	off	on
P0.3	Normal Operation (SRAM0 & SRAM1 off)	on	on	on	off	off
P1.0	Low Power	on	off	on	on	on
P1.1	Low Power (SRAM1 off)	on	off	on	on	off
P1.2	Low Power (SRAM0 off)	on	off	on	off	on
P1.3	Low Power (SRAM0 & SRAM1 off)	on	off	on	off	off
P1.4	Low Power (XIP off)	on	off	off	on	on
P1.5	Low Power (XIP & SRAM1 off)	on	off	off	on	off
P1.6	Low Power (XIP & SRAM0 off)	on	off	off	off	on

Power State	Description	AON	SWCORE	XIP	SRAM0	SRAM1
P1.7	Low Power (XIP & SRAM0 & SRAM1 off)	on	off	off	off	off
OFF	Not Powered	off	off	off	off	off

In the **OFF** state, the chip has no external power, and all domains are unpowered. The chip moves from **OFF** to **P0.0** automatically as soon as external power is applied.

6.5.3. Power State Transitions

Transitions between power states can be initiated by software, hardware, or via the chip's debug subsystem. Once initiated, transitions are managed by autonomous power sequencers in chip's AON power domain. The power sequencers can be configured, in a limited way, via the [SEQ_CFG](#) register. They sequencers can also be observed and controlled, again in a limited way, via the RP-AP registers in the chip's debug subsystem. The registers are described in [Section 3.5.10](#).

Valid power state transitions are as follows.

- all transitions from one P0.x state (switched core powered) to another P0.x state (switched core powered) if they increase or decrease the size of powered SRAM
- all transitions from a P0.x state (switched core powered) to a P1.x state (switched core unpowered) except transitions that would result in an unpowered SRAM becoming powered
- all transitions from a P1.x state (switched core unpowered) to a P0.x state (switched core powered) except transitions that would result in a powered SRAM becoming unpowered

Transitions from one P1.x state (switched core unpowered) to another P1.x state (switched core unpowered) are not supported and will be prevented by the hardware implementation. A request to move to an unsupported power state will result [STATE.BAD_SW_REQ](#) being set. No power state change will take place.

Valid transitions are shown in the table below.

Table 475. valid power state transitions

From	To												
P0.0		<i>P0.1</i>	<i>P0.2</i>	<i>P0.3</i>	<i>P1.0</i>	<i>P1.1</i>	<i>P1.2</i>	<i>P1.3</i>	<i>P1.4</i>	<i>P1.5</i>	<i>P1.6</i>	<i>P1.7</i>	
P0.1	<i>P0.0</i>			<i>P0.3</i>		<i>P1.1</i>		<i>P1.3</i>		<i>P1.5</i>		<i>P1.7</i>	
P0.2	<i>P0.0</i>			<i>P0.3</i>			<i>P1.2</i>	<i>P1.3</i>			<i>P1.6</i>	<i>P1.7</i>	
P0.3	<i>P0.0</i>	<i>P0.1</i>	<i>P0.2</i>					<i>P1.3</i>				<i>P1.7</i>	
P1.0	<i>P0.0</i>												
P1.1	<i>P0.0</i>	<i>P0.1</i>											
P1.2	<i>P0.0</i>		<i>P0.2</i>										
P1.3	<i>P0.0</i>	<i>P0.1</i>	<i>P0.2</i>	<i>P0.3</i>									
P1.4	<i>P0.0</i>												
P1.5	<i>P0.0</i>	<i>P0.1</i>											
P1.6	<i>P0.0</i>		<i>P0.2</i>										
P1.7	<i>P0.0</i>	<i>P0.1</i>	<i>P0.2</i>	<i>P0.3</i>									

6.5.3.1. Invalid Power State Transitions

Invalid power state transitions are not supported in the hardware. If an invalid transition is requested the Power

Manager will still register the request in `STATE.REQ` but will also set the `POWMAN_BAD_REQ` flag. It will then implement the power-up requests and ignore the power down requests. To do nothing would risk entering an unrecoverable lock-up state.

Invalid transitions are:

- transitions between P0.1 and P0.2 states, as they just swap which SRAM domain is powered without increasing or decreasing the size of powered SRAM
- transitions from a P0.x state (switched core powered) to a P1.x state (switched core unpowered) that would result in an unpowered SRAM becoming powered
- transitions from a P1.x state (switched core unpowered) to a P0.x state (switched core powered) that would result in a powered SRAM becoming unpowered
- transitions to a P0.x state where $x > 3$ (switched-core powered, xip unpowered)

These transitions are shown in the table below.

Table 476.
unsupported power
state transitions

From	To											
P0.0												
P0.1			<i>P0.2</i>		<i>P1.0</i>		<i>P1.2</i>		<i>P1.4</i>		<i>P1.6</i>	
P0.2		<i>P0.1</i>			<i>P1.0</i>	<i>P1.1</i>			<i>P1.4</i>	<i>P1.5</i>		
P0.3					<i>P1.0</i>	<i>P1.1</i>	<i>P1.2</i>		<i>P1.4</i>	<i>P1.5</i>	<i>P1.6</i>	
P1.0		<i>P0.1</i>	<i>P0.2</i>	<i>P0.3</i>								
P1.1			<i>P0.2</i>	<i>P0.3</i>								
P1.2		<i>P0.1</i>		<i>P0.3</i>								
P1.3												
P1.4		<i>P0.1</i>	<i>P0.2</i>	<i>P0.3</i>								
P1.5			<i>P0.2</i>	<i>P0.3</i>								
P1.6		<i>P0.1</i>		<i>P0.3</i>								
P1.7												

6.5.4. Initiating Power State Transitions

Power state transitions are initiated in various ways:

- transition from **OFF** to **P0.0** is initiated by deassertion of chip resets
- transitions from **P0.x** to **Px.x** states are initiated in software
- transitions from **P1.x** to **P0.x** states are initiated by GPIO events or the timer alarm

6.5.4.1. Software Initiated Power Transitions

Software transitions are initiated by writing to the `STATE` register.

The `STATE` register comprises:

- `STATE.CURRENT` - indicates the current power state
- `STATE.REQ` - requests a new power state - is written to by software and by hardware wake-up requests
- `STATE.BAD_SW_REQ` - indicates a software initiated req is bad

- `STATE.BAD_HW_REQ` - indicates a hardware initiated req is bad. Should never happen, always returns to P0.0
- `STATE.PWRUP WHILE_WAITING` - there was a powerup event while in `STATE_WAITING`
- `STATE.REQ IGNORED` - write to `STATE_REQ` ignored because of a pending powerup request
- `STATE.WAITING` - indicates the power manager is waiting before starting the state change
- `STATE.CHANGING` - indicates the state change is in progress

The coding of `STATE.CURRENT` & `STATE.REQ` corresponds to the power states defined in the datasheet:

On writing to `STATE.REQ`:

- If there is a pending powerup request then `STATE.REQ IGNORED` is set and no further action is taken
- If the requested state is invalid, then `STATE.BAD_SW_REQ` is set and no further action is taken
- If the switched core is being powered off then `STATE.WAITING` is set until both processors enter `_wfi()`. After which `STATE.CHANGING` will be set, but no processors will be powered up to read the flag at this time
 - If there is a powerup request while in `STATE.WAITING` then `STATE.PWRUP WHILE_WAITING` is set, which can also raise an interrupt to bring the processors out of `_wfi()`. No further action is taken
 - You can get out of `STATE.WAITING` by writing a new request to `STATE_REQ` before both processors have gone into `_wfi()`
- Any state request that isn't powering down the switched core, such as powering up or down SRAM domain 0 or 1 starts immediately. Software should wait until `STATE.CHANGING` has cleared to know the power down sequence. Once the `STATE.CHANGING` flag is cleared `STATE.CURRENT` is updated.
- If powering up software should also wait for `STATE.CHANGING` to make sure everything is powered up before continuing. In practice this is handled by the RP2350 bootrom.

Invalid state transitions are: * any combination of power up and power down requests * any request which would result in power down of XIP/bootRAM and power up of swcore

When powering down swcore, the pad output signals are latched and isolated from the IO logic. This avoids transitions on pads which could potentially corrupt external components. On swcore power up the isolation is not released automatically. The user releases the isolation by writing to `POWMAN_GPIO_ISO` once the IO logic has been configured (see [Section 6.4.1](#)).

When powering down swcore the processors are halted therefore software cannot observe the register updates, but the register is still visible to the debugger if the `CORESIGHT_POWMAN_DBGMODE` control is set.

If XIP/bootRAM|sram0|sram1 remain powered while swcore is powered off, the sram will automatically switch to a low power state and stored data will be retained.

Before transitioning to a switched-core powerdown state (P1.x), software needs to configure:

- the gpio wakeup conditions if required
- the wakeup alarm if required
- the return state of the SRAM0 & SRAM1 domains

6.5.4.2. Hardware Initiated Power State Transitions

There are up to 5 wakeup sources:

- up to 4 gpio level sensitive wakeups
- 1 alarm wakeup

The gpio wakeups are configured by the `POWMAN_PWRUP` register. The wakeups are not enabled until the power sequencer completes the power down operation. The wakeups are level sensitive so the sequencer can initiate a wakeup immediately if one of the gpions switched to the wakeup condition while the power down sequence was running.

The alarm wakeup is configured by writing to the POWMAN_ALARM_SEC_UPPER, POWMAN_ALARM_SEC_LOWER & POWMAN_ALARM_MSEC registers and has a resolution of 1ms. Once set the wakeup alarm is armed by writing POWMAN_TIMER_CFG_ALARM_SET=1 and POWMAN_ALARM_CFG_PWRUP_ON_ALARM=1. If the alarm fires during the powerdown sequence then a power up sequence will start when the power down sequence completes.

The POWMAN_LAST_PWRUP register indicates which event caused the most recent power up.

6.5.4.3. Debugger Initiated Power State Transitions

There are 2 levels of debugger interaction with the power sequencer.

The first is a user feature which allows the debugger to trigger a power up sequence via the coresight_pwrupreq input. This powers all domains (i.e. returns to state P0.0) and also inhibits any further software initiated power state transitions.

When the coresight_pwrupreq is asserted the power sequencer will:

- complete any power state transitions that are in progress
- return to power state P0.0
- assert coresight_pwrupack

If coresight_pwrupreq is deasserted then software initiated power transitions will be able to resume. If a software requested transition is ignored because of coresight_pwrupreq then the user can tell this by:

- Getting a `STATE.REQ_IGNORED` - write to `STATE.REQ` ignored because of a pending powerup request
- Read the `POWMAN_CURRENT_PWRUP_REQS` to find the cause (which will be coresight_pwrupreq)
- Either:
 - Get the debugger to deassert coresight_pwrupreq or
 - Mask out coresight_pwrupreq by setting `DBG_PWRCFG.IGNORE`

NOTE

`DBG_PWRCFG.IGNORE` is useful to test going to sleep with a debugger attached or ignoring coresight_pwrupreq. A debugger will likely leave coresight_pwrupreq set when disconnecting. It would be impossible to go to sleep after this without `DBG_PWRCFG.IGNORE`.

The second is for RP debug of faults in the power sequencer. This is described in the Debug Features section of this specification.

6.5.4.4. Power mode aware GPIO Control

The powman sequencer is able to switch the state of two GPIO outputs on entry to and exit from a P1.x state, i.e. one where the switched core is powered down. This allows external devices to be power aware. Note that the GPIOs switch to indicate the low power state after the core is powered down and switch to indicate the high power state before the core is powered up. This ensures the high power state of the external components always overlaps the high power state of the core. The GPIOs are configured by the `EXT_CTRL0` and `EXT_CTRL1` registers.

6.6. Power Management (POWMAN) Registers

Password protected POWMAN registers require a password (`0x5AFE`) to be written to the top 16 bits to enable the write operation. This protects against accidental writes which could crash the chip untraceably. A write to a protected register using an incorrect password is ignored and sets a flag in the `POWMAN_BADPASSWD` register. A read from a protected register

does not return the password. This protects against erroneous read-modify-write operations.

Protected registers obviously do not have writeable fields in the top 16 bits, however they may have read-only fields in that range.

All registers with address offsets up to and including `0x000000ac` are password protected. Therefore the following writeable registers are unprotected and have 32-bit write access:

- `POWMAN_SCRATCH0` → `POWMAN_SCRATCH7`
- `POWMAN_BOOT0` → `POWMAN_BOOT3`
- `POWMAN_INTR`
- `POWMAN_INTE`
- `POWMAN_INTF`

Table 477. List of POWMAN registers

Offset	Name	Info
0x00	<code>BADPASSWD</code>	Indicates a bad password has been used
0x04	<code>VREG_CTRL</code>	Voltage Regulator Control
0x08	<code>VREG_STS</code>	Voltage Regulator Status
0x0c	<code>VREG</code>	Voltage Regulator Settings
0x10	<code>VREG_LP_ENTRY</code>	Voltage Regulator Low Power Entry Settings
0x14	<code>VREG_LP_EXIT</code>	Voltage Regulator Low Power Exit Settings
0x18	<code>BOD_CTRL</code>	Brown-out Detection Control
0x1c	<code>BOD</code>	Brown-out Detection Settings
0x20	<code>BOD_LP_ENTRY</code>	Brown-out Detection Low Power Entry Settings
0x24	<code>BOD_LP_EXIT</code>	Brown-out Detection Low Power Exit Settings
0x28	<code>LPOSC</code>	Low power oscillator control register.
0x2c	<code>CHIP_RESET</code>	Chip reset control and status
0x30	<code>WATCHDOG</code>	Allows a watchdog reset to reset the internal state of powman in addition to the power-on state machine (PSM). Note that powman ignores watchdog resets that do not select at least the CLOCKS stage or earlier stages in the PSM. If using these bits, it's recommended to set PSM_WDSEL to all-ones in addition to the desired bits in this register. Failing to select CLOCKS or earlier will result in the <code>POWMAN_WATCHDOG</code> register having no effect.
0x34	<code>SEQ_CFG</code>	For configuration of the power sequencer Writes are ignored while <code>POWMAN_STATE_CHANGING=1</code>

Offset	Name	Info
0x38	STATE	<p>This register controls the power state of the 4 power domains. The current power state is indicated in POWMAN_STATE_CURRENT which is read-only. To change the state, write to POWMAN_STATE_REQ. The coding of POWMAN_STATE_CURRENT & POWMAN_STATE_REQ corresponds to the power states defined in the datasheet:</p> <p>bit 3 = SWCORE bit 2 = XIP cache bit 1 = SRAM0 bit 0 = SRAM1 0 = powered up 1 = powered down</p> <p>When POWMAN_STATE_REQ is written, the POWMAN_STATE_WAITING flag is set while the Power Manager determines what is required. If an invalid transition is requested the Power Manager will still register the request in POWMAN_STATE_REQ but will also set the POWMAN_BAD_REQ flag. It will then implement the power-up requests and ignore the power down requests. To do nothing would risk entering an unrecoverable lock-up state. Invalid requests are: any combination of power up and power down requests any request that results in swcore being powered and xip unpowered If the request is to power down the switched-core domain then POWMAN_STATE_WAITING stays active until the processors halt. During this time the POWMAN_STATE_REQ field can be re-written to change or cancel the request. When the power state transition begins the POWMAN_STATE_WAITING_flag is cleared, the POWMAN_STATE_CHANGING flag is set and POWMAN register writes are ignored until the transition completes.</p>
0x3c	POW_FASTDIV	
0x40	POW_DELAY	power state machine delays
0x44	EXT_CTRL0	Configures a gpio as a power mode aware control output
0x48	EXT_CTRL1	Configures a gpio as a power mode aware control output
0x4c	EXT_TIME_REF	Select a GPIO to use as a time reference, the source can be used to drive the low power clock at 32kHz, or to provide a 1ms tick to the timer, or provide a 1Hz tick to the timer. The tick selection is controlled by the POWMAN_TIMER register.
0x50	LPOSC_FREQ_KHZ_INT	Informs the AON-timer of the integer component of the clock frequency when running off the LPOSC.
0x54	LPOSC_FREQ_KHZ_FRAC	Informs the AON-timer of the fractional component of the clock frequency when running off the LPOSC.
0x58	XOSC_FREQ_KHZ_INT	Informs the AON-timer of the integer component of the clock frequency when running off the XOSC.
0x5c	XOSC_FREQ_KHZ_FRAC	Informs the AON-timer of the fractional component of the clock frequency when running off the XOSC.
0x60	SET_TIME_63T048	
0x64	SET_TIME_47T032	

Offset	Name	Info
0x68	SET_TIME_31T016	
0x6c	SET_TIME_15T00	
0x70	READ_TIME_UPPER	
0x74	READ_TIME_LOWER	
0x78	ALARM_TIME_63T048	
0x7c	ALARM_TIME_47T032	
0x80	ALARM_TIME_31T016	
0x84	ALARM_TIME_15T00	
0x88	TIMER	
0x8c	PWRUP0	<p>4 GPIO powerup events can be configured to wake the chip up from a low power state.</p> <p>The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event</p> <p>The number of gpios available depends on the package option.</p> <p>An invalid selection will be ignored</p> <p>source = 0 selects gpio0</p> <p>.</p> <p>.</p> <p>source = 47 selects gpio47</p> <p>source = 48 selects qspi_ss</p> <p>source = 49 selects qspi_sd0</p> <p>source = 50 selects qspi_sd1</p> <p>source = 51 selects qspi_sd2</p> <p>source = 52 selects qspi_sd3</p> <p>source = 53 selects qspi_sclk</p> <p>level = 0 triggers the pwrup when the source is low</p> <p>level = 1 triggers the pwrup when the source is high</p>
0x90	PWRUP1	<p>4 GPIO powerup events can be configured to wake the chip up from a low power state.</p> <p>The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event</p> <p>The number of gpios available depends on the package option.</p> <p>An invalid selection will be ignored</p> <p>source = 0 selects gpio0</p> <p>.</p> <p>.</p> <p>source = 47 selects gpio47</p> <p>source = 48 selects qspi_ss</p> <p>source = 49 selects qspi_sd0</p> <p>source = 50 selects qspi_sd1</p> <p>source = 51 selects qspi_sd2</p> <p>source = 52 selects qspi_sd3</p> <p>source = 53 selects qspi_sclk</p> <p>level = 0 triggers the pwrup when the source is low</p> <p>level = 1 triggers the pwrup when the source is high</p>

Offset	Name	Info
0x94	PWRUP2	<p>4 GPIO powerup events can be configured to wake the chip up from a low power state.</p> <p>The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event</p> <p>The number of gpios available depends on the package option. An invalid selection will be ignored</p> <p>source = 0 selects gpio0</p> <p>.</p> <p>.</p> <p>source = 47 selects gpio47</p> <p>source = 48 selects qspi_ss</p> <p>source = 49 selects qspi_sd0</p> <p>source = 50 selects qspi_sd1</p> <p>source = 51 selects qspi_sd2</p> <p>source = 52 selects qspi_sd3</p> <p>source = 53 selects qspi_sclk</p> <p>level = 0 triggers the pwrup when the source is low</p> <p>level = 1 triggers the pwrup when the source is high</p>
0x98	PWRUP3	<p>4 GPIO powerup events can be configured to wake the chip up from a low power state.</p> <p>The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event</p> <p>The number of gpios available depends on the package option. An invalid selection will be ignored</p> <p>source = 0 selects gpio0</p> <p>.</p> <p>.</p> <p>source = 47 selects gpio47</p> <p>source = 48 selects qspi_ss</p> <p>source = 49 selects qspi_sd0</p> <p>source = 50 selects qspi_sd1</p> <p>source = 51 selects qspi_sd2</p> <p>source = 52 selects qspi_sd3</p> <p>source = 53 selects qspi_sclk</p> <p>level = 0 triggers the pwrup when the source is low</p> <p>level = 1 triggers the pwrup when the source is high</p>
0x9c	CURRENT_PWRUP_REQ	<p>Indicates current powerup request state</p> <p>pwrup events can be cleared by removing the enable from the pwrup register. The alarm pwrup req can be cleared by clearing timer.alarm_enab</p> <p>0 = chip reset, for the source of the last reset see POWMAN_CHIP_RESET</p> <p>1 = pwrup0</p> <p>2 = pwrup1</p> <p>3 = pwrup2</p> <p>4 = pwrup3</p> <p>5 = coresight_pwrup</p> <p>6 = alarm_pwrup</p>

Offset	Name	Info
0xa0	LAST_SWCORE_PWRUP	Indicates which pwrup source triggered the last switched-core power up 0 = chip reset, for the source of the last reset see POWMAN_CHIP_RESET 1 = pwrup0 2 = pwrup1 3 = pwrup2 4 = pwrup3 5 = coresight_pwrup 6 = alarm_pwrup
0xa4	DBG_PWRCFG	
0xa8	BOOTDIS	Tell the bootrom to ignore the BOOT0..3 registers following the next RSM reset (e.g. the next core power down/up). If an early boot stage has soft-locked some OTP pages in order to protect their contents from later stages, there is a risk that Secure code running at a later stage can unlock the pages by powering the core up and down. This register can be used to ensure that the bootloader runs as normal on the next power up, preventing Secure code at a later stage from accessing OTP in its unlocked state. Should be used in conjunction with the OTP BOOTDIS register.
0xac	DBGCONFIG	
0xb0	SCRATCH0	Scratch register. Information persists in low power mode
0xb4	SCRATCH1	Scratch register. Information persists in low power mode
0xb8	SCRATCH2	Scratch register. Information persists in low power mode
0xbc	SCRATCH3	Scratch register. Information persists in low power mode
0xc0	SCRATCH4	Scratch register. Information persists in low power mode
0xc4	SCRATCH5	Scratch register. Information persists in low power mode
0xc8	SCRATCH6	Scratch register. Information persists in low power mode
0xcc	SCRATCH7	Scratch register. Information persists in low power mode
0xd0	BOOT0	Scratch register. Information persists in low power mode
0xd4	BOOT1	Scratch register. Information persists in low power mode
0xd8	BOOT2	Scratch register. Information persists in low power mode
0xdc	BOOT3	Scratch register. Information persists in low power mode
0xe0	INTR	Raw Interrupts
0xe4	INTE	Interrupt Enable
0xe8	INTF	Interrupt Force
0xec	INTS	Interrupt status after masking & forcing

POWMAN: BADPASSWD Register

Offset: 0x00

Table 478.
BADPASSWD Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Indicates a bad password has been used	WC	0x0

POWMAN: VREG_CTRL Register

Offset: 0x04

Description

Voltage Regulator Control

Table 479.
VREG_CTRL Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	RST_N	returns the regulator to its startup settings 0 - reset 1 - not reset (default)	RW	0x1
14	Reserved.	-	-	-
13	UNLOCK	unlocks the VREG control interface after power up 0 - Locked (default) 1 - Unlocked It cannot be relocked when it is unlocked.	RW	0x0
12	ISOLATE	isolates the VREG control interface 0 - not isolated (default) 1 - isolated	RW	0x0
11:9	Reserved.	-	-	-
8	DISABLE_VOLTAGE_LIMIT	0=not disabled, 1=enabled	RW	0x0
7	Reserved.	-	-	-
6:4	HT_TH	high temperature protection threshold regulator power transistors are disabled when junction temperature exceeds threshold 000 - 100C 001 - 105C 010 - 110C 011 - 115C 100 - 120C 101 - 125C 110 - 135C 111 - 150C	RW	0x5
3:2	Reserved.	-	-	-
1:0	RESERVED	write 0 to this field	RW	0x0

POWMAN: VREG_STS Register

Offset: 0x08

Description

Voltage Regulator Status

Table 480. VREG_STS Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	VOUT_OK	output regulation status 0=not in regulation, 1=in regulation	RO	0x0
3:1	Reserved.	-	-	-
0	STARTUP	startup status 0=startup complete, 1=starting up	RO	0x0

POWMAN: VREG Register

Offset: 0x0c

Description

Voltage Regulator Settings

Table 481. VREG Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	UPDATE_IN_PRO GRESS	regulator state is being updated writes to the vreg register will be ignored when this field is set	RO	0x0
14:9	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
8:4	VSEL	output voltage select the regulator output voltage is limited to 1.3V unless the voltage limit is disabled using the disable_voltage_limit field in the vreg_ctrl register 00000 - 0.55V 00001 - 0.60V 00010 - 0.65V 00011 - 0.70V 00100 - 0.75V 00101 - 0.80V 00110 - 0.85V 00111 - 0.90V 01000 - 0.95V 01001 - 1.00V 01010 - 1.05V 01011 - 1.10V (default) 01100 - 1.15V 01101 - 1.20V 01110 - 1.25V 01111 - 1.30V 10000 - 1.35V 10001 - 1.40V 10010 - 1.50V 10011 - 1.60V 10100 - 1.65V 10101 - 1.70V 10110 - 1.80V 10111 - 1.90V 11000 - 2.00V 11001 - 2.35V 11010 - 2.50V 11011 - 2.65V 11100 - 2.80V 11101 - 3.00V 11110 - 3.15V 11111 - 3.30V	RW	0x0b
3	Reserved.	-	-	-
2	RESERVED	write 0 to this field	RW	0x0
1	HIZ	high impedance mode select 0=not in high impedance mode, 1=in high impedance mode	RW	0x0
0	Reserved.	-	-	-

POWMAN: VREG_LP_ENTRY Register

Offset: 0x10

Description

Voltage Regulator Low Power Entry Settings

Table 482.
VREG_LP_ENTRY
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8:4	VSEL	output voltage select the regulator output voltage is limited to 1.3V unless the voltage limit is disabled using the disable_voltage_limit field in the vreg_ctrl register 00000 - 0.55V 00001 - 0.60V 00010 - 0.65V 00011 - 0.70V 00100 - 0.75V 00101 - 0.80V 00110 - 0.85V 00111 - 0.90V 01000 - 0.95V 01001 - 1.00V 01010 - 1.05V 01011 - 1.10V (default) 01100 - 1.15V 01101 - 1.20V 01110 - 1.25V 01111 - 1.30V 10000 - 1.35V 10001 - 1.40V 10010 - 1.50V 10011 - 1.60V 10100 - 1.65V 10101 - 1.70V 10110 - 1.80V 10111 - 1.90V 11000 - 2.00V 11001 - 2.35V 11010 - 2.50V 11011 - 2.65V 11100 - 2.80V 11101 - 3.00V 11110 - 3.15V 11111 - 3.30V	RW	0x0b
3	Reserved.	-	-	-
2	MODE	selects either normal (switching) mode or low power (linear) mode low power mode can only be selected for output voltages up to 1.3V 0 = normal mode (switching) 1 = low power mode (linear)	RW	0x1
1	HIZ	high impedance mode select 0=not in high impedance mode, 1=in high impedance mode	RW	0x0
0	Reserved.	-	-	-

POWMAN: VREG_LP_EXIT Register

Offset: 0x14

Description

Voltage Regulator Low Power Exit Settings

Table 483.
VREG_LP_EXIT
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8:4	VSEL	output voltage select the regulator output voltage is limited to 1.3V unless the voltage limit is disabled using the disable_voltage_limit field in the vreg_ctrl register 00000 - 0.55V 00001 - 0.60V 00010 - 0.65V 00011 - 0.70V 00100 - 0.75V 00101 - 0.80V 00110 - 0.85V 00111 - 0.90V 01000 - 0.95V 01001 - 1.00V 01010 - 1.05V 01011 - 1.10V (default) 01100 - 1.15V 01101 - 1.20V 01110 - 1.25V 01111 - 1.30V 10000 - 1.35V 10001 - 1.40V 10010 - 1.50V 10011 - 1.60V 10100 - 1.65V 10101 - 1.70V 10110 - 1.80V 10111 - 1.90V 11000 - 2.00V 11001 - 2.35V 11010 - 2.50V 11011 - 2.65V 11100 - 2.80V 11101 - 3.00V 11110 - 3.15V 11111 - 3.30V	RW	0x0b
3	Reserved.	-	-	-
2	MODE	selects either normal (switching) mode or low power (linear) mode low power mode can only be selected for output voltages up to 1.3V 0 = normal mode (switching) 1 = low power mode (linear)	RW	0x0

Bits	Name	Description	Type	Reset
1	HIZ	high impedance mode select 0=not in high impedance mode, 1=in high impedance mode	RW	0x0
0	Reserved.	-	-	-

POWMAN: BOD_CTRL Register

Offset: 0x18

Description

Brown-out Detection Control

Table 484. BOD_CTRL Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	ISOLATE	isolates the brown-out detection control interface 0 - not isolated (default) 1 - isolated	RW	0x0
11:0	Reserved.	-	-	-

POWMAN: BOD Register

Offset: 0x1c

Description

Brown-out Detection Settings

Table 485. BOD Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8:4	VSEL	threshold select 00000 - 0.473V 00001 - 0.516V 00010 - 0.559V 00011 - 0.602V 00100 - 0.645VS 00101 - 0.688V 00110 - 0.731V 00111 - 0.774V 01000 - 0.817V 01001 - 0.860V (default) 01010 - 0.903V 01011 - 0.946V 01100 - 0.989V 01101 - 1.032V 01110 - 1.075V 01111 - 1.118V 10000 - 1.161 10001 - 1.204V	RW	0x0b
3:1	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
0	EN	enable brown-out detection 0=not enabled, 1=enabled	RW	0x1

POWMAN: BOD_LP_ENTRY Register

Offset: 0x20

Description

Brown-out Detection Low Power Entry Settings

Table 486.
BOD_LP_ENTRY
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8:4	VSEL	threshold select 00000 - 0.473V 00001 - 0.516V 00010 - 0.559V 00011 - 0.602V 00100 - 0.645VS 00101 - 0.688V 00110 - 0.731V 00111 - 0.774V 01000 - 0.817V 01001 - 0.860V (default) 01010 - 0.903V 01011 - 0.946V 01100 - 0.989V 01101 - 1.032V 01110 - 1.075V 01111 - 1.118V 10000 - 1.161 10001 - 1.204V	RW	0x0b
3:1	Reserved.	-	-	-
0	EN	enable brown-out detection 0=not enabled, 1=enabled	RW	0x0

POWMAN: BOD_LP_EXIT Register

Offset: 0x24

Description

Brown-out Detection Low Power Exit Settings

Table 487.
BOD_LP_EXIT Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
8:4	VSEL	threshold select 00000 - 0.473V 00001 - 0.516V 00010 - 0.559V 00011 - 0.602V 00100 - 0.645VS 00101 - 0.688V 00110 - 0.731V 00111 - 0.774V 01000 - 0.817V 01001 - 0.860V (default) 01010 - 0.903V 01011 - 0.946V 01100 - 0.989V 01101 - 1.032V 01110 - 1.075V 01111 - 1.118V 10000 - 1.161 10001 - 1.204V	RW	0x0b
3:1	Reserved.	-	-	-
0	EN	enable brown-out detection 0=not enabled, 1=enabled	RW	0x1

POWMAN: LPOS C Register

Offset: 0x28

Description

Low power oscillator control register.

Table 488. LPOS C Register

Bits	Name	Description	Type	Reset
31:10	Reserved.	-	-	-
9:4	TRIM	Frequency trim - the trim step is typically 1% of the reset frequency, but can be up to 3%	RW	0x20
3:2	Reserved.	-	-	-
1:0	MODE	This feature has been removed	RW	0x3

POWMAN: CHIP_RESET Register

Offset: 0x2c

Description

Chip reset control and status

Table 489. CHIP_RESET Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
28	HAD_WATCHDOG_RESET_PSM	Last reset was a watchdog timeout which was configured to reset the power-on state machine This resets: double_tap flag no DP no RPAP no rescue_flag no timer no powman no swcore no psm yes and does not change the power state	RO	0x0
27	HAD_HZD_SYS_RESET_REQ	Last reset was a system reset from the hazard debugger This resets: double_tap flag no DP no RPAP no rescue_flag no timer no powman no swcore no psm yes and does not change the power state	RO	0x0
26	HAD_GLITCH_DETECT	Last reset was due to a power supply glitch This resets: double_tap flag no DP no RPAP no rescue_flag no timer no powman no swcore no psm yes and does not change the power state	RO	0x0
25	HAD_SWCORE_PD	Last reset was a switched core powerdown This resets: double_tap flag no DP no RPAP no rescue_flag no timer no powman no swcore yes psm yes then starts the power sequencer	RO	0x0

Bits	Name	Description	Type	Reset
24	HAD_WATCHDOG_RESET_SWCORE	Last reset was a watchdog timeout which was configured to reset the switched-core This resets: double_tap flag no DP no RPAP no rescue_flag no timer no powman no swcore yes psm yes then starts the power sequencer	RO	0x0
23	HAD_WATCHDOG_RESET_POWMAN	Last reset was a watchdog timeout which was configured to reset the power manager This resets: double_tap flag no DP no RPAP no rescue_flag no timer yes powman yes swcore yes psm yes then starts the power sequencer	RO	0x0
22	HAD_WATCHDOG_RESET_POWMAN_ASYNC	Last reset was a watchdog timeout which was configured to reset the power manager asynchronously This resets: double_tap flag no DP no RPAP no rescue_flag no timer yes powman yes swcore yes psm yes then starts the power sequencer	RO	0x0
21	HAD_RESCUE	Last reset was a rescue reset from the debugger This resets: double_tap flag no DP no RPAP no rescue_flag no, it sets this flag timer yes powman yes swcore yes psm yes then starts the power sequencer	RO	0x0
20	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
19	HAD_DP_RESET_EQ	Last reset was an reset request from the arm debugger This resets: double_tap flag no DP no RPAP no rescue_flag yes timer yes powman yes swcore yes psm yes then starts the power sequencer	RO	0x0
18	HAD_RUN_LOW	Last reset was from the RUN pin This resets: double_tap flag no DP yes RPAP yes rescue_flag yes timer yes powman yes swcore yes psm yes then starts the power sequencer	RO	0x0
17	HAD_BOR	Last reset was from the brown-out detection block This resets: double_tap flag yes DP yes RPAP yes rescue_flag yes timer yes powman yes swcore yes psm yes then starts the power sequencer	RO	0x0
16	HAD_POR	Last reset was from the power-on reset This resets: double_tap flag yes DP yes RPAP yes rescue_flag yes timer yes powman yes swcore yes psm yes then starts the power sequencer	RO	0x0
15:5	Reserved.	-	-	-
4	RESCUE_FLAG	This is set by a rescue reset from the RP-AP. Its purpose is to halt before the bootrom before booting from flash in order to recover from a boot lock-up. The debugger can then attach once the bootrom has been halted and flash some working code that does not lock up.	WC	0x0
3:1	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
0	DOUBLE_TAP	This flag is set by double-tapping RUN. It tells bootcode to go into the bootloader.	RW	0x0

POWMAN: WATCHDOG Register

Offset: 0x30

Description

Allows a watchdog reset to reset the internal state of powman in addition to the power-on state machine (PSM).

Note that powman ignores watchdog resets that do not select at least the CLOCKS stage or earlier stages in the PSM. If using these bits, it's recommended to set PSM_WDSEL to all-ones in addition to the desired bits in this register. Failing to select CLOCKS or earlier will result in the POWMAN_WATCHDOG register having no effect.

Table 490.
WATCHDOG Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	RESET_PSM	If set to 1, a watchdog reset will run the full power-on state machine (PSM) sequence From a user perspective it is the same as setting RSM_WDSEL_PROC_COLD From a hardware debug perspective it has the same effect as a reset from a glitch detector	RW	0x0
11:9	Reserved.	-	-	-
8	RESET_SWCORE	If set to 1, a watchdog reset will reset the switched core power domain and run the full power-on state machine (PSM) sequence From a user perspective it is the same as setting RSM_WDSEL_PROC_COLD From a hardware debug perspective it has the same effect as a power-on reset for the switched core power domain	RW	0x0
7:5	Reserved.	-	-	-
4	RESET_POWMAN	If set to 1, a watchdog reset will restore powman defaults, reset the timer, reset the switched core power domain and run the full power-on state machine (PSM) sequence This relies on clk_ref running. Use reset_powman_async if that may not be true	RW	0x0
3:1	Reserved.	-	-	-
0	RESET_POWMAN_ASYNC	If set to 1, a watchdog reset will restore powman defaults, reset the timer, reset the switched core domain and run the full power-on state machine (PSM) sequence This does not rely on clk_ref running	RW	0x0

POWMAN: SEQ_CFG Register

Offset: 0x34

Description

For configuration of the power sequencer

Writes are ignored while POWMAN_STATE_CHANGING=1

Table 491. SEQ_CFG Register

Bits	Name	Description	Type	Reset
31:21	Reserved.	-	-	-
20	USING_FAST_POWCK	0 indicates the POWMAN clock is running from the low power oscillator (32kHz) 1 indicates the POWMAN clock is running from the reference clock (2-50MHz)	RO	0x1
19:18	Reserved.	-	-	-
17	USING_BOD_LP	Indicates the brown-out detector (BOD) mode 0 = BOD high power mode which is the default 1 = BOD low power mode	RO	0x0
16	USING_VREG_LP	Indicates the voltage regulator (VREG) mode 0 = VREG high power mode which is the default 1 = VREG low power mode	RO	0x0
15:13	Reserved.	-	-	-
12	USE_FAST_POWCK	selects the reference clock (clk_ref) as the source of the POWMAN clock when switched-core is powered. The POWMAN clock always switches to the slow clock (lposc) when switched-core is powered down because the fast clock stops running. 0 always run the POWMAN clock from the slow clock (lposc) 1 run the POWMAN clock from the fast clock when available This setting takes effect when a power up sequence is next run	RW	0x1
11:9	Reserved.	-	-	-
8	RUN_LPOSOC_IN_LP	Set to 0 to stop the low power osc when the switched-core is powered down, which is unwise if using it to clock the timer This setting takes effect when the swcore is next powered down	RW	0x1
7	USE_BOD_HP	Set to 0 to prevent automatic switching to bod high power mode when switched-core is powered up This setting takes effect when the swcore is next powered up	RW	0x1
6	USE_BOD_LP	Set to 0 to prevent automatic switching to bod low power mode when switched-core is powered down This setting takes effect when the swcore is next powered down	RW	0x1
5	USE_VREG_HP	Set to 0 to prevent automatic switching to vreg high power mode when switched-core is powered up This setting takes effect when the swcore is next powered up	RW	0x1
4	USE_VREG_LP	Set to 0 to prevent automatic switching to vreg low power mode when switched-core is powered down This setting takes effect when the swcore is next powered down	RW	0x1
3:2	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
1	HW_PWRUP_SRAM0	Specifies the power state of SRAM0 when powering up swcore from a low power state (P1.xxx) to a high power state (P0.0xx). 0=power-up 1=no change	RW	0x0
0	HW_PWRUP_SRAM1	Specifies the power state of SRAM1 when powering up swcore from a low power state (P1.xxx) to a high power state (P0.0xx). 0=power-up 1=no change	RW	0x0

POWMAN: STATE Register

Offset: 0x38

Description

This register controls the power state of the 4 power domains.

The current power state is indicated in POWMAN_STATE_CURRENT which is read-only.

To change the state, write to POWMAN_STATE_REQ.

The coding of POWMAN_STATE_CURRENT & POWMAN_STATE_REQ corresponds to the power states defined in the datasheet:

bit 3 = SWCORE

bit 2 = XIP cache

bit 1 = SRAM0

bit 0 = SRAM1

0 = powered up

1 = powered down

When POWMAN_STATE_REQ is written, the POWMAN_STATE_WAITING flag is set while the Power Manager determines what is required. If an invalid transition is requested the Power Manager will still register the request in POWMAN_STATE_REQ but will also set the POWMAN_BAD_REQ flag. It will then implement the power-up requests and ignore the power down requests. To do nothing would risk entering an unrecoverable lock-up state. Invalid requests are: any combination of power up and power down requests any request that results in swcore being powered and xip unpowered If the request is to power down the switched-core domain then POWMAN_STATE_WAITING stays active until the processors halt. During this time the POWMAN_STATE_REQ field can be re-written to change or cancel the request. When the power state transition begins the POWMAN_STATE_WAITING_flag is cleared, the POWMAN_STATE_CHANGING flag is set and POWMAN register writes are ignored until the transition completes.

Table 492. STATE Register

Bits	Name	Description	Type	Reset
31:14	Reserved.	-	-	-
13	CHANGING		RO	0x0
12	WAITING		RO	0x0
11	BAD_HW_REQ	Bad hardware initiated state request. Went back to state 0 (i.e. everything powered up)	RO	0x0
10	BAD_SW_REQ	Bad software initiated state request. No action taken.	RO	0x0
9	PWRUP_WHILE_WAITING	Request ignored because of a pending pwrup request. See current_pwrup_req. Note this blocks powering up AND powering down.	WC	0x0
8	REQ_IGNORED		WC	0x0
7:4	REQ		RW	0x0

Bits	Name	Description	Type	Reset
3:0	CURRENT		RO	0xf

POWMAN: POW_FASTDIV Register

Offset: 0x3c

Table 493.
POW_FASTDIV
Register

Bits	Description	Type	Reset
31:11	Reserved.	-	-
10:0	divides the POWMAN clock to provide a tick for the delay module and state machines when clk_pow is running from the slow clock it is not divided when clk_pow is running from the fast clock it is divided by tick_div	RW	0x040

POWMAN: POW_DELAY Register

Offset: 0x40

Description

power state machine delays

Table 494.
POW_DELAY Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:8	SRAM_STEP	timing between the sram0 and sram1 power state machine steps measured in units of the powman tick period ($\geq 1\mu s$), 0 gives a delay of 1 unit	RW	0x20
7:4	XIP_STEP	timing between the xip power state machine steps measured in units of the lposc period, 0 gives a delay of 1 unit	RW	0x1
3:0	SWCORE_STEP	timing between the swcore power state machine steps measured in units of the lposc period, 0 gives a delay of 1 unit	RW	0x1

POWMAN: EXT_CTRL0 Register

Offset: 0x44

Description

Configures a gpio as a power mode aware control output

Table 495. EXT_CTRL0 Register

Bits	Name	Description	Type	Reset
31:15	Reserved.	-	-	-
14	LP_EXIT_STATE	output level when exiting the low power state	RW	0x0
13	LP_ENTRY_STATE	output level when entering the low power state	RW	0x0
12	INIT_STATE		RW	0x0
11:9	Reserved.	-	-	-
8	INIT		RW	0x0
7:6	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
5:0	GPIO_SELECT	selects from gpio 0→30 set to 31 to disable this feature	RW	0x3f

POWMAN: EXT_CTRL1 Register

Offset: 0x48

Description

Configures a gpio as a power mode aware control output

Table 496. EXT_CTRL1 Register

Bits	Name	Description	Type	Reset
31:15	Reserved.	-	-	-
14	LP_EXIT_STATE	output level when exiting the low power state	RW	0x0
13	LP_ENTRY_STATE	output level when entering the low power state	RW	0x0
12	INIT_STATE		RW	0x0
11:9	Reserved.	-	-	-
8	INIT		RW	0x0
7:6	Reserved.	-	-	-
5:0	GPIO_SELECT	selects from gpio 0→30 set to 31 to disable this feature	RW	0x3f

POWMAN: EXT_TIME_REF Register

Offset: 0x4c

Description

Select a GPIO to use as a time reference, the source can be used to drive the low power clock at 32kHz, or to provide a 1ms tick to the timer, or provide a 1Hz tick to the timer. The tick selection is controlled by the POWMAN_TIMER register.

Table 497.
EXT_TIME_REF Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	DRIVE_LPCK	Use the selected GPIO to drive the 32kHz low power clock, in place of LPOSC. This field must only be written when POWMAN_TIMER_RUN=0	RW	0x0
3:2	Reserved.	-	-	-
1:0	SOURCE_SEL	0 → gpio12 1 → gpio20 2 → gpio14 3 → gpio22	RW	0x0

POWMAN: LPOSC_FREQ_KHZ_INT Register

Offset: 0x50

Description

Informs the AON-timer of the integer component of the clock frequency when running off the LPOSC.

Table 498.
LPOS_C_FREQ_KHZ_IN
T Register

Bits	Description	Type	Reset
31:6	Reserved.	-	-
5:0	Integer component of the LPOS_C or GPIO clock source frequency in kHz. Default = 32 This field must only be written when POWMAN_TIMER_RUN=0 or POWMAN_TIMER_USING_XOSC=1	RW	0x20

POWMAN: LPOS_C_FREQ_KHZ_FRAC Register

Offset: 0x54

Description

Informs the AON-timer of the fractional component of the clock frequency when running off the LPOS_C.

Table 499.
LPOS_C_FREQ_KHZ_FR
AC Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Fractional component of the LPOS_C or GPIO clock source frequency in kHz. Default = 0.768 This field must only be written when POWMAN_TIMER_RUN=0 or POWMAN_TIMER_USING_XOSC=1	RW	0xc49c

POWMAN: XOSC_FREQ_KHZ_INT Register

Offset: 0x58

Description

Informs the AON-timer of the integer component of the clock frequency when running off the XOSC.

Table 500.
XOSC_FREQ_KHZ_INT
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Integer component of the XOSC frequency in kHz. Default = 12000 Must be >1 This field must only be written when POWMAN_TIMER_RUN=0 or POWMAN_TIMER_USING_XOSC=0	RW	0x2ee0

POWMAN: XOSC_FREQ_KHZ_FRAC Register

Offset: 0x5c

Description

Informs the AON-timer of the fractional component of the clock frequency when running off the XOSC.

Table 501.
XOSC_FREQ_KHZ_FRA
C Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Fractional component of the XOSC frequency in kHz. This field must only be written when POWMAN_TIMER_RUN=0 or POWMAN_TIMER_USING_XOSC=0	RW	0x0000

POWMAN: SET_TIME_63TO48 Register

Offset: 0x60

Table 502.
SET_TIME_63T048
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	For setting the time, do not use for reading the time, use POWMAN_READ_TIME_UPPER and POWMAN_READ_TIME_LOWER. This field must only be written when POWMAN_TIMER_RUN=0	RW	0x0000

POWMAN: SET_TIME_47T032 Register

Offset: 0x64

Table 503.
SET_TIME_47T032
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	For setting the time, do not use for reading the time, use POWMAN_READ_TIME_UPPER and POWMAN_READ_TIME_LOWER. This field must only be written when POWMAN_TIMER_RUN=0	RW	0x0000

POWMAN: SET_TIME_31T016 Register

Offset: 0x68

Table 504.
SET_TIME_31T016
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	For setting the time, do not use for reading the time, use POWMAN_READ_TIME_UPPER and POWMAN_READ_TIME_LOWER. This field must only be written when POWMAN_TIMER_RUN=0	RW	0x0000

POWMAN: SET_TIME_15T00 Register

Offset: 0x6c

Table 505.
SET_TIME_15T00
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	For setting the time, do not use for reading the time, use POWMAN_READ_TIME_UPPER and POWMAN_READ_TIME_LOWER. This field must only be written when POWMAN_TIMER_RUN=0	RW	0x0000

POWMAN: READ_TIME_UPPER Register

Offset: 0x70

Table 506.
READ_TIME_UPPER
Register

Bits	Description	Type	Reset
31:0	For reading bits 63:32 of the timer. When reading all 64 bits it is possible for the LOWER count to rollover during the read. It is recommended to read UPPER, then LOWER, then re-read UPPER and, if it has changed, re-read LOWER.	RO	0x00000000

POWMAN: READ_TIME_LOWER Register

Offset: 0x74

Table 507.
READ_TIME_LOWER
Register

Bits	Description	Type	Reset
31:0	For reading bits 31:0 of the timer.	RO	0x00000000

POWMAN: ALARM_TIME_63T048 Register

Offset: 0x78

Table 508.
ALARM_TIME_63T048
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	This field must only be written when POWMAN_ALARM_ENAB=0	RW	0x0000

POWMAN: ALARM_TIME_47T032 Register

Offset: 0x7c

Table 509.
ALARM_TIME_47T032
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	This field must only be written when POWMAN_ALARM_ENAB=0	RW	0x0000

POWMAN: ALARM_TIME_31T016 Register

Offset: 0x80

Table 510.
ALARM_TIME_31T016
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	This field must only be written when POWMAN_ALARM_ENAB=0	RW	0x0000

POWMAN: ALARM_TIME_15T00 Register

Offset: 0x84

Table 511.
ALARM_TIME_15T00
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	This field must only be written when POWMAN_ALARM_ENAB=0	RW	0x0000

POWMAN: TIMER Register

Offset: 0x88

Table 512. TIMER
Register

Bits	Name	Description	Type	Reset
31:20	Reserved.	-	-	-
19	USING_GPIO_1HZ	Timer is synchronised to a 1hz gpio source	RO	0x0
18	USING_GPIO_1KH Z	Timer is running from a 1khz gpio source	RO	0x0
17	USING_LPOSC	Timer is running from lposc	RO	0x0
16	USING_XOSC	Timer is running from xosc	RO	0x0
15:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	USE_GPIO_1HZ	Selects the gpio source as the reference for the sec counter. The msec counter will continue to use the lposc or xosc reference.	RW	0x0
12:11	Reserved.	-	-	-
10	USE_GPIO_1KHZ	switch to gpio as the source of the 1kHz timer tick	SC	0x0
9	USE_XOSC	switch to xosc as the source of the 1kHz timer tick	SC	0x0
8	USE_LPOSC	Switch to lposc as the source of the 1kHz timer tick	SC	0x0
7	Reserved.	-	-	-
6	ALARM	Alarm has fired. Write to 1 to clear the alarm.	WC	0x0
5	PWRUP_ON_ALARM	Alarm wakes the chip from low power mode	RW	0x0
4	ALARM_ENAB	Enables the alarm. The alarm must be disabled while writing the alarm time.	RW	0x0
3	Reserved.	-	-	-
2	CLEAR	Clears the timer, does not disable the timer and does not affect the alarm. This control can be written at any time.	SC	0x0
1	RUN	Enables the timer, the timer is not cleared when RUN is deasserted. To run the timer set the POWMAN_LPOSC_FREQ* and/or POWMAN_XOSC_FREQ* registers, then set the time in the POWMAN_TIMER_SEC* & POWMAN_TIMER_MSEC registers, then start the timer by setting POWMAN_TIMER_RUN=1. This will start the timer running from the LPOSC. When the XOSC is available switch the reference clock to XOSC then select it as the timer clock by setting POWMAN_TIMER_USE_XOSC=1	RW	0x0
0	NONSEC_WRITE	Control whether Non-secure software can write to the timer registers. All other registers are hardwired to be inaccessible to Non-secure.	RW	0x0

POWMAN: PWRUP0 Register

Offset: 0x8c

Description

4 GPIO powerup events can be configured to wake the chip up from a low power state. The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event. The number of gpios available depends on the package option. An invalid selection will be ignored

source = 0 selects gpio0

1. +
2. + source = 47 selects gpio47
source = 48 selects qspi_ss
source = 49 selects qspi_sd0
source = 50 selects qspi_sd1
source = 51 selects qspi_sd2
source = 52 selects qspi_sd3
source = 53 selects qspi_sclk

level = 0 triggers the pwrup when the source is low
 level = 1 triggers the pwrup when the source is high

Table 513. PWRUP0 Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	RAW_STATUS	Value of selected gpio pin (only if enable == 1)	RO	0x0
9	STATUS	Status of gpio wakeup. Write to 1 to clear a latched edge detect.	WC	0x0
8	MODE	Edge or level detect. Edge will detect a 0 to 1 transition (or 1 to 0 transition). Level will detect a 1 or 0. Both types of event get latched into the current_pwrup_req register. 0x0 → level 0x1 → edge	RW	0x0
7	DIRECTION	0x0 → low_falling 0x1 → high_rising	RW	0x0
6	ENABLE	Set to 1 to enable the wakeup source. Set to 0 to disable the wakeup source and clear a pending wakeup event. If using edge detect a latched edge needs to be cleared by writing 1 to the status register also.	RW	0x0
5:0	SOURCE		RW	0x3f

POWMAN: PWRUP1 Register

Offset: 0x90

Description

4 GPIO powerup events can be configured to wake the chip up from a low power state.
 The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event
 The number of gpios available depends on the package option. An invalid selection will be ignored
 source = 0 selects gpio0

1. +
2. + source = 47 selects gpio47
 source = 48 selects qspi_ss
 source = 49 selects qspi_sd0
 source = 50 selects qspi_sd1
 source = 51 selects qspi_sd2
 source = 52 selects qspi_sd3
 source = 53 selects qspi_sclk
 level = 0 triggers the pwrup when the source is low
 level = 1 triggers the pwrup when the source is high

Table 514. PWRUP1 Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	RAW_STATUS	Value of selected gpio pin (only if enable == 1)	RO	0x0
9	STATUS	Status of gpio wakeup. Write to 1 to clear a latched edge detect.	WC	0x0

Bits	Name	Description	Type	Reset
8	MODE	Edge or level detect. Edge will detect a 0 to 1 transition (or 1 to 0 transition). Level will detect a 1 or 0. Both types of event get latched into the current_pwrup_req register. 0x0 → level 0x1 → edge	RW	0x0
7	DIRECTION	0x0 → low_falling 0x1 → high_rising	RW	0x0
6	ENABLE	Set to 1 to enable the wakeup source. Set to 0 to disable the wakeup source and clear a pending wakeup event. If using edge detect a latched edge needs to be cleared by writing 1 to the status register also.	RW	0x0
5:0	SOURCE		RW	0x3f

POWMAN: PWRUP2 Register

Offset: 0x94

Description

4 GPIO powerup events can be configured to wake the chip up from a low power state.

The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event

The number of gpios available depends on the package option. An invalid selection will be ignored
source = 0 selects gpio0

1. +
2. + source = 47 selects gpio47
source = 48 selects qspi_ss
source = 49 selects qspi_sd0
source = 50 selects qspi_sd1
source = 51 selects qspi_sd2
source = 52 selects qspi_sd3
source = 53 selects qspi_sclk
level = 0 triggers the pwrup when the source is low
level = 1 triggers the pwrup when the source is high

Table 515. PWRUP2 Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	RAW_STATUS	Value of selected gpio pin (only if enable == 1)	RO	0x0
9	STATUS	Status of gpio wakeup. Write to 1 to clear a latched edge detect.	WC	0x0
8	MODE	Edge or level detect. Edge will detect a 0 to 1 transition (or 1 to 0 transition). Level will detect a 1 or 0. Both types of event get latched into the current_pwrup_req register. 0x0 → level 0x1 → edge	RW	0x0
7	DIRECTION	0x0 → low_falling 0x1 → high_rising	RW	0x0
6	ENABLE	Set to 1 to enable the wakeup source. Set to 0 to disable the wakeup source and clear a pending wakeup event. If using edge detect a latched edge needs to be cleared by writing 1 to the status register also.	RW	0x0

Bits	Name	Description	Type	Reset
5:0	SOURCE		RW	0x3f

POWMAN: PWRUP3 Register

Offset: 0x98

Description

4 GPIO powerup events can be configured to wake the chip up from a low power state.

The pwrups are level/edge sensitive and can be set to trigger on a high/rising or low/falling event

The number of gpios available depends on the package option. An invalid selection will be ignored
source = 0 selects gpio0

1. +
2. + source = 47 selects gpio47
source = 48 selects qspi_ss
source = 49 selects qspi_sd0
source = 50 selects qspi_sd1
source = 51 selects qspi_sd2
source = 52 selects qspi_sd3
source = 53 selects qspi_sclk
level = 0 triggers the pwrup when the source is low
level = 1 triggers the pwrup when the source is high

Table 516. PWRUP3 Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	RAW_STATUS	Value of selected gpio pin (only if enable == 1)	RO	0x0
9	STATUS	Status of gpio wakeup. Write to 1 to clear a latched edge detect.	WC	0x0
8	MODE	Edge or level detect. Edge will detect a 0 to 1 transition (or 1 to 0 transition). Level will detect a 1 or 0. Both types of event get latched into the current_pwrup_req register. 0x0 → level 0x1 → edge	RW	0x0
7	DIRECTION	0x0 → low_falling 0x1 → high_rising	RW	0x0
6	ENABLE	Set to 1 to enable the wakeup source. Set to 0 to disable the wakeup source and clear a pending wakeup event. If using edge detect a latched edge needs to be cleared by writing 1 to the status register also.	RW	0x0
5:0	SOURCE		RW	0x3f

POWMAN: CURRENT_PWRUP_REQ Register

Offset: 0x9c

Table 517. CURRENT_PWRUP_REQ Register

Bits	Description	Type	Reset
31:7	Reserved.	-	-

Bits	Description	Type	Reset
6:0	Indicates current powerup request state pwrup events can be cleared by removing the enable from the pwrup register. The alarm pwrup req can be cleared by clearing timer.alarm_enab 0 = chip reset, for the source of the last reset see POWMAN_CHIP_RESET 1 = pwrup0 2 = pwrup1 3 = pwrup2 4 = pwrup3 5 = coresight_pwrup 6 = alarm_pwrup	RO	0x00

POWMAN: LAST_SWCORE_PWRUP Register

Offset: 0xa0

Table 518.
LAST_SWCORE_PWRUP
P Register

Bits	Description	Type	Reset
31:7	Reserved.	-	-
6:0	Indicates which pwrup source triggered the last switched-core power up 0 = chip reset, for the source of the last reset see POWMAN_CHIP_RESET 1 = pwrup0 2 = pwrup1 3 = pwrup2 4 = pwrup3 5 = coresight_pwrup 6 = alarm_pwrup	RO	0x00

POWMAN: DBG_PWRCFG Register

Offset: 0xa4

Table 519.
DBG_PWRCFG
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	IGNORE	Ignore pwrup req from debugger. If pwrup req is asserted then this will prevent power down and set powerdown blocked. Set ignore to stop paying attention to pwrup_req	RW	0x0

POWMAN: BOOTDIS Register

Offset: 0xa8

Description

Tell the bootrom to ignore the BOOT0..3 registers following the next RSM reset (e.g. the next core power down/up).

If an early boot stage has soft-locked some OTP pages in order to protect their contents from later stages, there is a risk that Secure code running at a later stage can unlock the pages by powering the core up and down.

This register can be used to ensure that the bootloader runs as normal on the next power up, preventing Secure code at a later stage from accessing OTP in its unlocked state.

Should be used in conjunction with the OTP BOOTDIS register.

Table 520. BOOTDIS
Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
1	NEXT	<p>This flag always ORs writes into its current contents. It can be set but not cleared by software.</p> <p>The BOOTDIS_NEXT bit is OR'd into the BOOTDIS_NOW bit when the core is powered down. Simultaneously, the BOOTDIS_NEXT bit is cleared. Setting this bit means that the BOOT0..3 registers will be ignored following the next reset of the RSM by powman.</p> <p>This flag should be set by an early boot stage that has soft-locked OTP pages, to prevent later stages from unlocking it by power cycling.</p>	RW	0x0
0	NOW	<p>When powman resets the RSM, the current value of BOOTDIS_NEXT is OR'd into BOOTDIS_NOW, and BOOTDIS_NEXT is cleared.</p> <p>The bootrom checks this flag before reading the BOOT0..3 registers. If it is set, the bootrom clears it, and ignores the BOOT registers. This prevents Secure software from diverting the boot path before a bootloader has had the chance to soft lock OTP pages containing sensitive data.</p>	WC	0x0

POWMAN: DBGCONFIG Register

Offset: 0xac

Table 521.
DBGCONFIG Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	DP_INSTID	Configure DP instance ID for SWD multidrop selection. Recommend that this is NOT changed until you require debug access in multi-chip environment	RW	0x0

POWMAN: SCRATCH0, SCRATCH1, ..., SCRATCH6, SCRATCH7 Registers

Offsets: 0xb0, 0xb4, ..., 0xc8, 0xcc

Table 522. SCRATCH0,
SCRATCH1, ...,
SCRATCH6,
SCRATCH7 Registers

Bits	Description	Type	Reset
31:0	Scratch register. Information persists in low power mode	RW	0x00000000

POWMAN: BOOT0, BOOT1, BOOT2, BOOT3 Registers

Offsets: 0xd0, 0xd4, 0xd8, 0xdc

Table 523. BOOT0,
BOOT1, BOOT2,
BOOT3 Registers

Bits	Description	Type	Reset
31:0	Scratch register. Information persists in low power mode	RW	0x00000000

POWMAN: INTR Register

Offset: 0xe0

Description

Raw Interrupts

Table 524. INTR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	PWRUP WHILE_WAITING	Source is state.pwrup_while_waiting	RO	0x0
2	STATE_REQ_IGNORED	Source is state.req_ignored	RO	0x0
1	TIMER		RO	0x0
0	VREG_OUTPUT_LOW		WC	0x0

POWMAN: INTE Register

Offset: 0xe4

Description

Interrupt Enable

Table 525. INTE Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	PWRUP WHILE_WAITING	Source is state.pwrup_while_waiting	RW	0x0
2	STATE_REQ_IGNORED	Source is state.req_ignored	RW	0x0
1	TIMER		RW	0x0
0	VREG_OUTPUT_LOW		RW	0x0

POWMAN: INTF Register

Offset: 0xe8

Description

Interrupt Force

Table 526. INTF Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	PWRUP WHILE_WAITING	Source is state.pwrup_while_waiting	RW	0x0
2	STATE_REQ_IGNORED	Source is state.req_ignored	RW	0x0
1	TIMER		RW	0x0
0	VREG_OUTPUT_LOW		RW	0x0

POWMAN: INTS Register

Offset: 0xec

Description

Interrupt status after masking & forcing

Table 527. INTS Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	PWRUP WHILE_WAITING	Source is state.pwrup_while_waiting	RO	0x0
2	STATE_REQ_IGNORED	Source is state.req_ignored	RO	0x0
1	TIMER		RO	0x0
0	VREG_OUTPUT_LOW		RO	0x0

Chapter 7. Resets

7.1. Overview

Resets are divided into three categories, each of which applies to a subset of RP2350:

Chip-level Resets

apply to the entire chip. Used to put the entire chip into a default state. These are initiated by hardware events, the watchdog, or the debugger. When all chip level resets are de-asserted, the system resets are released and the processors boot.

System Resets

apply to components essential to processor operation. System components have interdependencies, therefore their resets are de-asserted in sequence by the Power-on State Machine (PSM). The full PSM sequence is triggered by deassertion of chip-level resets. A full or partial sequence can be triggered by the watchdog or debugger. The sequence culminates in processor boot.

Subsystem Resets

apply to components *not* essential for operation of the processors. The resets can be independently asserted by writing to the **RESETS** registers and de-asserted by software, the watchdog, or the debugger.

The watchdog can be programmed to trigger any of the above categories.

7.2. Changes from RP2350

RP2350 retains all RP2040 chip-level reset features.

RP2350 adds the following features:

- new chip reset sources:
 - glitch detector
 - watchdog
 - debugger
- new destinations:
 - new power management components

RP2350 makes the following modifications to existing features:

- Modified the **CHIP_RESET** register, which records the source of the last chip level reset. In RP2040, **CHIP_RESET** was stored in the **LDO_POR** register block. In RP2350, **CHIP_RESET** was extended and moved to the **POWMAN** register block, which is in the new always-on power domain (**AON**).
- Renamed the **BOR** registers to **BOD**, added functionality, and moved them to the new **POWMAN** register block.
- Added more system reset stages. To support this, added additional Power-on State Machine fields and rearranged the existing fields.
- Added additional **RESETS** registers and rearranged the existing fields.
- Extended watchdog options to enable triggers for new resets.

NOTE

Watchdog scratch registers are not preserved when the watchdog triggers a chip-level reset. However, watchdog scratch registers are preserved after a system or subsystem reset. For general purpose scratch registers that do not reset after a chip-level reset, see the [POWMAN](#) register block [Section 6.6, “Power Management \(POWMAN\) Registers”](#).

7.3. Chip Level Resets

Chip-level resets put the entire chip into a default state. These resets are only initiated by hardware events, the debugger, or a watchdog timeout. They cannot be initiated or blocked by software.

7.3.1. Chip-Level Reset table

Table 528, “Chip Level Reset Table” shows the components reset by each of the chip-level reset sources. “nc” indicates “no change”.

Table 528. Chip Level Reset Table

Reset Source	Debugger	Power Manager Scratch and Boot Registers	Power Manager	Power State	System Resets	Watchdog Scratch Registers	DOUBLE_TAP	RESCUE_FLAG
POR	reset	reset	hard reset	→ P0.0	PSM runs	reset	reset	reset
BOR	reset	reset	hard reset	→ P0.0	PSM runs	reset	reset	reset
EXTERNAL RESET (RUN)	reset	reset	hard reset	→ P0.0	PSM runs	reset	nc	reset
DEBUGGER RESET REQ	nc	nc	hard reset	→ P0.0	PSM runs	reset	nc	reset
DEBUGGER RESCUE	nc	nc	hard reset	→ P0.0	PSM runs	reset	nc	set
WATCHDOG POWMAN ASYNC RESET	nc	nc	hard reset	→ P0.0	PSM runs	reset	nc	nc
WATCHDOG POWMAN RESET	nc	nc	soft reset	→ P0.0	PSM runs	reset	nc	nc
WATCHDOG SWCORE RESET	nc	nc	nc	→ P0.0	PSM runs	reset	nc	nc
SWCORE POWERDOWN	nc	nc	nc	→ P0.0	PSM runs	reset	nc	nc
GLITCH_DETECTOR	nc	nc	nc	nc	PSM runs	reset	nc	nc
DEBUGGER HZD_RESET REQ	nc	nc	nc	nc	PSM runs	reset	nc	nc
WATCHDOG RESET PSM	nc	nc	nc	nc	PSM runs	reset	nc	nc

7.3.2. Chip-level Reset Destinations

Chip-level resets apply to the following primary components:

- debugger
- power manager scratch and boot registers
- power manager including the always-on timer
- power state (restored to state [P0.0](#), in which all domains are powered, see [Section 6.5, “Power Manager”](#))
- system resets (any chip-level reset triggers the PSM (power-on state machine), which sequences the system resets, see [Section 7.4, “System Resets \(Power-on State Machine\)”](#))
- watchdog scratch registers (reset by any chip-level reset, including one triggered by the watchdog)

Chip-level resets also reset the following two [CHIP_RESET](#) register flags:

- [CHIP_RESET.DOUBLE_TAP](#), which is used to detect a double tap on the [RUN](#) pin and trigger an alternate boot sequence
- [CHIP_RESET.RESCUE_FLAG](#), which is used by the debugger to enter an alternate debug state (see [Section 3.5.8, “Rescue Reset”](#))

INFO: These flags are located in the [CHIP_RESET](#) register so they are included in the always-on (AON) power domain.

7.3.3. Chip-level Reset Sources

In order of severity, the following events can trigger a chip-level reset:

7.3.3.1. Power-On Reset (POR)

The power-on reset ensures the chip starts up cleanly when power is first applied by holding it in reset until the digital core supply (DVDD) reaches a voltage high enough to reliably power the chip's core logic. The POR component is described in detail in [Section 7.6.1, “Power-on Reset \(POR\)”](#).

7.3.3.2. Brown-out Detection (BOD)

The brown-out reset prevents unreliable operation when the digital core supply (DVDD) drops below a safe operating level. The BOR component is described in detail in [Section 7.6.2, “Brownout Detection \(BOD\)”](#).

7.3.3.3. External Reset

The chip can be reset by taking the [RUN](#) pin low. This holds the chip in reset irrespective of the state of the core power supply (DVDD), the power-on reset block, and brownout detection block. [RUN](#) can be used to extend the initial power-on reset, or can be driven from an external source to start and stop the chip as required. If [RUN](#) is not used, it should be tied high. Double-tapping the [RUN](#) low will set [CHIP_RESET.DOUBLE_TAP](#). Boot code reads this flag and selects an alternate boot sequence if the flag is set.

7.3.3.4. Debugger Reset Request

The debugger is able to initiate a chip-level reset using the [CDBGWRUPREQ](#) control. For more information, see [Section 3.5, “Debug”](#).

7.3.3.5. Rescue Debug Port Reset

The chip can also be reset via the Rescue Debug Port. This allows the chip to be recovered from a locked-up state. In addition to resetting the chip, a Rescue Debug Port reset also sets `CHIP_RESET.RESCUE_FLAG`. This is checked by boot code at startup, causing it to enter a safe state if the bit is set. See [Section 3.5.8, “Rescue Reset”](#) for more information.

7.3.3.6. Watchdog

The watchdog can trigger various levels of chip-level reset by setting appropriate bits in the `WATCHDOG` register. A chip-level reset triggered by a watchdog reset will reset the watchdog scratch registers. Additional general purpose scratch registers are available in `POWMAN`. These are not reset by a chip-level reset triggered by the watchdog.

7.3.3.7. SWCORE Powerdown

For a list of operations that power down the switched-core power domain (SWCORE) and trigger this reset, see [Section 6.5](#).

7.3.3.8. Glitch Detector

This reset fires if a glitch is detected in SWCORE power supply. For more information, see [Section 10.7](#).

7.3.3.9. DEBUGGER HZD RESET REQ

This reset is triggered by a system reset request (`SYSRESETREQ`) from the Hazard processor debugger. For more information, see [Section 3.5.3](#).

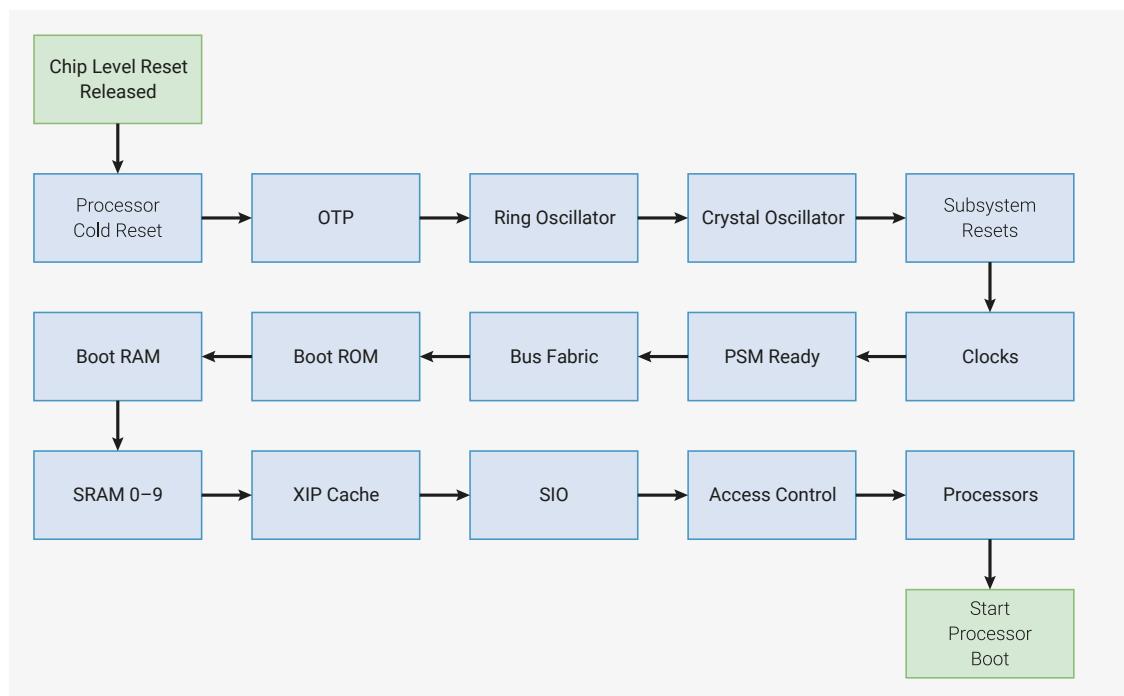
7.3.3.10. Source of Last Reset

The source of the last chip-level reset is recorded in the `CHIP_RESET` register. A complete list of `POWMAN` registers is provided in [Section 6.6](#), but information on registers associated with chip-level resets is repeated here.

- `CHIP_RESET`

7.4. System Resets (Power-on State Machine)

Figure 24. Power-on State Machine Sequence



System Resets apply to components essential to processor operation. System components have interdependencies, therefore their resets are de-asserted in sequence by the Power-on State Machine (PSM). Each stage of the sequencer outputs a reset done signal when complete, `rst_done`, which releases the reset input to the next stage. A partial sequence runs after a write to the `FRCE_OFF` register or a watchdog timeout. Note that the `FRCE_ON` register is intended for internal use only and is disabled in production devices.

7.4.1. Reset Sequence

The Power-on State Machine (PSM) sequence is:

1. Removes cold reset to processors.
2. Takes OTP out of reset. OTP reads any content required to boot and asserts `rst_done`.
3. Starts the Ring Oscillator. Asserts `rst_done` once the oscillator output is stable.
4. Removes Crystal Oscillator (XOSC) controller reset. The XOSC does not start yet, so `rst_done` is asserted immediately.
5. Deasserts the master subsystem reset, but does not remove individual subsystem resets.
6. Starts the `clk_ref` and `clk_sys` clock generators. In the initial configuration, `clk_ref` runs from the ring oscillator with no divider and `clk_sys` runs from `clk_ref`.
7. The PSM confirms the clocks are active.
8. Removes Bus Fabric reset and initialises logic.
9. Removes various memory controllers' resets and initialises logic.
10. Removes Single-cycle IO subsystem (SIO) reset and initialises logic.
11. Removes Access Controller reset and initialises logic.
12. Deasserts Processor Complex reset. Both `core0` and `core1` start executing the boot code from ROM. The boot code reads the core id and `core1` sleeps, leaving `core0` to continue bootrom execution.

7.4.2. Register Control

The PSM is a fully automated piece of hardware: it requires no input from the user to work. The debugger can trigger a full or partial sequence by writing to the `FRCE_OFF` register. The `FRCE_ON` register is a development feature that does nothing in production devices.

7.4.3. Interaction with Watchdog

The watchdog can trigger a full or partial sequence by writing to the `WDSEL` register.

7.4.4. List of Registers

The PSM registers start at a base address of `0x40018000` (defined as `PSM_BASE` in SDK).

Table 529. List of PSM registers

Offset	Name	Info
0x0	<code>FRCE_ON</code>	Force block out of reset (i.e. power it on)
0x4	<code>FRCE_OFF</code>	Force into reset (i.e. power it off)
0x8	<code>WDSEL</code>	Set to 1 if the watchdog should reset this
0xc	<code>DONE</code>	Is the subsystem ready?

PSM: FRCE_ON Register

Offset: 0x0

Description

Force block out of reset (i.e. power it on)

Table 530. `FRCE_ON` Register

Bits	Name	Description	Type	Reset
31:25	Reserved.	-	-	-
24	PROC1		RW	0x0
23	PROC0		RW	0x0
22	ACCESSCTRL		RW	0x0
21	SIO		RW	0x0
20	XIP		RW	0x0
19	SRAM9		RW	0x0
18	SRAM8		RW	0x0
17	SRAM7		RW	0x0
16	SRAM6		RW	0x0
15	SRAM5		RW	0x0
14	SRAM4		RW	0x0
13	SRAM3		RW	0x0
12	SRAM2		RW	0x0
11	SRAM1		RW	0x0
10	SRAM0		RW	0x0

Bits	Name	Description	Type	Reset
9	BOOTRAM		RW	0x0
8	ROM		RW	0x0
7	BUSFABRIC		RW	0x0
6	PSM_READY		RW	0x0
5	CLOCKS		RW	0x0
4	RESETS		RW	0x0
3	XOSC		RW	0x0
2	ROSC		RW	0x0
1	OTP		RW	0x0
0	PROC_COLD		RW	0x0

PSM: FRCE_OFF Register

Offset: 0x4

Description

Force into reset (i.e. power it off)

Table 531. FRCE_OFF Register

Bits	Name	Description	Type	Reset
31:25	Reserved.	-	-	-
24	PROC1		RW	0x0
23	PROC0		RW	0x0
22	ACCESSCTRL		RW	0x0
21	SIO		RW	0x0
20	XIP		RW	0x0
19	SRAM9		RW	0x0
18	SRAM8		RW	0x0
17	SRAM7		RW	0x0
16	SRAM6		RW	0x0
15	SRAM5		RW	0x0
14	SRAM4		RW	0x0
13	SRAM3		RW	0x0
12	SRAM2		RW	0x0
11	SRAM1		RW	0x0
10	SRAM0		RW	0x0
9	BOOTRAM		RW	0x0
8	ROM		RW	0x0
7	BUSFABRIC		RW	0x0
6	PSM_READY		RW	0x0

Bits	Name	Description	Type	Reset
5	CLOCKS		RW	0x0
4	RESETS		RW	0x0
3	XOSC		RW	0x0
2	ROSC		RW	0x0
1	OTP		RW	0x0
0	PROC_COLD		RW	0x0

PSM: WDSEL Register

Offset: 0x8

Description

Set to 1 if the watchdog should reset this

Table 532. WDSEL Register

Bits	Name	Description	Type	Reset
31:25	Reserved.	-	-	-
24	PROC1		RW	0x0
23	PROC0		RW	0x0
22	ACCESSCTRL		RW	0x0
21	SIO		RW	0x0
20	XIP		RW	0x0
19	SRAM9		RW	0x0
18	SRAM8		RW	0x0
17	SRAM7		RW	0x0
16	SRAM6		RW	0x0
15	SRAM5		RW	0x0
14	SRAM4		RW	0x0
13	SRAM3		RW	0x0
12	SRAM2		RW	0x0
11	SRAM1		RW	0x0
10	SRAM0		RW	0x0
9	BOOTRAM		RW	0x0
8	ROM		RW	0x0
7	BUSFABRIC		RW	0x0
6	PSM_READY		RW	0x0
5	CLOCKS		RW	0x0
4	RESETS		RW	0x0
3	XOSC		RW	0x0
2	ROSC		RW	0x0

Bits	Name	Description	Type	Reset
1	OTP		RW	0x0
0	PROC_COLD		RW	0x0

PSM: DONE Register

Offset: 0xc

Description

Is the subsystem ready?

Table 533. DONE Register

Bits	Name	Description	Type	Reset
31:25	Reserved.	-	-	-
24	PROC1		RO	0x0
23	PROC0		RO	0x0
22	ACCESSCTRL		RO	0x0
21	SIO		RO	0x0
20	XIP		RO	0x0
19	SRAM9		RO	0x0
18	SRAM8		RO	0x0
17	SRAM7		RO	0x0
16	SRAM6		RO	0x0
15	SRAM5		RO	0x0
14	SRAM4		RO	0x0
13	SRAM3		RO	0x0
12	SRAM2		RO	0x0
11	SRAM1		RO	0x0
10	SRAM0		RO	0x0
9	BOOTRAM		RO	0x0
8	ROM		RO	0x0
7	BUSFABRIC		RO	0x0
6	PSM_READY		RO	0x0
5	CLOCKS		RO	0x0
4	RESETS		RO	0x0
3	XOSC		RO	0x0
2	ROSC		RO	0x0
1	OTP		RO	0x0
0	PROC_COLD		RO	0x0

7.5. Subsystem Resets

7.5.1. Overview

The reset controller allows software to reset non-critical components in RP2350. The reset controller can reset the following components:

- USB Controller
- PIO
- Peripherals, including UART, I2C, SPI, PWM, Timer, ADC
- PLLs
- IO and Pad registers

For a full list of components that can be reset using the reset controller, see the register descriptions (Section 7.5.3).

When reset, components are held in reset at power-up. To use the component, software must deassert the reset.

i NOTE

The SDK automatically deasserts some components after a reset.

7.5.2. Programmer's Model

The SDK uses the following struct to represent the resets registers:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2350/hardware_structs/include/hardware/structs/resets.h Lines 24 - 122

```

24 typedef struct {
25     _REG_(RESETS_RESET_OFFSET) // RESETS_RESET
26     // 0x10000000 [28] : usbctrl (1)
27     // 0x08000000 [27] : uart1 (1)
28     // 0x04000000 [26] : uart0 (1)
29     // 0x02000000 [25] : trng (1)
30     // 0x01000000 [24] : timer1 (1)
31     // 0x00800000 [23] : timer0 (1)
32     // 0x00400000 [22] : tbman (1)
33     // 0x00200000 [21] : sysinfo (1)
34     // 0x00100000 [20] : syscfg (1)
35     // 0x00080000 [19] : spi1 (1)
36     // 0x00040000 [18] : spi0 (1)
37     // 0x00020000 [17] : sha256 (1)
38     // 0x00010000 [16] : pwm (1)
39     // 0x00008000 [15] : pll_usb (1)
40     // 0x00004000 [14] : pll_sys (1)
41     // 0x00002000 [13] : pio2 (1)
42     // 0x00001000 [12] : pio1 (1)
43     // 0x00000800 [11] : pio0 (1)
44     // 0x00000400 [10] : pads_qspi (1)
45     // 0x00000200 [9] : pads_bank0 (1)
46     // 0x00000100 [8] : jtag (1)
47     // 0x00000080 [7] : io_qspi (1)
48     // 0x00000040 [6] : io_bank0 (1)
49     // 0x00000020 [5] : i2c1 (1)
50     // 0x00000010 [4] : i2c0 (1)
51     // 0x00000008 [3] : hstx (1)
52     // 0x00000004 [2] : dma (1)
53     // 0x00000002 [1] : busctrl (1)

```

```

54  // 0x00000001 [0]      : adc (1)
55  io_rw_32 reset;
56
57  _REG_(RESETS_WDSEL_OFFSET) // RESETS_WDSEL
58  // 0x10000000 [28]    : usbctrl (0)
59  // 0x08000000 [27]    : uart1 (0)
60  // 0x04000000 [26]    : uart0 (0)
61  // 0x02000000 [25]    : trng (0)
62  // 0x01000000 [24]    : timer1 (0)
63  // 0x00800000 [23]    : timer0 (0)
64  // 0x00400000 [22]    : tbman (0)
65  // 0x00200000 [21]    : sysinfo (0)
66  // 0x00100000 [20]    : syscfg (0)
67  // 0x00080000 [19]    : spi1 (0)
68  // 0x00040000 [18]    : spi0 (0)
69  // 0x00020000 [17]    : sha256 (0)
70  // 0x00010000 [16]    : pwm (0)
71  // 0x00008000 [15]    : pll_usb (0)
72  // 0x00004000 [14]    : pll_sys (0)
73  // 0x00002000 [13]    : pio2 (0)
74  // 0x00001000 [12]    : pio1 (0)
75  // 0x00000800 [11]    : pio0 (0)
76  // 0x00000400 [10]    : pads_qspi (0)
77  // 0x00000200 [9]     : pads_bank0 (0)
78  // 0x00000100 [8]     : jtag (0)
79  // 0x00000080 [7]     : io_qspi (0)
80  // 0x00000040 [6]     : io_bank0 (0)
81  // 0x00000020 [5]     : i2c1 (0)
82  // 0x00000010 [4]     : i2c0 (0)
83  // 0x00000008 [3]     : hstx (0)
84  // 0x00000004 [2]     : dma (0)
85  // 0x00000002 [1]     : busctrl (0)
86  // 0x00000001 [0]     : adc (0)
87  io_rw_32 wdsel;
88
89  _REG_(RESETS_RESET_DONE_OFFSET) // RESETS_RESET_DONE
90  // 0x10000000 [28]    : usbctrl (0)
91  // 0x08000000 [27]    : uart1 (0)
92  // 0x04000000 [26]    : uart0 (0)
93  // 0x02000000 [25]    : trng (0)
94  // 0x01000000 [24]    : timer1 (0)
95  // 0x00800000 [23]    : timer0 (0)
96  // 0x00400000 [22]    : tbman (0)
97  // 0x00200000 [21]    : sysinfo (0)
98  // 0x00100000 [20]    : syscfg (0)
99  // 0x00080000 [19]    : spi1 (0)
100 // 0x00040000 [18]    : spi0 (0)
101 // 0x00020000 [17]    : sha256 (0)
102 // 0x00010000 [16]    : pwm (0)
103 // 0x00008000 [15]    : pll_usb (0)
104 // 0x00004000 [14]    : pll_sys (0)
105 // 0x00002000 [13]    : pio2 (0)
106 // 0x00001000 [12]    : pio1 (0)
107 // 0x00000800 [11]    : pio0 (0)
108 // 0x00000400 [10]    : pads_qspi (0)
109 // 0x00000200 [9]     : pads_bank0 (0)
110 // 0x00000100 [8]     : jtag (0)
111 // 0x00000080 [7]     : io_qspi (0)
112 // 0x00000040 [6]     : io_bank0 (0)
113 // 0x00000020 [5]     : i2c1 (0)
114 // 0x00000010 [4]     : i2c0 (0)
115 // 0x00000008 [3]     : hstx (0)
116 // 0x00000004 [2]     : dma (0)
117 // 0x00000002 [1]     : busctrl (0)

```

```

118     // 0x00000001 [0]      : adc (0)
119     io_ro_32 reset_done;
120 } resets_hw_t;
121
122 #define resets_hw ((resets_hw_t *)RESETS_BASE)

```

This struct defines the following registers:

- **reset**: This register contains a bit for each component that can be reset. When set to 1, the reset is asserted. If the bit is cleared, the reset is deasserted.
- **wdsel**: This register contains a bit for each component that can be reset. When set to 1, this component will reset if the watchdog fires. If you reset the power-on state machine, the entire reset controller will reset, which includes every component.
- **reset_done**: This register contains a bit for each component that is automatically set when the component is out of reset. This allows software to wait for this status bit in case the component requires initialisation before use.

The SDK defines reset functions as follows:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h Lines 156 - 158

```

156 static __force_inline void reset_block(uint32_t bits) {
157     reset_block_mask(bits);
158 }

```

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h Lines 160 - 162

```

160 static __force_inline void unreset_block(uint32_t bits) {
161     unreset_block_mask(bits);
162 }

```

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_resets/include/hardware/resets.h Lines 164 - 166

```

164 static __force_inline void unreset_block_wait(uint32_t bits) {
165     return unreset_block_mask_wait_blocking(bits);
166 }

```

One example use of reset functions is the UART driver, which defines a **uart_reset** function that selects a different bit of the reset register depending on the UART specified:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_uart/uart.c Lines 32 - 38

```

32 static inline void uart_reset(uart_inst_t *uart) {
33     reset_block_num(uart_get_reset_num(uart));
34 }
35
36 static inline void uart_unreset(uart_inst_t *uart) {
37     unreset_block_num_wait_blocking(uart_get_reset_num(uart));
38 }

```

7.5.3. List of Registers

The reset controller registers start at a base address of **0x40020000** (defined as **RESETS_BASE** in SDK).

Table 534. List of RESETS registers

Offset	Name	Info
0x0	RESET	
0x4	WDSEL	
0x8	RESET_DONE	

RESETS: RESET Register

Offset: 0x0

Table 535. RESET Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	USBCTRL		RW	0x1
27	UART1		RW	0x1
26	UART0		RW	0x1
25	TRNG		RW	0x1
24	TIMER1		RW	0x1
23	TIMER0		RW	0x1
22	TBMAN		RW	0x1
21	SYSINFO		RW	0x1
20	SYSCFG		RW	0x1
19	SPI1		RW	0x1
18	SPI0		RW	0x1
17	SHA256		RW	0x1
16	PWM		RW	0x1
15	PLL_USB		RW	0x1
14	PLL_SYS		RW	0x1
13	PIO2		RW	0x1
12	PIO1		RW	0x1
11	PIO0		RW	0x1
10	PADS_QSPI		RW	0x1
9	PADS_BANK0		RW	0x1
8	JTAG		RW	0x1
7	IO_QSPI		RW	0x1
6	IO_BANK0		RW	0x1
5	I2C1		RW	0x1
4	I2C0		RW	0x1
3	HSTX		RW	0x1
2	DMA		RW	0x1
1	BUSCTRL		RW	0x1

Bits	Name	Description	Type	Reset
0	ADC		RW	0x1

RESETS: WDSEL Register

Offset: 0x4

Table 536. WDSEL Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	USBCTRL		RW	0x0
27	UART1		RW	0x0
26	UART0		RW	0x0
25	TRNG		RW	0x0
24	TIMER1		RW	0x0
23	TIMER0		RW	0x0
22	TBMAN		RW	0x0
21	SYSINFO		RW	0x0
20	SYSCFG		RW	0x0
19	SPI1		RW	0x0
18	SPI0		RW	0x0
17	SHA256		RW	0x0
16	PWM		RW	0x0
15	PLL_USB		RW	0x0
14	PLL_SYS		RW	0x0
13	PIO2		RW	0x0
12	PIO1		RW	0x0
11	PIO0		RW	0x0
10	PADS_QSPI		RW	0x0
9	PADS_BANK0		RW	0x0
8	JTAG		RW	0x0
7	IO_QSPI		RW	0x0
6	IO_BANK0		RW	0x0
5	I2C1		RW	0x0
4	I2C0		RW	0x0
3	HSTX		RW	0x0
2	DMA		RW	0x0
1	BUSCTRL		RW	0x0
0	ADC		RW	0x0

RESETS: RESET_DONE Register

Offset: 0x8

Table 537.
RESET_DONE Register

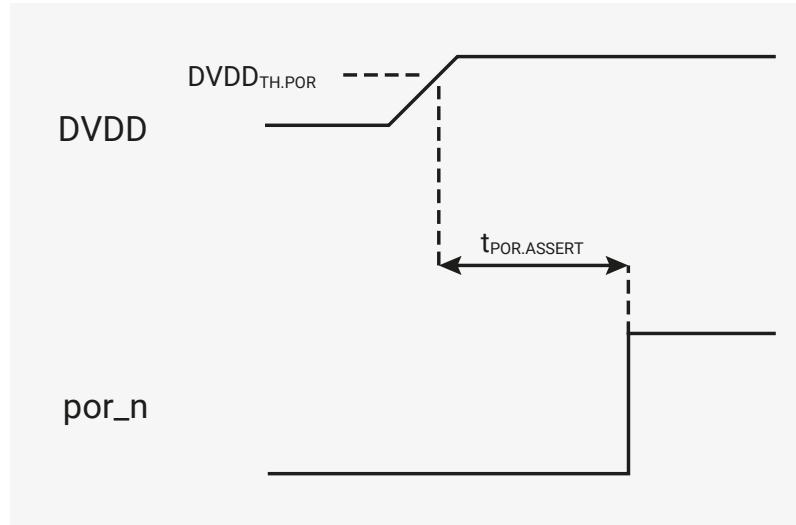
Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	USBCTRL		RO	0x0
27	UART1		RO	0x0
26	UART0		RO	0x0
25	TRNG		RO	0x0
24	TIMER1		RO	0x0
23	TIMER0		RO	0x0
22	TBMAN		RO	0x0
21	SYSINFO		RO	0x0
20	SYSCFG		RO	0x0
19	SPI1		RO	0x0
18	SPI0		RO	0x0
17	SHA256		RO	0x0
16	PWM		RO	0x0
15	PLL_USB		RO	0x0
14	PLL_SYS		RO	0x0
13	PIO2		RO	0x0
12	PIO1		RO	0x0
11	PIO0		RO	0x0
10	PADS_QSPI		RO	0x0
9	PADS_BANK0		RO	0x0
8	JTAG		RO	0x0
7	IO_QSPI		RO	0x0
6	IO_BANK0		RO	0x0
5	I2C1		RO	0x0
4	I2C0		RO	0x0
3	HSTX		RO	0x0
2	DMA		RO	0x0
1	BUSCTRL		RO	0x0
0	ADC		RO	0x0

7.6. Power-on Reset & Brownout Detection

7.6.1. Power-on Reset (POR)

The power-on reset block ensures the chip starts up cleanly when power is first applied. It accomplishes this by holding the chip in reset until the digital core supply ($DVDD$) reaches a voltage high enough to reliably power the chip's core logic. The block holds its por_n output low until $DVDD$ exceeds the **power-on reset threshold** ($DVDD_{TH.POR}$) for a period greater than the **power-on reset assertion delay** ($t_{POR ASSERT}$). Once high, por_n remains high even if $DVDD$ subsequently falls below $DVDD_{TH.POR}$. The behaviour of por_n when power is applied is shown in Figure 25.

Figure 25. A power-on reset cycle



$DVDD_{TH.POR}$ is fixed at a nominal 0.957V, which should result in a threshold between 0.924V and 0.99V. The threshold assumes a nominal $DVDD$ of 1.1V at initial power-on, and por_n may never go high if a lower voltage is used. Once the chip is out of reset, $DVDD$ can be reduced without por_n going low.

7.6.1.1. Detailed Specifications

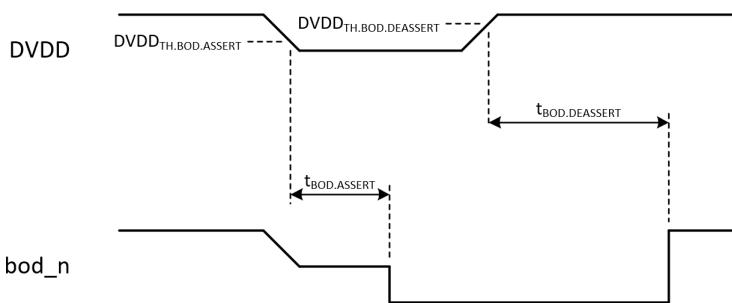
Table 538. Power-on Reset Parameters

Parameter	Description	Min	Typ	Max	Units
$DVDD_{TH.POR}$	power-on reset threshold	0.924	0.957	0.99	V
$t_{POR ASSERT}$	power-on reset assertion delay		3	10	μ s

7.6.2. Brownout Detection (BOD)

The brownout detection block prevents unreliable operation when the digital core supply ($DVDD$) drops below a safe operating level. If enabled, the block resets the chip by taking its bor_n output low when $DVDD$ drops below the **brownout detection assertion threshold** ($DVDD_{TH.BOD ASSERT}$) for a period greater than the **brownout detection assertion delay** ($t_{BOD ASSERT}$). If $DVDD$ subsequently rises above the **brownout detection deassertion threshold** ($DVDD_{TH.BOD DEASSERT}$) for a period greater than the **brownout detection deassertion delay** ($t_{BOD DEASSERT}$), the block releases reset by taking bor_n high. A brownout, followed by supply recovery, is shown in Figure 26.

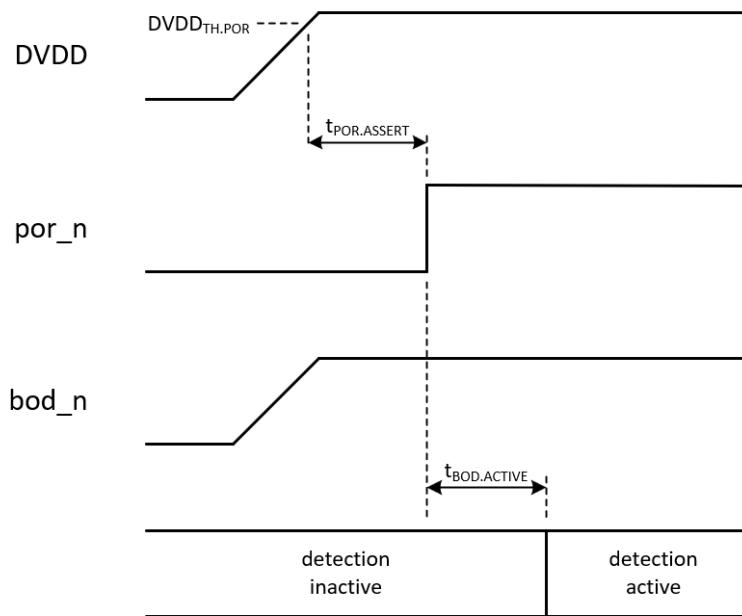
Figure 26. A brownout detection cycle



7.6.2.1. Detection Enable

Brownout detection is always enabled at initial power-on. There is, however, a short delay, the **brownout detection activation delay** ($t_{BOD.ACTIVE}$), between `por_n` going high and detection becoming active. This is shown in Figure 27.

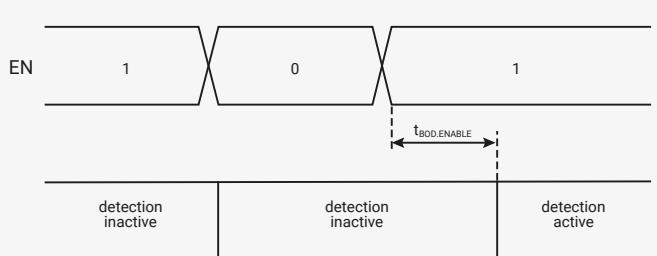
Figure 27. Activation of brownout detection at initial power-on and following a brownout event.



Once the chip is out of reset, detection can be disabled under software control. This saves a small amount of power. If detection is subsequently re-enabled, there will be another short delay, the **brownout detection enable delay** ($t_{BOD.ENABLE}$), before it becomes active again. This is shown in Figure 28.

Detection is disabled by writing a `0` to the `EN` field in the `BOD` register and is re-enabled by writing a `1` to the same field. The block's `bod_n` output is high when detection is disabled.

Figure 28. Disabling and enabling brownout detection



Detection is re-enabled if the `BOD` register is reset, as this sets the register's `EN` field to `1`. Again, detection will become active after a delay equal to the brownout detection enable delay ($t_{BOD.ENABLE}$).

NOTE

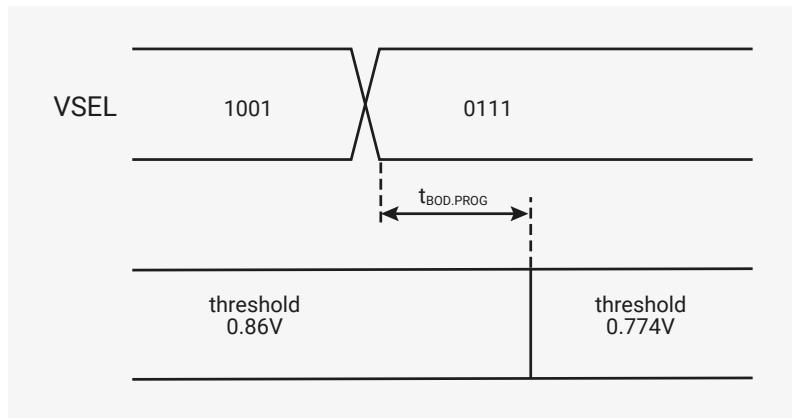
If the **BOD** register is reset by a power-on or brownout-initiated reset, the delay between the register being reset and brownout detection becoming active will be equal to the brownout detection activation delay ($t_{BOD,ACTIVE}$). The delay will be equal to the brownout detection enable delay ($t_{BOD,ENABLE}$) for all other reset sources.

7.6.2.2. Adjusting the Detection Threshold

The **brownout detection threshold** ($DVDD_{TH,BOD}$) has a nominal value of 0.946V at initial power-on or after a reset event. This should result in a detection threshold between 0.913V and 0.979V. Once out of reset, the threshold can be adjusted under software control. The new detection threshold will take effect after the **brownout detection programming delay** ($t_{BOD,PROG}$). An example of this is shown in [Figure 29](#).

The threshold is adjusted by writing to the **VSEL** field in the **BOD** register. See the **BOD** register description for details.

Figure 29. Adjusting the brownout detection threshold



7.6.2.3. Detailed Specifications

Table 539. Brownout Detection Parameters

Parameter	Description	Min	Typ	Max	Units
$DVDD_{TH,BOD,ASSERT}$	brownout detection assertion threshold	96.5	100	103.5	% of selected threshold voltage
$DVDD_{TH,BOD,DEASSERT}$	brownout detection deassertion threshold	97.4	101	105	% of selected threshold voltage
$t_{BOD,ACTIVE}$	brownout detection activation delay		55	80	μs
$t_{BOD,ASSERT}$	brownout detection assertion delay		3	10	μs
$t_{BOD,DEASSERT}$	brownout detection deassertion delay		55	80	μs

Parameter	Description	Min	Typ	Max	Units
$t_{BOD.ENABLE}$	brownout detection enable delay		35	55	μs
$t_{BOD.PROG}$	brownout detection programming delay		20	30	μs

7.6.3. Supply Monitor

The power-on and brownout reset blocks are powered by the core voltage regulator's analogue supply ([VREG_AVDD](#)). The blocks are initialised when power is first applied, but may not be reliably re-initialised if power is removed and then re-applied before [VREG_AVDD](#) has dropped to a sufficiently low level. To prevent this happening, [VREG_AVDD](#) is monitored and the power-on reset block is re-initialised if it drops below the **VREG_AVDD activation threshold** ([VREG_AVDD_{TH.ACTIVE}](#)). [VREG_AVDD_{TH.ACTIVE}](#) is fixed at a nominal 1.1V, which should result in a threshold between 0.87V and 1.26V. This threshold does not represent a safe operating voltage. Instead, it represents the voltage that [VREG_AVD](#) must drop below to reliably re-initialise the power-on reset block. For safe operation, [VREG_AVDD](#) must be at a nominal voltage of 3.3V. See [\[power-supply-spec-table\]](#).

7.6.3.1. Detailed Specifications

Table 540. Voltage Regulator Input Supply Monitor Parameters

Parameter	Description	Min	Typ	Max	Units
$VREG_{VIN_{TH.ACTIVE}}$	VREG_VIN activation threshold	0.87	1.1	1.26	V

7.6.4. List of Registers

The chip-level reset subsystem shares a register address space with other power management subsystems in the always-on domain. The address space is referred to as [POWMAN](#) elsewhere in this document. A complete list of [POWMAN](#) registers is provided in [Section 6.6](#), but information on registers associated with the brownout detector are repeated here.

The [POWMAN](#) registers start at a base address of [0x40100000](#) (defined as [POWMAN_BASE](#) in SDK).

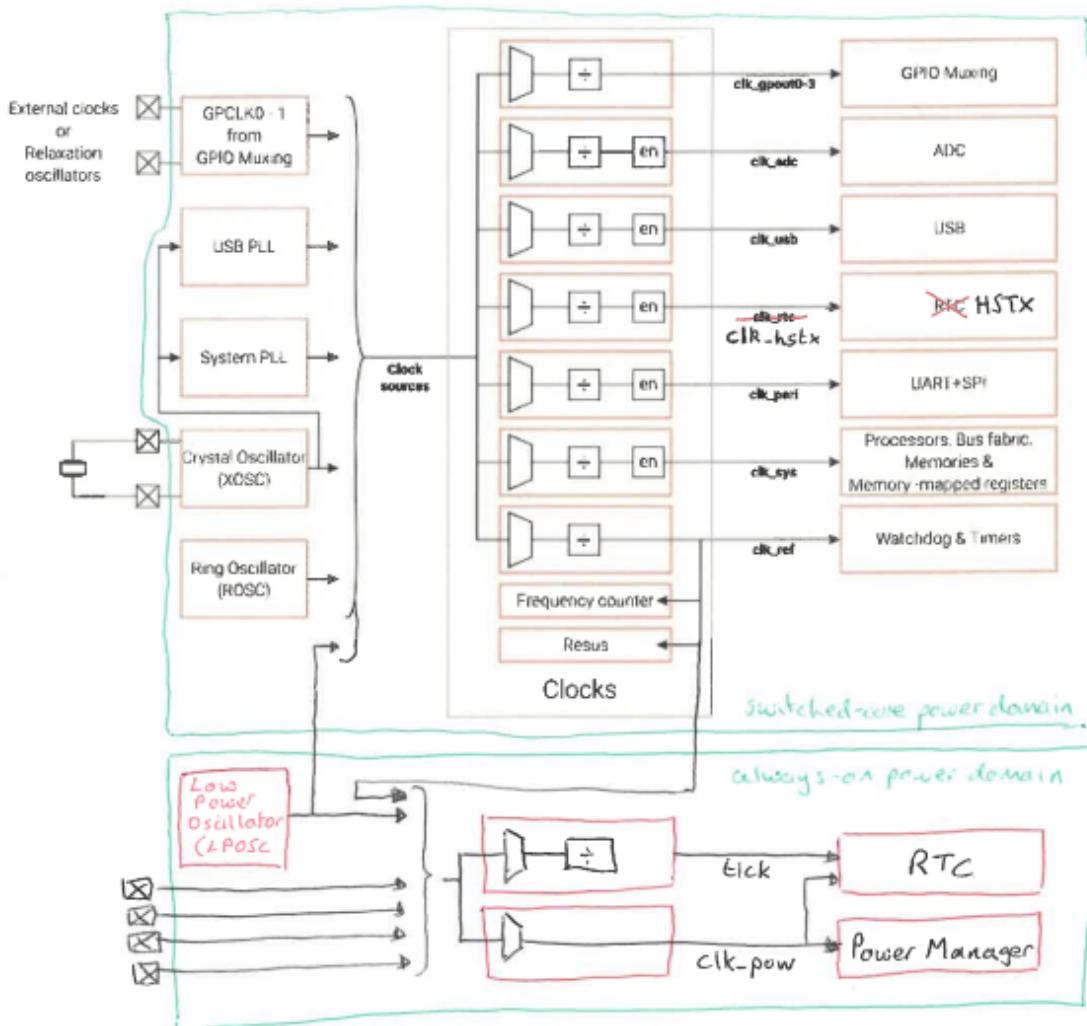
- [BOD_CTRL](#)
- [BOD](#)
- [BOD_LP_ENTRY](#)
- [BOD_LP_EXIT](#)

Chapter 8. Clocks

8.1. Overview

The clocks block provides independent clocks to on-chip and external components. It takes inputs from a variety of clock sources, allowing the user to trade off performance against cost, board area and power consumption. From these sources it uses multiple clock generators to provide the required clocks. This architecture allows the user flexibility to start and stop clocks independently and to vary some clock frequencies whilst maintaining others at their optimum frequencies.

Figure 30. Clocks overview



Please update
clocks-block-overview.ai

The Crystal Oscillator (XOSC) provides a reference to two PLLs which provide high precision clocks to the processors and peripherals. These are slow to start when waking from the various low power modes, so the on-chip Ring Oscillator (ROSC) is provided to boot the device until they are available. When the switched-core is powered down or the device is in DORMANT mode (see Section 6.3.5) the on-chip 32kHz Low Power Oscillator (LPOSC) provides a clock to the power

manager and a tick to the Always-on Timer (AON-Timer).

The clock generators select from the clock sources and optionally divide the selected clock before outputting through enable logic which provides automatic clock disabling in sleep mode (see [Section 8.1.2.5.2](#)).

An on-chip frequency counter facilitates debugging of the clock setup and also allows measurement of the frequencies of LPOSC, ROSC and external clocks. If the system clock stops accidentally, the on-chip `resus` (short for *resuscitate*) component restarts it from a known good clock. This allows the software debugger to access registers and debug the problem.

When the switched-core is powered, the power manager clock automatically switches to the reference clock (`clk_ref`). The user can optionally switch the AON-Timer tick, though we recommend waiting until `clk_ref` is running from the XOSC, because the ROSC frequency is imprecise.

You can substitute the clock sources with up to 2 GPIO clock inputs. This helps avoid adding a second crystal into systems that already have an accurate clock source and enables replacement of the ROSC and LPOSC with more accurate external sources.

You can also output up to 4 generated clocks to GPIOs at up to 50MHz. This enables you to supply clocks to external devices, reducing the need for additional clock components that consume power and board area.

8.1.1. Clock sources

RP2350 can use a variety of clock sources. This flexibility allows the user to optimise the clock setup for performance, cost, board area and power consumption. RP2350 supports the following potential clock sources:

- on-chip 32kHz Low Power Oscillator ([Section 8.4](#))
- on-chip Ring Oscillator ([Section 8.3](#))
- Crystal Oscillator ([Section 8.2](#))
- external clocks from GPIOs ([Section 8.1.5.4](#)) and PLLs ([Section 8.6](#))

The list of clock sources is different per clock generator and can be found as enumerated values in the `CTRL` register. See `CLK_SYS_CTRL` as an example.

8.1.1.1. Low Power Oscillator

The on-chip 32kHz Low Power Oscillator ([Section 8.4](#)) requires no external components. It starts automatically when the always-on domain is powered, providing a clock for the power manager and a tick for the Always-on Timer (AON-Timer) when the switched-core power domain is powered off.

The LPOSC can be tuned to 1% accuracy, and the divider in the AON-Timer tick generator can further tune the 1ms tick. However, the LPOSC frequency varies with voltage and temperature, so fine-tuning is only useful in systems with stable voltage and temperature.

When the switched-core is powered, the LPOSC clock can drive the reference clock (`clk_ref`), which in turn can drive the system clock (`clk_sys`). This allows another low power mode where the processors remain powered but, unlike the SLEEP and DORMANT modes, clocks are running. The LPOSC clock can also be sent to the frequency counter for calibration or output to a GPIO.

8.1.1.2. Ring Oscillator

The on-chip Ring Oscillator ([Section 8.3](#)) requires no external components. It starts automatically when the switched-core domain is powered and is used to clock the chip during the initial boot stages. During boot, the ROSC runs at a nominal 11MHz, but varies with PVT (Process, Voltage, and Temperature). The ROSC frequency is guaranteed to be in the range 4.6MHz to 19.6MHz.

For low-cost applications where frequency accuracy is unimportant, the chip can continue to run from the ROSC. If your application requires greater performance, the frequency can be increased by programming the registers as described in [Section 8.3](#). Because the frequency varies with PVT (Process, Voltage, and Temperature), the user must take care to avoid exceeding the maximum frequencies described in the clock generators section. For information about reducing this variation when running the ROSC at frequencies close to the maximum, see [Section 8.1.1.2.1](#). Alternatively, use an external clock or the XOSC to provide a stable reference clock and use the PLLs to generate higher frequencies. However, this approach requires external components, which will cost board area and increase power consumption.

When using an external clock or the XOSC, you can stop the ROSC to save power. Before stopping the ROSC, you must switch the reference clock generator and the system clock generator to an alternate source.

The ROSC is unpowered when the switched-core domain is powered down, but starts immediately when the switched-core powers up. It is not affected by sleep mode. To save power, reduce the frequency before entering sleep mode. When entering DORMANT mode, the ROSC is automatically stopped. When exiting DORMANT mode, the ROSC restarts in the same configuration. If you drive clocks at close to their maximum frequencies with the ROSC, drop the frequency before entering SLEEP or DORMANT mode. This allows for frequency variation due to changes in environmental conditions during SLEEP or DORMANT mode.

To use ROSC clock externally, output it to a GPIO pin using one of the [clk_gpc1k0-3](#) generators.

The following sections describe techniques for mitigating PVT variation of the ROSC frequency. They also provide some interesting design challenges for use in teaching both the effects of PVT and writing software to control real time functions.

 **TIP**

Because the ROSC frequency varies with PVT (Process, Voltage, and Temperature), you can use the ROSC frequency to measure any one of the three PVT variables as long as you know the other two variables.

8.1.1.2.1. Mitigate ROSC frequency variation due to process

Process varies for the following reasons:

- Chips leave the factory with a spread of process parameters. This causes variation in the ROSC frequency across chips.
- Process parameters vary slightly as the chip ages. This is only observable over many thousands of hours of operation.

To mitigate process variation, the user can characterise individual chips and program the ROSC frequency accordingly. This is an adequate solution for small numbers of chips, but does not scale well to volume production. For high-volume applications, consider using [automatic mitigation](#).

8.1.1.2.2. Mitigate ROSC frequency variation due to voltage

Supply voltage varies for the following reasons:

- The power supply itself may vary.
- As chip activity varies, on-chip IR varies.

To mitigate voltage variation, calibrate for the minimum performance target of your application, then adjust the ROSC frequency to always exceed that minimum.

8.1.1.2.3. Mitigate ROSC frequency variation due to temperature

Temperature varies for the following reasons:

- The ambient temperature may vary.

- The chip temperature varies as chip activity varies due to self-heating.

To mitigate temperature variations, stabilise the temperature. You can use a temperature controlled environment, passive cooling, or active cooling. Alternatively, track the temperature using the on-chip temperature sensor and adjust the ROSC frequency so it remains within the required bounds.

8.1.1.2.4. Automatic mitigation of ROSC frequency variation due to PVT

Techniques for automatic ROSC frequency control avoid the need to calibrate individual chips, but require periodic access to a clock reference or to a time reference.

If a clock reference is available, you can use it to periodically measure the ROSC frequency and adjust accordingly. The on-chip XOSC is one potential clock reference. You can even run the XOSC intermittently to save power for very low power application where it is too costly to run the XOSC continuously or use the PLLs to achieve high frequencies.

If a time reference is available, you can clock the on-chip AON-Timer from the ROSC and periodically compare it against the time reference, adjusting the ROSC frequency as necessary. Using these techniques, the ROSC frequency still drifts due to voltage and temperature variation. As a result, you should also implement mitigations for voltage and temperature to ensure that variations do not allow the ROSC frequency to drift out of the acceptable range.

8.1.1.2.5. Automatic overclocking using the ROSC

The datasheet maximum frequencies for any digital device are quoted for worst case PVT. Most chips in most normal environments can run significantly faster than the quoted maximum, and therefore support overclocking. When RP2350 runs from the ROSC, PVT affects both the ROSC and the digital components. As the ROSC gets faster, the processors can also run faster. This means the user can overclock from the ROSC, then rely on the ROSC frequency tracking with PVT variations. The tracking of ROSC frequency and the processor capability is not perfect, and currently there is insufficient data to specify a safe ROSC setting for this mode of operation, so some experimentation is required.

This mode of operation maximises processor performance, but causes variations in the time taken to complete a task. Only use overclocking for applications where this variation is acceptable. If your application uses frequency sensitive interfaces such as USB or UART, you must use the XOSC and PLL to provide a precise clock for those components.

8.1.1.3. Crystal Oscillator

The Crystal Oscillator (Section 8.2) provides a precise, stable clock reference and should be used where accurate timing is required and no suitable external clocks are available. The XOSC requires an external crystal component. The external crystal determines the frequency. The RP2350 supports 1MHz to 100MHz crystals and the RP2350 reference design (see [Hardware design with RP2040, Minimal Design Example](#)) uses a 12MHz crystal. Using the XOSC and the PLLs, you can run on-chip components at their maximum frequencies. Appropriate margin is built into the design to tolerate up to 1000ppm variation in the XOSC frequency.

The XOSC is unpowered when the switched-core domain is powered down. It remains inactive when the switched-core is powered up. If required, you must enable it in software. XOSC startup takes several milliseconds, and software must wait for the `XOSC_STABLE` flag to be set before starting the PLLs and changing any clock generators. Before the XOSC completes startup, output may be non-existent or exhibit very short pulse widths; this will corrupt logic if used. Once XOSC startup is complete, the reference clock (`clk_ref`) and the system clock (`clk_sys`) can run from the XOSC. If you switch the system and reference clocks to run from the XOSC, you can stop the ROSC to save power.

The XOSC is not affected by sleep mode. It automatically stops and restarts in the same configuration when entering and exiting DORMANT mode.

To use the XOSC clock externally, output it to a GPIO pin using one of the `clk_gpclk0-clk_gpclk03` generators. You cannot take XOSC output directly from the XIN (XI) or XOUT (XO) pins.

8.1.1.4. External Clocks

If external clocks exist in the hardware design, you can use them to clock RP2350. You can use clocks individually or in conjunction with the other (internal or external) clock sources. Use XIN and one of GPIN0-GPIN1 to input external clocks.

If you drive an external clock into XIN, you don't need an external crystal. When driving an external clock into XIN, you must configure the XOSC to pass through the XIN signal. When the switched-core powers down, this configuration will be lost, but the configuration is unaffected by SLEEP and DORMANT modes. The input is limited to 50MHz, but the on-chip PLLs can synthesise higher frequencies from the XIN input if required.

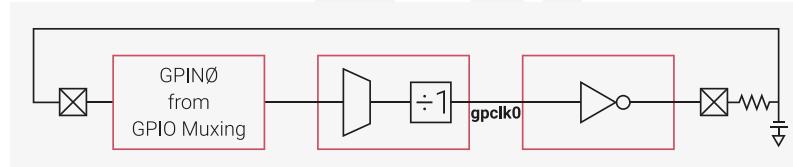
GPIN0-GPIN1 can provide system and peripherals clocks, but is limited to 50MHz. This can potentially save power and allows components on the RP2350 to run synchronously with external components, which simplifies data transfer between chips. If the frequency accuracy of the external clocks is poorer than 1000ppm, the generated clocks should not run at their maximum frequencies since they could exceed their design margins. Once the external clocks begin to run, the reference clock (`clk_ref`) and the system clock (`clk_sys`) can run from the external clocks and you can stop the ROSC to save power. When the switched-core powers down, GPIN0-GPIN1 configuration will be lost, but the configuration is unaffected by SLEEP and DORMANT modes.

To provide a more accurate tick to the AON-Timer, use one of the GPIN0-GPIN3 inputs to replace the clock from the LPOSC. These inputs are limited to 29MHz. GPIN0-GPIN3 configuration is unaffected by switched-core power down, sleep mode, and DORMANT mode.

8.1.1.5. Relaxation Oscillators

If there is no appropriate clock available, but you still want to replace or supplement external clocks with another clock source, you can construct one or two relaxation oscillators from external passive components. Send the clock source (GPIN0-GPIN1) to one of the `clk_gpclk0-clk_gpclk03` generators, invert it through the GPIO inverter `OUTOVER`, and connect back to the clock source input via an RC circuit:

Figure 31. Simple relaxation oscillator example



The frequency of clocks generated from relaxation oscillators depend on the delay through the chip and the drive current from the GPIO output, both of which vary with PVT. The frequency and frequency accuracy depend on the quality and accuracy of the external components. More elaborate external components such as ceramic resonators, can improve performance, but also increase cost and complexity. Such an oscillator will not achieve 1000ppm, so they cannot drive internal clocks at their maximum frequencies. To drive internal clocks at the maximum possible frequency, use the XOSC.

The configuration of the relaxation oscillators will be lost when the switched-core powers down, but is not affected by sleep mode or DORMANT mode.

8.1.1.6. PLLs

The PLLs (Section 8.6) are used to provide fast clocks when running from the XOSC or an external clock source driven into the XIN pin. In a fully-featured application, the USB PLL provides a fixed 48MHz clock to the ADC and USB while `clk_ref` is driven from the XOSC or external clock source. This allows the user to drive `clk_sys` from the system PLL and vary the frequency according to demand to save power without having to change the setups of the other clocks. `clk_peri` can be driven either from the fixed frequency USB PLL or from the variable frequency system PLL. If `clk_sys` never needs to exceed 48MHz, one PLL can be used and the divider in the `clk_sys` clock generator can scale the `clk_sys` frequency according to demand.

When a PLL starts, you cannot use the output until the PLL locks as indicated by the `LOCK` bit in the `STATUS` register. As a

result, the PLL output cannot be used during changes to the reference clock divider, the output dividers or the bypass mode. The output can be used during feedback divisor changes, though the output frequency may overshoot or undershoot during large changes to the feedback divisor. For more information, see [Section 8.6](#).

The PLLs can drive clocks at their maximum frequency as long as the reference clock is accurate to 1000ppm, since this keeps the frequency of the generated clocks within design margins.

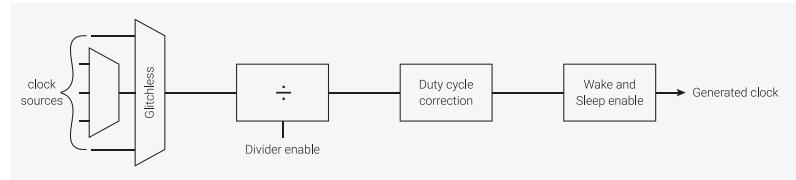
The PLLs are not affected by sleep mode. To save power in sleep mode, switch all clock generators away from the PLLs stop them in software before entering sleep mode.

The PLLs do not stop and restart automatically when entering and exiting DORMANT mode. If the PLLs are running when entering DORMANT mode, they will be corrupted because the reference clock in the XOSC stops. This generates out-of-control clocks that consume power unnecessarily. Before entering DORMANT mode, always switch all clock generators away from the PLLs and stop the PLLs in software.

8.1.2. Clock Generators

The clock generators are built on a standard design which incorporates clock source multiplexing, division, duty cycle correction and sleep mode enabling. To save chip area and power, individual clock generators do not support all features.

Figure 32. A generic clock generator



8.1.2.1. Instances

RP2350 has several clock generators which are listed below.

Table 541. RP2350 clock generators

Clock	Description	Nominal Frequency
<code>clk_gpout0</code>	Clock output to GPIO. Can be used to clock external devices or debug on chip clocks with a logic analyser or oscilloscope.	N/A
<code>clk_gpout1</code>		
<code>clk_gpout2</code>		
<code>clk_gpout3</code>		
<code>clk_ref</code>	Reference clock that is always running unless in DORMANT mode. Runs from Ring Oscillator (ROSC) at power-up but can be switched to Crystal Oscillator (XOSC) for more accuracy.	6 - 12MHz
<code>clk_sys</code>	System clock that is always running unless in DORMANT mode. Runs from <code>clk_ref</code> at power-up, but is typically switched to a PLL.	150MHz
<code>clk_peri</code>	Peripheral clock. Typically runs from <code>clk_sys</code> but allows peripherals to run at a consistent speed if <code>clk_sys</code> is changed by software.	12 - 150MHz
<code>clk_usb</code>	USB reference clock. Must be 48MHz.	48MHz
<code>clk_adc</code>	ADC reference clock. Must be 48MHz.	48MHz

Clock	Description	Nominal Frequency
<code>clk_hstx</code>	HSTX clock.	150MHz

For a full list of clock sources for each clock generator, see the appropriate [CTRL](#) register. For example, [CLK_SYS_CTRL](#).

8.1.2.2. Multiplexers

All clock generators have a multiplexer referred to as the auxiliary (aux) mux. This mux has a conventional design whose output will glitch when changing the select control. The reference clock (`clk_ref`) and the system clock (`clk_sys`) have an additional multiplexer referred to as the **glitchless mux**. The glitchless mux can switch between clock sources without generating a glitch on the output.

Avoid clock glitches at all costs: they may corrupt the logic running on the clock. While switching the clock source, always disable any clock generator that only has an aux mux. If the clock generator has a glitchless mux (e.g. `clk_ref` and `clk_sys`), then the glitchless mux should switch away from the aux mux while changing the aux mux source. Clock generators require two cycles of the source clock to stop the output and two cycles of the new source to restart the output. Wait for the generator to stop before changing the auxiliary mux; therefore, you must be aware of the source clock frequency.

The glitchless mux is only implemented for always-on clocks. On RP2350, the always-on clocks are the reference clock (`clk_ref`) and the system clock (`clk_sys`). Such clocks must run continuously unless the chip is in DORMANT mode. The glitchless mux has a status output ([SELECTED](#)) which indicates which source is selected. You can read this status output from software to confirm that a change of clock source has completed.

The recommended control sequences are as follows.

To switch between clock sources for the glitchless mux:

1. Switch the glitchless mux to an alternate source.
2. Poll the [SELECTED](#) register until the switch completes.

To switch between clock sources for the aux mux when the generator has a glitchless mux:

1. Switch the glitchless mux to a source that isn't the aux mux.
2. Poll the [SELECTED](#) register until the switch completes.
3. Change the auxiliary mux select control.
4. Switch the glitchless mux back to the aux mux.
5. If required, poll the [SELECTED](#) register until the switch completes.

To switch between clock sources for the aux mux when the generator does not have a glitchless mux:

1. Disable the clock divider.
2. Wait for the generated clock to stop (two cycles of the clock source).
3. Change the auxiliary mux select control.
4. Enable the clock divider.
5. If required, wait for the clock generator to restart (two cycles of the clock source).

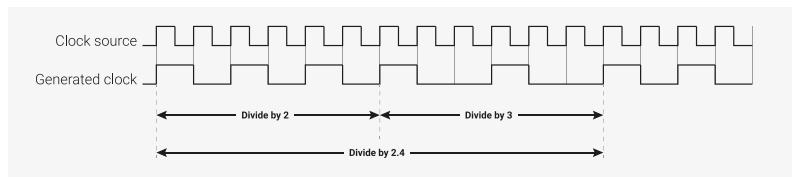
See [Section 8.1.5.1](#) for a code example of this.

8.1.2.3. Divider

A fully featured divider divides by 1 or a fractional number in the range 2.0 to 2^{16} . Fractional division is achieved by toggling between 2 integer divisors; this yields a jittery clock which may not be suitable for some applications. For example, when dividing by 2.4 the divider divides by 2 for 3 cycles and by 3 for 2 cycles. For divisors with large integer

components, the jitter will be much smaller and less critical.

Figure 33. An example of fractional division.



All dividers support **on-the-fly divisor changes**: the output clock can switch cleanly from one divisor to another. The clock generator does not need to be stopped during clock divisor changes, because the dividers synchronise the divisor change to the end of the clock cycle. Similarly, dividers synchronise the enable to the end of the clock cycle to avoid glitches when the clock generator is enabled or disabled. Clock generators for always-on clocks are permanently enabled and therefore do not have an enable control.

In the event that a clock generator locks up and never completes the current clock cycle, it can be forced to stop using the **KILL** control. This may result in an output glitch, which may corrupt the logic driven by the clock. Always reset the destination logic before using the **KILL** control. Clock generators for always-on clocks are permanently active and therefore do not have a **KILL** control.

NOTE

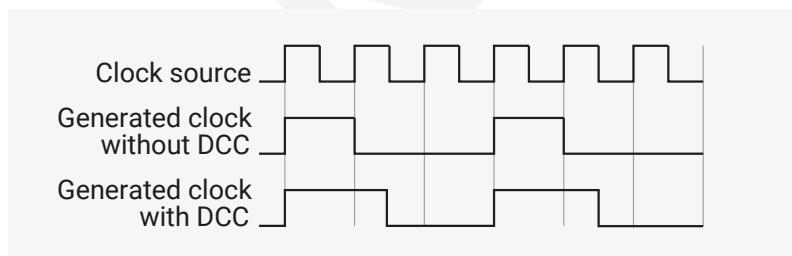
This clock generator design has been used in numerous chips and has never been known to lock up. The **KILL** control is inelegant and unnecessary and should not be used as an alternative to the enable.

8.1.2.4. Duty Cycle Correction

The divider operates on the rising edge of the input clock, so it does not generate an even duty cycle clock when dividing by odd numbers. For example, divide by 3 gives a duty cycle of 33.3%, and divide by 5 gives a duty cycle of 40%.

If enabled, duty cycle correction logic will shift the falling edge of the output clock to the falling edge of the input clock and restore a 50% duty cycle. The duty cycle correction can be enabled and disabled while the clock is running. It will not operate when dividing by an even number.

Figure 34. An example of duty_cycle_correction.



8.1.2.5. Clock Enables

Each clock goes to multiple destinations. With a few exceptions, each destination has two enables. Use the **WAKE_EN** registers to enable the clocks when the system is awake. Use the **SLEEP_EN** registers to enable the clocks when the system is in sleep mode. Enables help reduce power in the clock distribution networks for unused components. Any component which is not clocked will retain its configuration so it can restart quickly.

NOTE

By default, the `WAKE_EN` and `SLEEP_EN` registers reset to `0x1`, which enables all clocks. Only use this feature for low-power designs.

8.1.2.5.1. Clock Enable Exceptions

The following destinations do not have clock enables:

- the `clk_gpc1k0-clk_gpc1k03` generators
- the processor cores, because they require a clock at all times to manage their own power-saving features
- `clk_sys_busfabric` (in wake mode), because that would prevent the cores from accessing any chip registers, including those that control the clock enables
- `clk_sys_clocks` (in wake mode), because that would prevent the cores from accessing the clocks control registers

8.1.2.5.2. System Sleep Mode

System sleep mode is entered automatically when both cores are in sleep and the DMA has no outstanding transactions. In system sleep mode, the clock enables described in the previous paragraphs are switched from the `WAKE_EN` registers to the `SLEEP_EN` registers. Sleep mode helps reduce power consumed in the clock distribution networks when the chip is inactive. If the user has not configured the `WAKE_EN` and `SLEEP_EN` registers, system sleep does nothing.

There is little value in using system sleep without taking other measures to reduce power before the cores are put to sleep. Things to consider include:

- stop unused clock sources such as the PLLs and Crystal Oscillator
- reduce the frequencies of generated clocks by increasing the clock divisors
- stop external clocks

For maximum power saving when the chip is inactive, the user should consider DORMANT (see [Section 6.3.5](#)) mode in which clocks are sourced from the Crystal Oscillator and/or the Ring Oscillator and those clock sources are stopped.

For more information about sleep, see [Section 6.3.4](#).

8.1.3. Frequency Counter

The frequency counter measures the frequency of internal and external clocks by counting the clock edges seen over a test interval. The interval is defined by counting cycles of `clk_ref`, which must be driven either from XOSC or a stable external source of known frequency.

The user can pick between accuracy and test time using the `FC0_INTERVAL` register. [Table 542](#) shows this trade off:

Table 542. Frequency Counter Test Interval vs Accuracy

Interval Register	Test Interval	Accuracy
0	1µs	2048kHz
1	2µs	1024kHz
2	4µs	512kHz
3	8µs	256kHz
4	16µs	128kHz
5	32µs	64kHz
6	64µs	32kHz

Interval Register	Test Interval	Accuracy
7	125µs	16kHz
8	250µs	8kHz
9	500µs	4kHz
10	1ms	2kHz
11	2ms	1kHz
12	4ms	500Hz
13	8ms	250Hz
14	16ms	125Hz
15	32ms	62.5Hz

8.1.4. Resus

It is possible to write software that inadvertently stops `clk_sys`. This normally causes an unrecoverable lock-up of the cores and the on-chip debugger, leaving the user unable to trace the problem. To mitigate against unrecoverable core lock-up, an automatic **resuscitation circuit** is provided; this switches `clk_sys` to a known good clock source (`clk_ref`) if it detects no edges over a user-defined interval. `clk_ref` can be driven from the XOSC, ROSC or an external source. The interval is programmable via `CLK_SYS_RESUS_CTRL`.

 **WARNING**

There is no way for resus to revive the chip if `clk_ref` is also stopped.

To enable the resus:

- set the timeout interval
- set the `ENABLE` bit in `CLK_SYS_RESUS_CTRL`

To detect a resus event:

- enable the `CLK_SYS_RESUS` interrupt by setting the interrupt enable bit in `INTE`
- enable the `CLOCKS_DEFAULT_IRQ` processor interrupt (see [Section 3.2](#))

Resus is intended as a debugging aid, so the user can trace the software error that triggered the resus, then correct the error and reboot. It is possible to continue running after a resus event by reconfiguring `clk_sys`, then clearing the resus by writing the `CLEAR` bit in `CLK_SYS_RESUS_CTRL`.

 **WARNING**

Only use resus for debugging. If `clk_sys` runs slower than expected, a resus could trigger. This could result in a `clk_sys` glitch, which could corrupt the chip.

8.1.5. Programmer's Model

8.1.5.1. Configuring a clock generator

The SDK defines an enum of clocks:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2350/hardware_structs/include/hardware/structs/clocks.h Lines 27 - 39

```

27 enum clock_index {
28     clk_gpout0 = 0,      ///<> GPIO Muxing 0
29     clk_gpout1,          ///<> GPIO Muxing 1
30     clk_gpout2,          ///<> GPIO Muxing 2
31     clk_gpout3,          ///<> GPIO Muxing 3
32     clk_ref,             ///<> Watchdog and timers reference clock
33     clk_sys,              ///<> Processors, bus fabric, memory, memory mapped registers
34     clk_peri,             ///<> Peripheral clock for UART and SPI
35     clk_hstx,             ///<> Peripheral clock for HSTX
36     clk_usb,              ///<> USB clock
37     clk_adc,              ///<> ADC clock
38     CLK_COUNT
39 };

```

Additionally, the SDK defines a struct to describe the registers of a clock generator:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2350/hardware_structs/include/hardware/structs/clocks.h Lines 43 - 64

```

43 typedef struct {
44     _REG_(CLOCKS_CLK_GPOUT0_CTRL_OFFSET) // CLOCKS_CLK_GPOUT0_CTRL
45     // Clock control, can be changed on-the-fly (except for auxsrc)
46     // 0x10000000 [28] : ENABLED (0): clock generator is enabled
47     // 0x00100000 [20] : NUDGE (0): An edge on this signal shifts the phase of the output by
48     // 1 cycle of the input clock
49     // 0x00030000 [17:16] : PHASE (0): This delays the enable signal by up to 3 cycles of the
50     // input clock
51     // 0x00001000 [12] : DC50 (0): Enables duty cycle correction for odd divisors, can be
52     // changed on-the-fly
53     // 0x00000800 [11] : ENABLE (0): Starts and stops the clock generator cleanly
54     // 0x00000400 [10] : KILL (0): Asynchronously kills the clock generator, enable must be
55     // set low before deasserting kill
56     // 0x000001e0 [8:5] : AUXSRC (0): Selects the auxiliary clock source, will glitch when
57     // switching
58     io_rw_32 ctrl;
59
60     _REG_(CLOCKS_CLK_GPOUT0_DIV_OFFSET) // CLOCKS_CLK_GPOUT0_DIV
61     // 0xfffff000 [31:16] : INT (1): Integer part of clock divisor, 0 -> max+1, can be changed
62     // on-the-fly
63     // 0x0000ffff [15:0] : FRAC (0): Fractional component of the divisor, can be changed on-
64     // the-fly
65     io_rw_32 div;
66
67     _REG_(CLOCKS_CLK_GPOUT0_SELECTED_OFFSET) // CLOCKS_CLK_GPOUT0_SELECTED
68     // Indicates which src is currently selected (one-hot)
69     // 0x00000001 [0] : CLK_GPOUT0_SELECTED (1): This slice does not have a glitchless mux
70     // (only the AUX_SRC field is...
71     io_ro_32 selected;
72 } clock_hw_t;

```

Clock configuration requires the following pieces of information:

- The frequency of the clock source
- The mux / aux mux position of the clock source
- The desired output frequency

The SDK provides `clock_configure` to configure a clock:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/p2_common/hardware_clocks/clocks.c Lines 40 - 118

```

40 static void clock_configure_internal(clock_handle_t clock, uint32_t src, uint32_t auxsrc,
41     uint32_t actual_freq, uint32_t div) {
42     clock_hw_t *clock_hw = &clocks_hw->clk[clock];
43
44     // If increasing divisor, set divisor before source. Otherwise set source
45     // before divisor. This avoids a momentary overspeed when e.g. switching
46     // to a faster source and increasing divisor to compensate.
47     if (div > clock_hw->div)
48         clock_hw->div = div;
49
50     // If switching a glitchless slice (ref or sys) to an aux source, switch
51     // away from aux *first* to avoid passing glitches when changing aux mux.
52     // Assume (!!!) glitchless source 0 is no faster than the aux source.
53     if (has_glitchless_mux(clock) && src ==
54         CLOCKS_CLK_SYS_CTRL_SRC_VALUE_CLKSRC_CLK_SYS_AUX) {
55         hw_clear_bits(&clock_hw->ctrl, CLOCKS_CLK_REF_CTRL_SRC_BITS);
56         while (!(clock_hw->selected & 1u))
57             tight_loop_contents();
58     }
59     // If no glitchless mux, cleanly stop the clock to avoid glitches
60     // propagating when changing aux mux. Note it would be a really bad idea
61     // to do this on one of the glitchless clocks (clk_sys, clk_ref).
62     else {
63         // Disable clock. On clk_ref and clk_sys this does nothing,
64         // all other clocks have the ENABLE bit in the same position.
65         hw_clear_bits(&clock_hw->ctrl, CLOCKS_CLK_GPOUT0_CTRL_ENABLE_BITS);
66         if (configured_freq[clock] > 0) {
67             // Delay for 3 cycles of the target clock, for ENABLE propagation.
68             // Note XOSC_COUNT is not helpful here because XOSC is not
69             // necessarily running, nor is timer...
70             uint delay_cyc = configured_freq[clk_sys] / configured_freq[clock] + 1;
71             busy_wait_at_least_cycles(delay_cyc * 3);
72         }
73     }
74     // Set aux mux first, and then glitchless mux if this clock has one
75     hw_write_masked(&clock_hw->ctrl,
76         (auxsrc << CLOCKS_CLK_SYS_CTRL_AUXSRC_LSB),
77         CLOCKS_CLK_SYS_CTRL_AUXSRC_BITS
78     );
79     if (has_glitchless_mux(clock)) {
80         hw_write_masked(&clock_hw->ctrl,
81             src << CLOCKS_CLK_REF_CTRL_SRC_LSB,
82             CLOCKS_CLK_REF_CTRL_SRC_BITS
83         );
84         while (!(clock_hw->selected & (1u << src)))
85             tight_loop_contents();
86     }
87
88     // Enable clock. On clk_ref and clk_sys this does nothing,
89     // all other clocks have the ENABLE bit in the same position.
90     hw_set_bits(&clock_hw->ctrl, CLOCKS_CLK_GPOUT0_CTRL_ENABLE_BITS);
91
92     // Now that the source is configured, we can trust that the user-supplied
93     // divisor is a safe value.
94     clock_hw->div = div;
95     configured_freq[clock] = actual_freq;
96 }
97
98 bool clock_configure(clock_handle_t clock, uint32_t src, uint32_t auxsrc, uint32_t src_freq,

```

```

 99     assert(src_freq >= freq);
100
101     if (freq > src_freq)
102         return false;
103
104     uint32_t div = (uint32_t)((uint64_t) src_freq) << CLOCKS_CLK_GPOUT0_DIV_INT_LSB) /
freq);
105     uint32_t actual_freq = (uint32_t) (((uint64_t) src_freq) <<
CLOCKS_CLK_GPOUT0_DIV_INT_LSB) / div);
106
107     clock_configure_internal(clock, src, auxsrc, actual_freq, div);
108     // Store the configured frequency
109     return true;
110 }
111
112 void clock_configure_int_divider(clock_handle_t clock, uint32_t src, uint32_t auxsrc,
uint32_t src_freq, uint32_t int_divider) {
113     clock_configure_internal(clock, src, auxsrc, src_freq / int_divider, int_divider <<
CLOCKS_CLK_GPOUT0_DIV_INT_LSB);
114 }
115
116 void clock_configure_undivided(clock_handle_t clock, uint32_t src, uint32_t auxsrc, uint32_t
src_freq) {
117     clock_configure_internal(clock, src, auxsrc, src_freq, 1u <<
CLOCKS_CLK_GPOUT0_DIV_INT_LSB);
118 }

```

`clocks_init` calls `clock_configure` for each clock. The following example shows the `clk_sys` configuration:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/p2_common/pico_runtime_init/runtime_init_clocks.c Lines 90 - 94

```

90     // CLK_SYS = PLL_SYS (usually) 125MHz / 1 = 125MHz
91     clock_configure_undivided(clk_sys,
92                             CLOCKS_CLK_SYS_CTRL_SRC_VALUE_CLKSRC_CLK_SYS_AUX,
93                             CLOCKS_CLK_SYS_CTRL_AUXSRC_VALUE_CLKSRC_PLL_SYS,
94                             SYS_CLK_HZ);

```

Once a clock is configured, call `clock_get_hz` to get the output frequency in Hz.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/p2_common/hardware_clocks/clocks.c Lines 123 - 125

```

123 uint32_t clock_get_hz(clock_handle_t clock) {
124     return configured_freq[clock];
125 }

```

⚠️ WARNING

The frequency returned by `clock_get_hz` will be inaccurate if the provided source frequency is incorrect.

8.1.5.2. Using the frequency counter

To use the frequency counter, the programmer must:

1. Set the reference frequency: `clk_ref`.
2. Set the mux position of the source they want to measure. See `FC0_SRC`.

3. Wait for the `DONE` status bit in `FC0_STATUS` to be set.

4. Read the result.

The SDK defines a `frequency_count` function which takes the source as an argument and returns the frequency in kHz:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_clocks/clocks.c Lines 133 - 160

```

133 uint32_t frequency_count_khz(uint src) {
134     fc_hw_t *fc = &clocks_hw->fc0;
135
136     // If frequency counter is running need to wait for it. It runs even if the source is NULL
137     while(fc->status & CLOCKS_FC0_STATUS_RUNNING_BITS) {
138         tight_loop_contents();
139     }
140
141     // Set reference freq
142     fc->ref_khz = clock_get_hz(clk_ref) / 1000;
143
144     // FIXME: Don't pick random interval. Use best interval
145     fc->interval = 10;
146
147     // No min or max
148     fc->min_khz = 0;
149     fc->max_khz = 0xffffffff;
150
151     // Set SRC which automatically starts the measurement
152     fc->src = src;
153
154     while(!(fc->status & CLOCKS_FC0_STATUS_DONE_BITS)) {
155         tight_loop_contents();
156     }
157
158     // Return the result
159     return fc->result >> CLOCKS_FC0_RESULT_KHZ_LSB;
160 }
```

There is also a wrapper function to change the unit to MHz:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_clocks/include/hardware_clocks.h Lines 271 - 273

```

271 static inline float frequency_count_mhz(uint src) {
272     return ((float) (frequency_count_khz(src))) / KHZ;
273 }
```

NOTE

The frequency counter can also be used in a test mode. This allows the hardware to check if the frequency is between a minimum and a maximum frequency, set in `FC0_MIN_KHZ` and `FC0_MAX_KHZ`. This mode will set one of the following bits in `FC0_STATUS` when `DONE` is set:

- `SLOW`: if the frequency is below the specified range
- `PASS`: if the frequency is within the specified range
- `FAST`: if the frequency is above the specified range
- `DIED`: if the clock is stopped or stops running

Test mode will also set the `FAIL` bit if `DIED`, `FAST`, or `SLOW` are set.

8.1.5.3. Configuring a GPIO output clock

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_clocks/clocks.c Lines 231 - 255

```

231 void clock_gpio_init_int_frac(uint gpio, uint src, uint32_t div_int, uint8_t div_frac) {
232     // Bit messy but it's as much code to loop through a lookup
233     // table. The sources for each gput generators are the same
234     // so just call with the sources from GP0
235     uint gpclk = 0;
236     if (gpio == 21) gpclk = clk_gpout0;
237     else if (gpio == 23) gpclk = clk_gpout1;
238     else if (gpio == 24) gpclk = clk_gpout2;
239     else if (gpio == 25) gpclk = clk_gpout3;
240 #if !PICO_RP2040
241     else if (gpio == 13) gpclk = clk_gpout0;
242     else if (gpio == 15) gpclk = clk_gpout1;
243 #endif
244     else {
245         invalid_params_if(HARDWARE_CLOCKS, true);
246     }
247
248     // Set up the gpclk generator
249     clocks_hw->clk[gpclk].ctrl = (src << CLOCKS_CLK_GPOUT0_CTRL_AUXSRC_LSB) |
250                                     CLOCKS_CLK_GPOUT0_CTRL_ENABLE_BITS;
251     clocks_hw->clk[gpclk].div = (div_int << CLOCKS_CLK_GPOUT0_DIV_INT_LSB) | div_frac;
252
253     // Set gpio pin to gpclock function
254     gpio_set_function(gpio, GPIO_FUNC_GPCK);
255 }
```

8.1.5.4. Configuring a GPIO input clock

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_clocks/clocks.c Lines 292 - 322

```

292 bool clock_configure_gpin(clock_handle_t clock, uint gpio, uint32_t src_freq, uint32_t freq)
{
293     // Configure a clock to run from a GPIO input
294     uint gpin = 0;
295     if (gpio == 20) gpin = 0;
296     else if (gpio == 22) gpin = 1;
297 #if PICO_RP2350
298     else if (gpio == 12) gpin = 0;
299     else if (gpio == 14) gpin = 1;
300 #endif
301     else {
302         invalid_params_if(HARDWARE_CLOCKS, true);
303     }
304
305     // Work out sources. GPIN is always an auxsrc
306     uint src = 0;
307
308     // GPIN1 == GPIN0 + 1
309     uint auxsrc = gpin0_src[clock] + gpin;
310
311     if (has_glitchless_mux(clock)) {
312         // AUX src is always 1
313         src = 1;
314     }
315 }
```

```

316    // Set the GPIO function
317    gpio_set_function(gpio, GPIO_FUNC_GPCK);
318
319    // Now we have the src, auxsrc, and configured the gpio input
320    // call clock configure to run the clock from a gpio
321    return clock_configure(clock, src, auxsrc, src_freq, freq);
322 }

```

8.1.5.5. Enabling resus

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_clocks/clocks.c Lines 207 - 229

```

207 void clocks_enable_resus(resus_callback_t resus_callback) {
208     // Restart clk_sys if it is stopped by forcing it
209     // to the default source of clk_ref. If clk_ref stops running this will
210     // not work.
211
212     // Store user's resus callback
213     _resus_callback = resus_callback;
214
215     irq_set_exclusive_handler(CLOCKS_IRQ, clocks_irq_handler);
216
217     // Enable the resus interrupt in clocks
218     clocks_hw->inte = CLOCKS_INTE_CLK_SYS_RESUS_BITS;
219
220     // Enable the clocks irq
221     irq_set_enabled(CLOCKS_IRQ, true);
222
223     // 2 * clk_ref freq / clk_sys_min_freq;
224     // assume clk_ref is 3MHz and we want clk_sys to be no lower than 1MHz
225     uint timeout = 2 * 3 * 1;
226
227     // Enable resus with the maximum timeout
228     clocks_hw->resus.ctrl = CLOCKS_CLK_SYS_RESUS_CTRL_ENABLE_BITS | timeout;
229 }

```

8.1.5.6. Configuring sleep mode

Sleep mode is active when neither processor core nor the DMA are requesting clocks. For example, sleep mode is active when the DMA is not active and both core0 and core1 are waiting for an interrupt.

The `SLEEP_EN` registers set what clocks run in sleep mode. The `hello_sleep` example (https://github.com/raspberrypi/pico-playground/blob/master/sleep/hello_sleep/hello_sleep.c) illustrates how to put the chip to sleep until the AON-Timer fires.

 **NOTE**

`clk_sys` is always sent to `proc0` and `proc1` during sleep mode, as some logic must be clocked for the processor to wake up again.

8.1.6. List of Registers

The clocks registers start at a base address of `0x40010000` (defined as `CLOCKS_BASE` in SDK).

Table 543. List of
CLOCKS registers

Offset	Name	Info
0x00	CLK_GPOUT0_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x04	CLK_GPOUT0_DIV	
0x08	CLK_GPOUT0_SELECTED	Indicates which src is currently selected (one-hot)
0x0c	CLK_GPOUT1_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x10	CLK_GPOUT1_DIV	
0x14	CLK_GPOUT1_SELECTED	Indicates which src is currently selected (one-hot)
0x18	CLK_GPOUT2_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x1c	CLK_GPOUT2_DIV	
0x20	CLK_GPOUT2_SELECTED	Indicates which src is currently selected (one-hot)
0x24	CLK_GPOUT3_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x28	CLK_GPOUT3_DIV	
0x2c	CLK_GPOUT3_SELECTED	Indicates which src is currently selected (one-hot)
0x30	CLK_REF_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x34	CLK_REF_DIV	
0x38	CLK_REF_SELECTED	Indicates which src is currently selected (one-hot)
0x3c	CLK_SYS_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x40	CLK_SYS_DIV	
0x44	CLK_SYS_SELECTED	Indicates which src is currently selected (one-hot)
0x48	CLK_PERI_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x4c	CLK_PERI_DIV	
0x50	CLK_PERI_SELECTED	Indicates which src is currently selected (one-hot)
0x54	CLK_HSTX_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x58	CLK_HSTX_DIV	
0x5c	CLK_HSTX_SELECTED	Indicates which src is currently selected (one-hot)
0x60	CLK_USB_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x64	CLK_USB_DIV	
0x68	CLK_USB_SELECTED	Indicates which src is currently selected (one-hot)
0x6c	CLK_ADC_CTRL	Clock control, can be changed on-the-fly (except for auxsrc)
0x70	CLK_ADC_DIV	
0x74	CLK_ADC_SELECTED	Indicates which src is currently selected (one-hot)
0x78	DFTCLK_XOSC_CTRL	
0x7c	DFTCLK_ROSC_CTRL	
0x80	DFTCLK_LPOSC_CTRL	
0x84	CLK_SYS_RESUS_CTRL	
0x88	CLK_SYS_RESUS_STATUS	
0x8c	FC0_REF_KHZ	Reference clock frequency in kHz

Offset	Name	Info
0x90	FC0_MIN_KHZ	Minimum pass frequency in kHz. This is optional. Set to 0 if you are not using the pass/fail flags
0x94	FC0_MAX_KHZ	Maximum pass frequency in kHz. This is optional. Set to 0x1fffff if you are not using the pass/fail flags
0x98	FC0_DELAY	Delays the start of frequency counting to allow the mux to settle Delay is measured in multiples of the reference clock period
0x9c	FC0_INTERVAL	The test interval is 0.98us * 2**interval, but let's call it 1us * 2**interval The default gives a test interval of 250us
0xa0	FC0_SRC	Clock sent to frequency counter, set to 0 when not required Writing to this register initiates the frequency count
0xa4	FC0_STATUS	Frequency counter status
0xa8	FC0_RESULT	Result of frequency measurement, only valid when status_done=1
0xac	WAKE_EN0	enable clock in wake mode
0xb0	WAKE_EN1	enable clock in wake mode
0xb4	SLEEP_EN0	enable clock in sleep mode
0xb8	SLEEP_EN1	enable clock in sleep mode
0xbc	ENABLED0	indicates the state of the clock enable
0xc0	ENABLED1	indicates the state of the clock enable
0xc4	INTR	Raw Interrupts
0xc8	INTE	Interrupt Enable
0xcc	INTF	Interrupt Force
0xd0	INTS	Interrupt status after masking & forcing

CLOCKS: CLK_GPOUT0_CTRL Register

Offset: 0x00

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 544.
CLK_GPOUT0_CTRL
Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:21	Reserved.	-	-	-
20	NUDGE	An edge on this signal shifts the phase of the output by 1 cycle of the input clock This can be done at any time	RW	0x0
19:18	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
17:16	PHASE	This delays the enable signal by up to 3 cycles of the input clock This must be set before the clock is enabled to have any effect	RW	0x0
15:13	Reserved.	-	-	-
12	DC50	Enables duty cycle correction for odd divisors, can be changed on-the-fly	RW	0x0
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9	Reserved.	-	-	-
8:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_sys 0x1 → clksrc_gpin0 0x2 → clksrc_gpin1 0x3 → clksrc_pll_usb 0x4 → clksrc_pll_usb_primary_ref_opcg 0x5 → rosc_clksrc 0x6 → xosc_clksrc 0x7 → lposc_clksrc 0x8 → clk_sys 0x9 → clk_usb 0xa → clk_adc 0xb → clk_ref 0xc → clk_peri 0xd → clk_hstx 0xe → otp_clk2fc	RW	0x0
4:0	Reserved.	-	-	-

CLOCKS: CLK_GPOUT0_DIV Register

Offset: 0x04

Table 545.
CLK_GPOUT0_DIV
Register

Bits	Name	Description	Type	Reset
31:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x0001
15:0	FRAC	Fractional component of the divisor, can be changed on-the-fly	RW	0x0000

CLOCKS: CLK_GPOUT0_SELECTED Register

Offset: 0x08

Description

Indicates which src is currently selected (one-hot)

Table 546.
CLK_GPOUT0_SELECT
ED Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: CLK_GPOUT1_CTRL Register

Offset: 0x0c

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 547.
CLK_GPOUT1_CTRL
Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:21	Reserved.	-	-	-
20	NUDGE	An edge on this signal shifts the phase of the output by 1 cycle of the input clock This can be done at any time	RW	0x0
19:18	Reserved.	-	-	-
17:16	PHASE	This delays the enable signal by up to 3 cycles of the input clock This must be set before the clock is enabled to have any effect	RW	0x0
15:13	Reserved.	-	-	-
12	DC50	Enables duty cycle correction for odd divisors, can be changed on-the-fly	RW	0x0
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9	Reserved.	-	-	-
8:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_sys 0x1 → clksrc_gpin0 0x2 → clksrc_gpin1 0x3 → clksrc_pll_usb 0x4 → clksrc_pll_usb_primary_ref_opcg 0x5 → rosc_clksrc 0x6 → xosc_clksrc 0x7 → lposc_clksrc 0x8 → clk_sys 0x9 → clk_usb 0xa → clk_adc 0xb → clk_ref 0xc → clk_peri 0xd → clk_hstx 0xe → otp_clk2fc	RW	0x0

Bits	Name	Description	Type	Reset
4:0	Reserved.	-	-	-

CLOCKS: CLK_GPOUT1_DIV Register

Offset: 0x10

Table 548.
CLK_GPOUT1_DIV
Register

Bits	Name	Description	Type	Reset
31:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x0001
15:0	FRAC	Fractional component of the divisor, can be changed on-the-fly	RW	0x0000

CLOCKS: CLK_GPOUT1_SELECTED Register

Offset: 0x14

Description

Indicates which src is currently selected (one-hot)

Table 549.
CLK_GPOUT1_SELECT
ED Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: CLK_GPOUT2_CTRL Register

Offset: 0x18

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 550.
CLK_GPOUT2_CTRL
Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:21	Reserved.	-	-	-
20	NUDGE	An edge on this signal shifts the phase of the output by 1 cycle of the input clock This can be done at any time	RW	0x0
19:18	Reserved.	-	-	-
17:16	PHASE	This delays the enable signal by up to 3 cycles of the input clock This must be set before the clock is enabled to have any effect	RW	0x0
15:13	Reserved.	-	-	-
12	DC50	Enables duty cycle correction for odd divisors, can be changed on-the-fly	RW	0x0
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0

Bits	Name	Description	Type	Reset
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9	Reserved.	-	-	-
8:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_sys 0x1 → clksrc_gpin0 0x2 → clksrc_gpin1 0x3 → clksrc_pll_usb 0x4 → clksrc_pll_usb_primary_ref_opcg 0x5 → rosc_clksrc_ph 0x6 → xosc_clksrc 0x7 → lposc_clksrc 0x8 → clk_sys 0x9 → clk_usb 0xa → clk_adc 0xb → clk_ref 0xc → clk_peri 0xd → clk_hstx 0xe → otp_clk2fc	RW	0x0
4:0	Reserved.	-	-	-

CLOCKS: CLK_GPOUT2_DIV Register

Offset: 0x1c

Table 551.
CLK_GPOUT2_DIV
Register

Bits	Name	Description	Type	Reset
31:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x0001
15:0	FRAC	Fractional component of the divisor, can be changed on-the-fly	RW	0x0000

CLOCKS: CLK_GPOUT2_SELECTED Register

Offset: 0x20

Description

Indicates which src is currently selected (one-hot)

Table 552.
CLK_GPOUT2_SELECT
ED Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: CLK_GPOUT3_CTRL Register

Offset: 0x24

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 553.
CLK_GPOUT3_CTRL
Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:21	Reserved.	-	-	-
20	NUDGE	An edge on this signal shifts the phase of the output by 1 cycle of the input clock This can be done at any time	RW	0x0
19:18	Reserved.	-	-	-
17:16	PHASE	This delays the enable signal by up to 3 cycles of the input clock This must be set before the clock is enabled to have any effect	RW	0x0
15:13	Reserved.	-	-	-
12	DC50	Enables duty cycle correction for odd divisors, can be changed on-the-fly	RW	0x0
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9	Reserved.	-	-	-
8:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_sys 0x1 → clksrc_gpin0 0x2 → clksrc_gpin1 0x3 → clksrc_pll_usb 0x4 → clksrc_pll_usb_primary_ref_opcg 0x5 → rosc_clksrc_ph 0x6 → xosc_clksrc 0x7 → lposc_clksrc 0x8 → clk_sys 0x9 → clk_usb 0xa → clk_adc 0xb → clk_ref 0xc → clk_peri 0xd → clk_hstx 0xe → otp_clk2fc	RW	0x0
4:0	Reserved.	-	-	-

CLOCKS: CLK_GPOUT3_DIV Register

Offset: 0x28

Table 554.
CLK_GPOUT3_DIV
Register

Bits	Name	Description	Type	Reset
31:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x0001
15:0	FRAC	Fractional component of the divisor, can be changed on-the-fly	RW	0x0000

CLOCKS: CLK_GPOUT3_SELECTED Register

Offset: 0x2c

Description

Indicates which src is currently selected (one-hot)

Table 555.
CLK_GPOUT3_SELECT
ED Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: CLK_REF_CTRL Register

Offset: 0x30

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 556.
CLK_REF_CTRL
Register

Bits	Name	Description	Type	Reset
31:7	Reserved.	-	-	-
6:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_usb 0x1 → clksrc_gpin0 0x2 → clksrc_gpin1 0x3 → clksrc_pll_usb_primary_ref_opcg	RW	0x0
4:2	Reserved.	-	-	-
1:0	SRC	Selects the clock source glitchlessly, can be changed on-the-fly 0x0 → rosc_clksrc_ph 0x1 → clksrc_clk_ref_aux 0x2 → xosc_clksrc 0x3 → lposc_clksrc	RW	-

CLOCKS: CLK_REF_DIV Register

Offset: 0x34

Table 557.
CLK_REF_DIV Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x01
15:0	Reserved.	-	-	-

CLOCKS: CLK_REF_SELECTED Register

Offset: 0x38

Description

Indicates which src is currently selected (one-hot)

Table 558.
CLK_REF_SELECTED
Register

Bits	Description	Type	Reset
31:4	Reserved.	-	-
3:0	The glitchless multiplexer does not switch instantaneously (to avoid glitches), so software should poll this register to wait for the switch to complete. This register contains one decoded bit for each of the clock sources enumerated in the CTRL SRC field. At most one of these bits will be set at any time, indicating that clock is currently present at the output of the glitchless mux. Whilst switching is in progress, this register may briefly show all-0s.	RO	0x1

CLOCKS: CLK_SYS_CTRL Register

Offset: 0x3c

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 559.
CLK_SYS_CTRL
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_sys 0x1 → clksrc_pll_usb 0x2 → rosc_clksrc 0x3 → xosc_clksrc 0x4 → clksrc_gpin0 0x5 → clksrc_gpin1	RW	0x0
4:1	Reserved.	-	-	-
0	SRC	Selects the clock source glitchlessly, can be changed on-the-fly 0x0 → clk_ref 0x1 → clksrc_clk_sys_aux	RW	0x0

CLOCKS: CLK_SYS_DIV Register

Offset: 0x40

Table 560.
CLK_SYS_DIV Register

Bits	Name	Description	Type	Reset
31:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x0001
15:0	FRAC	Fractional component of the divisor, can be changed on-the-fly	RW	0x0000

CLOCKS: CLK_SYS_SELECTED Register

Offset: 0x44

Description

Indicates which src is currently selected (one-hot)

Table 561.
CLK_SYS_SELECTED
Register

Bits	Description	Type	Reset
31:2	Reserved.	-	-
1:0	The glitchless multiplexer does not switch instantaneously (to avoid glitches), so software should poll this register to wait for the switch to complete. This register contains one decoded bit for each of the clock sources enumerated in the CTRL SRC field. At most one of these bits will be set at any time, indicating that clock is currently present at the output of the glitchless mux. Whilst switching is in progress, this register may briefly show all-0s.	RO	0x1

CLOCKS: CLK_PERI_CTRL Register

Offset: 0x48

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 562.
CLK_PERI_CTRL
Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:12	Reserved.	-	-	-
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9:8	Reserved.	-	-	-
7:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clk_sys 0x1 → clksrc_pll_sys 0x2 → clksrc_pll_usb 0x3 → rosc_clksrc_ph 0x4 → xosc_clksrc 0x5 → clksrc_gpin0 0x6 → clksrc_gpin1	RW	0x0
4:0	Reserved.	-	-	-

CLOCKS: CLK_PERI_DIV Register

Offset: 0x4c

Table 563.
CLK_PERI_DIV
Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x1
15:0	Reserved.	-	-	-

CLOCKS: CLK_PERI_SELECTED Register

Offset: 0x50

Description

Indicates which src is currently selected (one-hot)

Table 564.
CLK_PERI_SELECTED
Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: CLK_HSTX_CTRL Register

Offset: 0x54

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 565.
CLK_HSTX_CTRL
Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:21	Reserved.	-	-	-
20	NUDGE	An edge on this signal shifts the phase of the output by 1 cycle of the input clock This can be done at any time	RW	0x0
19:18	Reserved.	-	-	-
17:16	PHASE	This delays the enable signal by up to 3 cycles of the input clock This must be set before the clock is enabled to have any effect	RW	0x0
15:12	Reserved.	-	-	-
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9:8	Reserved.	-	-	-
7:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clk_sys 0x1 → clksrc_pll_sys 0x2 → clksrc_pll_usb 0x3 → clksrc_gpin0 0x4 → clksrc_gpin1	RW	0x0
4:0	Reserved.	-	-	-

CLOCKS: CLK_HSTX_DIV Register

Offset: 0x58

Table 566.
CLK_HSTX_DIV
Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x1
15:0	Reserved.	-	-	-

CLOCKS: CLK_HSTX_SELECTED Register

Offset: 0x5c

Description

Indicates which src is currently selected (one-hot)

Table 567.
CLK_HSTX_SELECTED
Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: CLK_USB_CTRL Register

Offset: 0x60

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 568.
CLK_USB_CTRL
Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:21	Reserved.	-	-	-
20	NUDGE	An edge on this signal shifts the phase of the output by 1 cycle of the input clock This can be done at any time	RW	0x0
19:18	Reserved.	-	-	-
17:16	PHASE	This delays the enable signal by up to 3 cycles of the input clock This must be set before the clock is enabled to have any effect	RW	0x0
15:12	Reserved.	-	-	-
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_usb 0x1 → clksrc_pll_sys 0x2 → rosc_clksrc_ph 0x3 → xosc_clksrc 0x4 → clksrc_gpin0 0x5 → clksrc_gpin1	RW	0x0
4:0	Reserved.	-	-	-

CLOCKS: CLK_USB_DIV Register

Offset: 0x64

Table 569.
CLK_USB_DIV Register

Bits	Name	Description	Type	Reset
31:20	Reserved.	-	-	-
19:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x1
15:0	Reserved.	-	-	-

CLOCKS: CLK_USB_SELECTED Register

Offset: 0x68

Description

Indicates which src is currently selected (one-hot)

Table 570.
CLK_USB_SELECTED Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: CLK_ADC_CTRL Register

Offset: 0x6c

Description

Clock control, can be changed on-the-fly (except for auxsrc)

Table 571.
CLK_ADC_CTRL Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	ENABLED	clock generator is enabled	RO	0x0
27:21	Reserved.	-	-	-
20	NUDGE	An edge on this signal shifts the phase of the output by 1 cycle of the input clock This can be done at any time	RW	0x0
19:18	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
17:16	PHASE	This delays the enable signal by up to 3 cycles of the input clock This must be set before the clock is enabled to have any effect	RW	0x0
15:12	Reserved.	-	-	-
11	ENABLE	Starts and stops the clock generator cleanly	RW	0x0
10	KILL	Asynchronously kills the clock generator, enable must be set low before deasserting kill	RW	0x0
9:8	Reserved.	-	-	-
7:5	AUXSRC	Selects the auxiliary clock source, will glitch when switching 0x0 → clksrc_pll_usb 0x1 → clksrc_pll_sys 0x2 → rosc_clksrc_ph 0x3 → xosc_clksrc 0x4 → clksrc_gpin0 0x5 → clksrc_gpin1	RW	0x0
4:0	Reserved.	-	-	-

CLOCKS: CLK_ADC_DIV Register

Offset: 0x70

Table 572.
CLK_ADC_DIV Register

Bits	Name	Description	Type	Reset
31:20	Reserved.	-	-	-
19:16	INT	Integer part of clock divisor, 0 → max+1, can be changed on-the-fly	RW	0x1
15:0	Reserved.	-	-	-

CLOCKS: CLK_ADC_SELECTED Register

Offset: 0x74

Description

Indicates which src is currently selected (one-hot)

Table 573.
CLK_ADC_SELECTED Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This slice does not have a glitchless mux (only the AUX_SRC field is present, not SRC) so this register is hardwired to 0x1.	RO	0x1

CLOCKS: DFTCLK_XOSC_CTRL Register

Offset: 0x78

Table 574.
DFTCLK_XOSC_CTRL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
1:0	SRC	0x0 → NULL 0x1 → clksrc_pll_u sb_primary 0x2 → clksrc_gpin 0	RW	0x0

CLOCKS: DFTCLK_ROSC_CTRL Register

Offset: 0x7c

Table 575.
DFTCLK_ROSC_CTRL
Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1:0	SRC	0x0 → NULL 0x1 → clksrc_pll_s ys_primary_ rosc 0x2 → clksrc_gpin 1	RW	0x0

CLOCKS: DFTCLK_LPOSC_CTRL Register

Offset: 0x80

Table 576.
DFTCLK_LPOSC_CTRL
Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1:0	SRC	0x0 → NULL 0x1 → clksrc_pll_u sb_primary_ _lposc 0x2 → clksrc_gpin 1	RW	0x0

CLOCKS: CLK_SYS_RESUS_CTRL Register

Offset: 0x84

Table 577.
CLK_SYS_RESUS_CTRL
Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	CLEAR	For clearing the resus after the fault that triggered it has been corrected	RW	0x0
15:13	Reserved.	-	-	-
12	FRCE	Force a resus, for test purposes only	RW	0x0

Bits	Name	Description	Type	Reset
11:9	Reserved.	-	-	-
8	ENABLE	Enable resus	RW	0x0
7:0	TIMEOUT	This is expressed as a number of clk_ref cycles and must be $\geq 2 \times \text{clk_ref_freq}/\text{min_clk_tst_freq}$	RW	0xff

CLOCKS: CLK_SYS_RESUS_STATUS Register

Offset: 0x88

Table 578.
CLK_SYS_RESUS_STA
TUS Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	RESUSSED	Clock has been resuscitated, correct the error then send ctrl_clear=1	RO	0x0

CLOCKS: FC0_REF_KHZ Register

Offset: 0x8c

Table 579.
FC0_REF_KHZ Register

Bits	Description	Type	Reset
31:20	Reserved.	-	-
19:0	Reference clock frequency in kHz	RW	0x00000

CLOCKS: FC0_MIN_KHZ Register

Offset: 0x90

Table 580.
FC0_MIN_KHZ
Register

Bits	Description	Type	Reset
31:25	Reserved.	-	-
24:0	Minimum pass frequency in kHz. This is optional. Set to 0 if you are not using the pass/fail flags	RW	0x0000000

CLOCKS: FC0_MAX_KHZ Register

Offset: 0x94

Table 581.
FC0_MAX_KHZ
Register

Bits	Description	Type	Reset
31:25	Reserved.	-	-
24:0	Maximum pass frequency in kHz. This is optional. Set to 0xffffffff if you are not using the pass/fail flags	RW	0xffffffff

CLOCKS: FC0_DELAY Register

Offset: 0x98

Table 582. FC0_DELAY Register

Bits	Description	Type	Reset
31:3	Reserved.	-	-
2:0	Delays the start of frequency counting to allow the mux to settle Delay is measured in multiples of the reference clock period	RW	0x1

CLOCKS: FC0_INTERVAL Register

Offset: 0x9c

Table 583.
FC0_INTERVAL Register

Bits	Description	Type	Reset
31:4	Reserved.	-	-
3:0	The test interval is 0.98us * 2**interval, but let's call it 1us * 2**interval The default gives a test interval of 250us	RW	0x8

CLOCKS: FC0_SRC Register

Offset: 0xa0

Table 584. FC0_SRC Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	Clock sent to frequency counter, set to 0 when not required Writing to this register initiates the frequency count 0x00 → NULL 0x01 → pll_sys_clksrc_primary 0x02 → pll_usb_clksrc_primary 0x03 → rosc_clksrc 0x04 → rosc_clksrc_ph 0x05 → xosc_clksrc 0x06 → clksrc_gpin0 0x07 → clksrc_gpin1 0x08 → clk_ref 0x09 → clk_sys 0x0a → clk_peri 0x0b → clk_usb 0x0c → clk_adc 0x0d → clk_hstx 0x0e → lposc_clksrc 0x0f → otp_clk2fc 0x10 → pll_usb_clksrc_primary_dft	RW	0x00

CLOCKS: FC0_STATUS Register

Offset: 0xa4

Description

Frequency counter status

Table 585.
FC0_STATUS Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	DIED	Test clock stopped during test	RO	0x0
27:25	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
24	FAST	Test clock faster than expected, only valid when status_done=1	RO	0x0
23:21	Reserved.	-	-	-
20	SLOW	Test clock slower than expected, only valid when status_done=1	RO	0x0
19:17	Reserved.	-	-	-
16	FAIL	Test failed	RO	0x0
15:13	Reserved.	-	-	-
12	WAITING	Waiting for test clock to start	RO	0x0
11:9	Reserved.	-	-	-
8	RUNNING	Test running	RO	0x0
7:5	Reserved.	-	-	-
4	DONE	Test complete	RO	0x0
3:1	Reserved.	-	-	-
0	PASS	Test passed	RO	0x0

CLOCKS: FC0_RESULT Register

Offset: 0xa8

Description

Result of frequency measurement, only valid when status_done=1

Table 586.
FC0_RESULT Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:5	KHZ		RO	0x00000000
4:0	FRAC		RO	0x00

CLOCKS: WAKE_EN0 Register

Offset: 0xac

Description

enable clock in wake mode

Table 587. WAKE_EN0 Register

Bits	Name	Description	Type	Reset
31	CLK_SYS_SIO		RW	0x1
30	CLK_SYS_SHA256		RW	0x1
29	CLK_SYS_PSM		RW	0x1
28	CLK_SYS_ROSC		RW	0x1
27	CLK_SYS_ROM		RW	0x1
26	CLK_SYS_RESETS		RW	0x1
25	CLK_SYS_PWM		RW	0x1

Bits	Name	Description	Type	Reset
24	CLK_SYS_POWMAN		RW	0x1
23	CLK_REF_POWMAN		RW	0x1
22	CLK_SYS_PLL_USB		RW	0x1
21	CLK_SYS_PLL_SYS		RW	0x1
20	CLK_SYS_PIO2		RW	0x1
19	CLK_SYS_PIO1		RW	0x1
18	CLK_SYS_PIO0		RW	0x1
17	CLK_SYS_PADS		RW	0x1
16	CLK_SYS OTP		RW	0x1
15	CLK_REF OTP		RW	0x1
14	CLK_SYS_JTAG		RW	0x1
13	CLK_SYS_IO		RW	0x1
12	CLK_SYS_I2C1		RW	0x1
11	CLK_SYS_I2C0		RW	0x1
10	CLK_SYS_HSTX		RW	0x1
9	CLK_HSTX		RW	0x1
8	CLK_SYS_GLITCH_DETECTOR		RW	0x1
7	CLK_SYS_DMA		RW	0x1
6	CLK_SYS_BUSFABRIC		RW	0x1
5	CLK_SYS_BUSCTRL		RW	0x1
4	CLK_SYS_BOOTRAM		RW	0x1
3	CLK_SYS_ADC		RW	0x1
2	CLK_ADC_ADC		RW	0x1
1	CLK_SYS_ACCESSCTRL		RW	0x1
0	CLK_SYS_CLOCKS		RW	0x1

CLOCKS: WAKE_EN1 Register

Offset: 0xb0

Description

enable clock in wake mode

Table 588. WAKE_EN1 Register

Bits	Name	Description	Type	Reset
31	Reserved.	-	-	-
30	CLK_SYS_XOSC		RW	0x1
29	CLK_SYS_XIP		RW	0x1
28	CLK_SYS_WATCHDOG		RW	0x1
27	CLK_USB		RW	0x1

Bits	Name	Description	Type	Reset
26	CLK_SYS_USBCTRL		RW	0x1
25	CLK_SYS_UART1		RW	0x1
24	CLK_PERI_UART1		RW	0x1
23	CLK_SYS_UART0		RW	0x1
22	CLK_PERI_UART0		RW	0x1
21	CLK_SYS_TRNG		RW	0x1
20	CLK_SYS_TIMER1		RW	0x1
19	CLK_SYS_TIMER0		RW	0x1
18	CLK_SYS_TICKS		RW	0x1
17	CLK_REF_TICKS		RW	0x1
16	CLK_SYS_TRMAN		RW	0x1
15	CLK_SYS_SYSINFO		RW	0x1
14	CLK_SYS_SYSFCFG		RW	0x1
13	CLK_SYS_SRAM9		RW	0x1
12	CLK_SYS_SRAM8		RW	0x1
11	CLK_SYS_SRAM7		RW	0x1
10	CLK_SYS_SRAM6		RW	0x1
9	CLK_SYS_SRAM5		RW	0x1
8	CLK_SYS_SRAM4		RW	0x1
7	CLK_SYS_SRAM3		RW	0x1
6	CLK_SYS_SRAM2		RW	0x1
5	CLK_SYS_SRAM1		RW	0x1
4	CLK_SYS_SRAM0		RW	0x1
3	CLK_SYS_SPI1		RW	0x1
2	CLK_PERI_SPI1		RW	0x1
1	CLK_SYS_SPI0		RW	0x1
0	CLK_PERI_SPI0		RW	0x1

CLOCKS: SLEEP_EN0 Register

Offset: 0xb4

Description

enable clock in sleep mode

Table 589. SLEEP_EN0 Register

Bits	Name	Description	Type	Reset
31	CLK_SYS_SIO		RW	0x1
30	CLK_SYS_SHA256		RW	0x1
29	CLK_SYS_PSM		RW	0x1

Bits	Name	Description	Type	Reset
28	CLK_SYS_ROSC		RW	0x1
27	CLK_SYS_ROM		RW	0x1
26	CLK_SYS_RESETS		RW	0x1
25	CLK_SYS_PWM		RW	0x1
24	CLK_SYS_POWMAN		RW	0x1
23	CLK_REF_POWMAN		RW	0x1
22	CLK_SYS_PLL_USB		RW	0x1
21	CLK_SYS_PLL_SYS		RW	0x1
20	CLK_SYS_PIO2		RW	0x1
19	CLK_SYS_PIO1		RW	0x1
18	CLK_SYS_PIO0		RW	0x1
17	CLK_SYS_PADS		RW	0x1
16	CLK_SYS OTP		RW	0x1
15	CLK_REF OTP		RW	0x1
14	CLK_SYS_JTAG		RW	0x1
13	CLK_SYS_IO		RW	0x1
12	CLK_SYS_I2C1		RW	0x1
11	CLK_SYS_I2C0		RW	0x1
10	CLK_SYS_HSTX		RW	0x1
9	CLK_HSTX		RW	0x1
8	CLK_SYS_GLITCH_DETECTOR		RW	0x1
7	CLK_SYS_DMA		RW	0x1
6	CLK_SYS_BUSFABRIC		RW	0x1
5	CLK_SYS_BUSCTRL		RW	0x1
4	CLK_SYS_BOOTRAM		RW	0x1
3	CLK_SYS_ADC		RW	0x1
2	CLK_ADC_ADC		RW	0x1
1	CLK_SYS_ACCESSCTRL		RW	0x1
0	CLK_SYS_CLOCKS		RW	0x1

CLOCKS: SLEEP_EN1 Register

Offset: 0xb8

Description

enable clock in sleep mode

Table 590. SLEEP_EN1 Register

Bits	Name	Description	Type	Reset
31	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
30	CLK_SYS_XOSC		RW	0x1
29	CLK_SYS_XIP		RW	0x1
28	CLK_SYS_WATCHDOG		RW	0x1
27	CLK_USB		RW	0x1
26	CLK_SYS_USBCTRL		RW	0x1
25	CLK_SYS_UART1		RW	0x1
24	CLK_PERI_UART1		RW	0x1
23	CLK_SYS_UART0		RW	0x1
22	CLK_PERI_UART0		RW	0x1
21	CLK_SYS_TRNG		RW	0x1
20	CLK_SYS_TIMER1		RW	0x1
19	CLK_SYS_TIMER0		RW	0x1
18	CLK_SYS_TICKS		RW	0x1
17	CLK_REF_TICKS		RW	0x1
16	CLK_SYS_TBMAN		RW	0x1
15	CLK_SYS_SYSINFO		RW	0x1
14	CLK_SYS_SYSCFG		RW	0x1
13	CLK_SYS_SRAM9		RW	0x1
12	CLK_SYS_SRAM8		RW	0x1
11	CLK_SYS_SRAM7		RW	0x1
10	CLK_SYS_SRAM6		RW	0x1
9	CLK_SYS_SRAM5		RW	0x1
8	CLK_SYS_SRAM4		RW	0x1
7	CLK_SYS_SRAM3		RW	0x1
6	CLK_SYS_SRAM2		RW	0x1
5	CLK_SYS_SRAM1		RW	0x1
4	CLK_SYS_SRAM0		RW	0x1
3	CLK_SYS_SPI1		RW	0x1
2	CLK_PERI_SPI1		RW	0x1
1	CLK_SYS_SPI0		RW	0x1
0	CLK_PERI_SPI0		RW	0x1

CLOCKS: ENABLED0 Register

Offset: 0xbc

Description

indicates the state of the clock enable

Table 591. ENABLED0 Register

Bits	Name	Description	Type	Reset
31	CLK_SYS_SIO		RO	0x0
30	CLK_SYS_SHA256		RO	0x0
29	CLK_SYS_PSM		RO	0x0
28	CLK_SYS_ROSC		RO	0x0
27	CLK_SYS_ROM		RO	0x0
26	CLK_SYS_RESETS		RO	0x0
25	CLK_SYS_PWM		RO	0x0
24	CLK_SYS_POWMAN		RO	0x0
23	CLK_REF_POWMAN		RO	0x0
22	CLK_SYS_PLL_USB		RO	0x0
21	CLK_SYS_PLL_SYS		RO	0x0
20	CLK_SYS_PIO2		RO	0x0
19	CLK_SYS_PIO1		RO	0x0
18	CLK_SYS_PIO0		RO	0x0
17	CLK_SYS_PADS		RO	0x0
16	CLK_SYS OTP		RO	0x0
15	CLK_REF OTP		RO	0x0
14	CLK_SYS_JTAG		RO	0x0
13	CLK_SYS_IO		RO	0x0
12	CLK_SYS_I2C1		RO	0x0
11	CLK_SYS_I2C0		RO	0x0
10	CLK_SYS_HSTX		RO	0x0
9	CLK_HSTX		RO	0x0
8	CLK_SYS_GLITCH_DETECTOR		RO	0x0
7	CLK_SYS_DMA		RO	0x0
6	CLK_SYS_BUSFABRIC		RO	0x0
5	CLK_SYS_BUSCTRL		RO	0x0
4	CLK_SYS_BOOTRAM		RO	0x0
3	CLK_SYS_ADC		RO	0x0
2	CLK_ADC_ADC		RO	0x0
1	CLK_SYS_ACCESSCTRL		RO	0x0
0	CLK_SYS_CLOCKS		RO	0x0

CLOCKS: ENABLED1 Register

Offset: 0xc0

Description

indicates the state of the clock enable

Table 592. ENABLED1 Register

Bits	Name	Description	Type	Reset
31	Reserved.	-	-	-
30	CLK_SYS_XOSC		RO	0x0
29	CLK_SYS_XIP		RO	0x0
28	CLK_SYS_WATCHDOG		RO	0x0
27	CLK_USB		RO	0x0
26	CLK_SYS_USBCTRL		RO	0x0
25	CLK_SYS_UART1		RO	0x0
24	CLK_PERI_UART1		RO	0x0
23	CLK_SYS_UART0		RO	0x0
22	CLK_PERI_UART0		RO	0x0
21	CLK_SYS_TRNG		RO	0x0
20	CLK_SYS_TIMER1		RO	0x0
19	CLK_SYS_TIMER0		RO	0x0
18	CLK_SYS_TICKS		RO	0x0
17	CLK_REF_TICKS		RO	0x0
16	CLK_SYS_TBMAN		RO	0x0
15	CLK_SYS_SYSINFO		RO	0x0
14	CLK_SYS_SYSFCFG		RO	0x0
13	CLK_SYS_SRAM9		RO	0x0
12	CLK_SYS_SRAM8		RO	0x0
11	CLK_SYS_SRAM7		RO	0x0
10	CLK_SYS_SRAM6		RO	0x0
9	CLK_SYS_SRAM5		RO	0x0
8	CLK_SYS_SRAM4		RO	0x0
7	CLK_SYS_SRAM3		RO	0x0
6	CLK_SYS_SRAM2		RO	0x0
5	CLK_SYS_SRAM1		RO	0x0
4	CLK_SYS_SRAM0		RO	0x0
3	CLK_SYS_SPI1		RO	0x0
2	CLK_PERI_SPI1		RO	0x0
1	CLK_SYS_SPI0		RO	0x0
0	CLK_PERI_SPI0		RO	0x0

CLOCKS: INTR Register

Offset: 0xc4

Description

Raw Interrupts

Table 593. INTR Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLK_SYS_RESUS		RO	0x0

CLOCKS: INTE Register

Offset: 0xc8

Description

Interrupt Enable

Table 594. INTE Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLK_SYS_RESUS		RW	0x0

CLOCKS: INTF Register

Offset: 0xcc

Description

Interrupt Force

Table 595. INTF Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLK_SYS_RESUS		RW	0x0

CLOCKS: INTS Register

Offset: 0xd0

Description

Interrupt status after masking & forcing

Table 596. INTS Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLK_SYS_RESUS		RO	0x0

8.2. Crystal Oscillator (XOSC)

8.2.1. Overview

The Crystal Oscillator (XOSC) uses an external crystal to produce an accurate reference clock. The RP2350 supports 1MHz to 100MHz crystals and the RP2350 reference design (see [Hardware design with RP2040, Minimal Design Example](#)) uses a 12MHz crystal. The reference clock is distributed to the PLLs, which can be used to multiply the XOSC frequency to provide accurate high speed clocks. For example, they can generate a 48MHz clock which meets the frequency accuracy requirement of the USB interface and a 150MHz maximum speed system clock. The XOSC clock is also a clock source for the clock generators and can be used directly if required.

If the user already has an accurate clock source, it is possible to drive an external clock directly into XIN (aka XI), and disable the oscillator circuit. In this mode XIN can be driven at up to 50MHz.

To use XOSC clock externally, output it to a GPIO pin using one of the `clk_gpclk0-clk_gpclk3` generators. You cannot take XOSC output directly from the XIN (XI) or XOUT (XO) pins.

ⓘ NOTE

A minimum crystal frequency of 5MHz is needed for the PLL. See [Section 8.6](#).

8.2.1.1. Recommended Crystals

For the best performance and stability across typical operating temperature ranges, it is recommended to use the Abracon ABM8-272-T3. You can source the ABM8-272-T3 directly from Abracon or from an authorised reseller. The Abracon ABM8-272-T3 has the following specifications:

Table 597. Key Crystal Specifications.

Parameters	Minimum	Typical	Maximum	Units	Notes
Center Frequency	12.000	12.000	12.000	MHz	
Operation Mode	Fundamental-AT	Fundamental-AT	Fundamental-AT		
Operating Temperature	-40		+85	°C	
Storage Temperature	-55		+125	°C	
Frequency Tolerance (25°C)	-30		+30	ppm	
Frequency Stability (25°C)	-30		+30	ppm	
Equivalent Series Resistance (R1)			50	Ω	
Shunt Capacitance (C0)			3.0	pF	
Load Capacitance (CL)	10	10	10	pF	
Drive Level		10	200	µW	
Aging	-5		+5	ppm	@25±3°C, 1st year
Insulation Resistance	500			MΩ	@100Vdc±15V

Even if you use a crystal with similar specifications, you will need to test the circuit over a range of temperatures to ensure stability.

The crystal oscillator is powered from the VDDIO voltage. As a result, the Abracon crystal and that particular damping resistor are tuned for 3.3V operation. If you use a different IO voltage, you will need to re-tune.

Any changes to crystal parameters risk instability across any components connected to the crystal circuit.

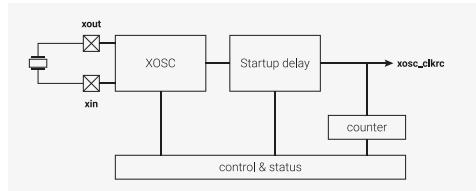
If you can't source the recommended crystal directly from Abracon or a reseller, contact applications@raspberrypi.com.

Raspberry Pi Pico 2 has been specifically tuned for the specifications of the Abracon ABM8-272-T3 crystal. For an example of how to use a crystal with RP2350, see the Raspberry Pi Pico 2 board schematic in <https://datasheets.raspberrypi.com/pico2/pico-2-datasheet.pdf#pico-schematic-diagram>, Appendix B of the Pico Datasheet>> and the [Raspberry Pi Pico 2 design files](#).

8.2.2. Changes from RP2040

- Maximum crystal frequency increased from 15MHz to 100MHz

Figure 35. XOSC overview



8.2.3. Usage

The XOSC is disabled on chip startup and the RP2350 boots using the Ring Oscillator (ROSC). To start the XOSC, the programmer must set the `CTRL_ENABLE` register. The XOSC is not immediately usable because it takes time for the oscillations to build to sufficient amplitude. This time will be dependent on the chosen crystal but will be of the order of a few milliseconds. The XOSC incorporates a timer controlled by the `STARTUP_DELAY` register to automatically manage this, which sets a flag (`STATUS_STABLE`) when the XOSC clock is usable.

8.2.4. Startup Delay

The `STARTUP_DELAY` register specifies how many clock cycles must be seen from the crystal before it can be used. This is specified in multiples of 256. The SDK `xosc_init` function sets this value. The 1ms default is sufficient for the RP2350 reference design (see [Hardware design with RP2040, Minimal Design Example](#)) which runs the XOSC at 12MHz. When the timer expires, the `STATUS_STABLE` flag will be set to indicate the XOSC output can be used.

Before starting the XOSC the programmer must ensure the `STARTUP_DELAY` register is correctly configured. The required value can be calculated by:

$$(f_{Crystal} \times t_{Stable}) \div 256$$

So with a 12MHz crystal and a 1ms wait time, the calculation is:

$$(12 \times 10^6 \cdot 1 \times 10^{-3}) \div 256 \approx 47$$

NOTE

The value is rounded up to the nearest integer, so the wait time will be just over 1ms.

8.2.5. XOSC Counter

The `COUNT` register provides a method of managing short software delays. To use this method:

1. Write a value to the `COUNT` register. The register automatically begins to count down to zero at the XOSC frequency.
2. Poll the register until it reaches zero.

This is preferable to using NOPs in software loops because it is independent of the core clock frequency, the compiler, and the execution time of the compiled code.

8.2.6. DORMANT mode

In DORMANT mode (see [Section 6.3.5](#)), all of the on-chip clocks can be paused to save power. This is particularly useful in battery-powered applications. RP2350 wakes from DORMANT mode by interrupt: either from an external event, such as an edge on a GPIO pin, or from the AON-Timer. This must be configured before entering DORMANT mode. To use the AON-Timer to trigger a wake from DORMANT mode, it must be clocked from an external source.

To enter DORMANT mode:

1. Switch all internal clocks to be driven from XOSC or ROSC and stop the PLLs.
2. Choose an oscillator (XOSC or ROSC). Write a specific 32-bit value to the **DORMANT** register of the chosen oscillator to stop it.

When exiting DORMANT mode, the chosen oscillator will restart. If you chose XOSC, the frequency will be more precise, but the restart will take more time due to startup delay (>1ms on the RP2350 reference design (see [Hardware design with RP2040, Minimal Design Example](#))). If you chose ROSC, the frequency will be less precise, but the start-up time is very short (approximately 1μs).

NOTE

You must stop the PLLs before entering DORMANT mode.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_xosc/xosc.c Lines 56 - 63

```
56 void xosc_dormant(void) {
57     // WARNING: This stops the xosc until woken up by an irq
58     xosc_hw->dormant = XOSC_DORMANT_VALUE_DORMANT;
59     // Wait for it to become stable once woken up
60     while(!(xosc_hw->status & XOSC_STATUS_STABLE_BITS)) {
61         tight_loop_contents();
62     }
63 }
```

WARNING

If you do not configure IRQ before entering DORMANT mode, neither oscillator will restart.

See [Section 6.3.7.2](#) for a complete example of DORMANT mode using the XOSC.

8.2.7. Programmer's Model

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2350/hardware_structs/include/hardware_structs/xosc.h Lines 24 - 55

```
24 typedef struct {
25     _REG_(XOSC_CTRL_OFFSET) // XOSC_CTRL
26     // Crystal Oscillator Control
27     // 0x00ffff000 [23:12] : ENABLE (0): On power-up this field is initialised to DISABLE and the chip runs from the ROSC
28     // 0x000000ff [11:0] : FREQ_RANGE (0): The 12-bit code is intended to give some protection against accidental writes
29     io_rw_32 ctrl;
30
31     _REG_(XOSC_STATUS_OFFSET) // XOSC_STATUS
32     // Crystal Oscillator Status
33     // 0x00000000 [31] : STABLE (0): Oscillator is running and stable
34     // 0x01000000 [24] : BADWRITE (0): An invalid value has been written to CTRL_ENABLE or CTRL_FREQ_RANGE or DORMANT
35     // 0x00001000 [12] : ENABLED (0): Oscillator is enabled but not necessarily running and stable, resets to 0
36     // 0x00000003 [1:0] : FREQ_RANGE (0): The current frequency range setting
37     io_rw_32 status;
38
39     _REG_(XOSC_DORMANT_OFFSET) // XOSC_DORMANT
40     // Crystal Oscillator pause control
41     io_rw_32 dormant;
42 }
```

```

43     _REG_(XOSC_STARTUP_OFFSET) // XOSC_STARTUP
44     // Controls the startup delay
45     // 0x00100000 [20] : X4 (0): Multiplies the startup_delay by 4, just in case
46     // 0x00003fff [13:0] : DELAY (0xc4): in multiples of 256*xtal_period
47     io_rw_32 startup;
48
49     _REG_(XOSC_COUNT_OFFSET) // XOSC_COUNT
50     // A down counter running at the xosc frequency which counts to zero and stops
51     // 0x0000ffff [15:0] : COUNT (0)
52     io_rw_32 count;
53 } xosc_hw_t;
54
55 #define xosc_hw ((xosc_hw_t *)XOSC_BASE)

```

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_xosc/xosc.c Lines 29 - 43

```

29 void xosc_init(void) {
30     // Assumes 1-15 MHz input, checked above.
31     xosc_hw->ctrl = XOSC_CTRL_FREQ_RANGE_VALUE_1_15MHZ;
32
33     // Set xosc startup delay
34     xosc_hw->startup = STARTUP_DELAY;
35
36     // Set the enable bit now that we have set freq range and startup delay
37     hw_set_bits(&xosc_hw->ctrl, XOSC_CTRL_ENABLE_VALUE_ENABLE << XOSC_CTRL_ENABLE_LSB);
38
39     // Wait for XOSC to be stable
40     while(!(xosc_hw->status & XOSC_STATUS_STABLE_BITS)) {
41         tight_loop_contents();
42     }
43 }

```

8.2.8. List of Registers

The XOSC registers start at a base address of **0x40048000** (defined as **XOSC_BASE** in SDK).

Table 598. List of XOSC registers

Offset	Name	Info
0x00	CTRL	Crystal Oscillator Control
0x04	STATUS	Crystal Oscillator Status
0x08	DORMANT	Crystal Oscillator pause control
0x0c	STARTUP	Controls the startup delay
0x10	COUNT	A down counter running at the XOSC frequency which counts to zero and stops.

XOSC: CTRL Register

Offset: 0x00

Description

Crystal Oscillator Control

Table 599. CTRL Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
23:12	ENABLE	<p>On power-up this field is initialised to DISABLE and the chip runs from the ROSC.</p> <p>If the chip has subsequently been programmed to run from the XOSC then setting this field to DISABLE may lock-up the chip. If this is a concern then run the clk_ref from the ROSC and enable the clk_sys RESUS feature.</p> <p>The 12-bit code is intended to give some protection against accidental writes. An invalid setting will retain the previous value. The actual value being used can be read from STATUS_ENABLED</p> <p>0xd1e → DISABLE</p> <p>0xfb → ENABLE</p>	RW	-
11:0	FREQ_RANGE	<p>The 12-bit code is intended to give some protection against accidental writes. An invalid setting will retain the previous value. The actual value being used can be read from STATUS_FREQ_RANGE</p> <p>0xaa0 → 1_15MHZ</p> <p>0xaa1 → 10_30MHZ</p> <p>0xaa2 → 25_60MHZ</p> <p>0xaa3 → 40_100MHZ</p>	RW	-

XOSC: STATUS Register

Offset: 0x04

Description

Crystal Oscillator Status

Table 600. STATUS Register

Bits	Name	Description	Type	Reset
31	STABLE	Oscillator is running and stable	RO	0x0
30:25	Reserved.	-	-	-
24	BADWRITE	An invalid value has been written to CTRL_ENABLE or CTRL_FREQ_RANGE or DORMANT	WC	0x0
23:13	Reserved.	-	-	-
12	ENABLED	Oscillator is enabled but not necessarily running and stable, resets to 0	RO	-
11:2	Reserved.	-	-	-
1:0	FREQ_RANGE	<p>The current frequency range setting</p> <p>0x0 → 1_15MHZ</p> <p>0x1 → 10_30MHZ</p> <p>0x2 → 25_60MHZ</p> <p>0x3 → 40_100MHZ</p>	RO	-

XOSC: DORMANT Register

Offset: 0x08

Description

Crystal Oscillator pause control

Table 601. DORMANT Register

Bits	Description	Type	Reset
31:0	This is used to save power by pausing the XOSC On power-up this field is initialised to WAKE An invalid write will also select WAKE WARNING: stop the PLLs before selecting dormant mode WARNING: setup the irq before selecting dormant mode 0x636f6d61 → DORMANT 0x77616b65 → WAKE	RW	-

XOSC: STARTUP Register

Offset: 0x0c

Description

Controls the startup delay

Table 602. STARTUP Register

Bits	Name	Description	Type	Reset
31:21	Reserved.	-	-	-
20	X4	Multiplies the startup_delay by 4, just in case. The reset value is controlled by a mask-programmable tiecell and is provided in case we are booting from XOSC and the default startup delay is insufficient	RW	0x0
19:14	Reserved.	-	-	-
13:0	DELAY	in multiples of 256*xtal_period. The reset value of 0xc4 corresponds to approx 50 000 cycles.	RW	0x00c4

XOSC: COUNT Register

Offset: 0x10

Table 603. COUNT Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	A down counter running at the xosc frequency which counts to zero and stops. Can be used for short software pauses when setting up time sensitive hardware. To start the counter, write a non-zero value. Reads will return 1 while the count is running and 0 when it has finished. Minimum count value is 4. Count values <4 will be treated as count value =4. Note that synchronisation to the register clock domain costs 2 register clock cycles and the counter cannot compensate for that.	RW	0x0000

8.3. Ring Oscillator (ROSC)

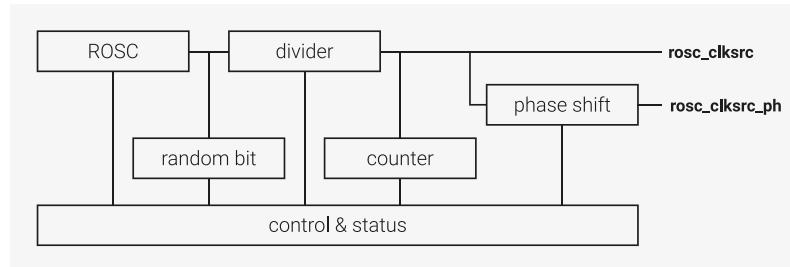
8.3.1. Overview

The Ring Oscillator (ROSC) is an on-chip oscillator built from a ring of inverters. It requires no external components and is started automatically during RP2350 power up. It provides the clock to the cores during boot. The frequency of the ROSC is programmable and it can directly provide a high speed clock to the cores, but the frequency varies with Process, Voltage, and Temperature (PVT) so it cannot provide clocks for components which require an accurate frequency such as the AON-Timer, USB, and ADC. Methods for mitigating the frequency variation are discussed in

[Section 8.1](#), but these are only relevant to very low power designs. For most applications requiring accurate clock frequencies, switch to the XOSC and PLLs. During boot, the ROSC runs at a nominal 11MHz and is guaranteed to be in the range 4.6MHz to 19.6MHz.

Once the chip has booted, the programmer can choose to continue running from the ROSC and increase its frequency or start the Crystal Oscillator (XOSC) and PLLs. You can disable the ROSC once you've switched the system clocks to the XOSC. Each oscillator has advantages; switch between them to achieve the best solution for your application.

Figure 36. ROSC overview.



8.3.2. ROSC/XOSC trade-offs

The ROSC has several advantages:

- flexibility due to programmable frequency
- low power requirements
- no need for internal or external components

Because the ROSC has programmable frequency, it can provide a fast core clock without starting the PLLs and can generate slower peripheral clocks by dividing by clock generators ([Section 8.1](#)). The ROSC starts immediately and responds immediately to frequency controls. It retains the frequency setting when entering and exiting the DORMANT state (see [Section 6.3.5](#)). However, the user must be aware that the frequency may have drifted when exiting the DORMANT state due to changes in the supply voltage and the chip temperature.

The disadvantage of the ROSC is its frequency variation with PVT (Process, Voltage, and Temperature) which makes it unsuitable for generating precise clocks or for applications where software execution timing is important. However, the PVT frequency variation can be exploited to provide automatic frequency scaling to maximise performance. This is discussed in [Section 8.1](#).

The only advantage of the XOSC is its accurate frequency, but this is an overriding requirement in many applications.

The XOSC has the following disadvantages:

- the requirement for external components (a crystal, etc.)
- higher power consumption
- slow startup time (>1ms)
- fixed, low frequency

PLLs are required to produce higher-frequency clocks. They consume more power and take significant time to start up or change frequency. Exiting DORMANT mode is much slower than for ROSC because the XOSC must restart and the PLLs must be reconfigured.

8.3.3. Modifying the frequency

The ROSC is arranged as 8 stages, each with programmable drive. The ROSC provides two methods of controlling the frequency. The frequency range controls the number of stages in the ROSC loop and the **FREQA** & **FREQB** registers control the drive strength of the stages.

To change the frequency range, write to the **FREQ_RANGE** register, which controls the number of stages in the ROSC loop.

The `FREQ_RANGE` register supports the following configurations:

Table 604. ROSC stage ranges

Name	Number of stages	Range (stages)
LOW	8	0-7
MEDIUM	6	2-7
HIGH	4	4-7
TOOHIGH	2	6-7

Change `FREQ_RANGE` one step at a time until you reach the desired range. When increasing the frequency range, ROSC output will not glitch, so the output clock can continue to be used. When decreasing the frequency range, ROSC output *will* glitch, so you must select an alternate clock source for the modules clocked by ROSC or hold them in reset during the transition.

The behaviour has not been fully characterised, but the `MEDIUM` range will be approximately 1.33 times the `LOW` range, the `HIGH` range will be 2 times the `LOW` range and the `TOOHIGH` range will be 4 times the `LOW` range. The `TOOHIGH` range is aptly named. It should not be used because the internal logic of the ROSC will not run at that frequency.

The `FREQA` and `FREQB` registers control the drive strength of the stages in the ROSC loop. As the drive strength increases, the delay through the stage decreases and the oscillation frequency increases. Each stage has 3 drive strength control bits. Each bit turns on an additional drive, therefore each stage has 4 drive strength settings equal to the number of bits set, with 0 being the default, 1 being double drive, 2 being triple drive and 3 being quadruple drive. Extra drives do not have a linear effect on frequency: the second has less impact than the first, the third has less impact than the second, and so on. To ensure smooth transitions, change one drive strength bit at a time. When `FREQ_RANGE` shortens the ROSC loop, the bypassed stages still propagate the signal and therefore their drive strengths must be set to at least the same level as the lowest drive strength in the stages that are in the loop. This will not affect the oscillation frequency.

8.3.4. ROSC divider

The ROSC frequency is too fast to be used directly, so it is divided in an integer divider controlled by the `DIV` register. You can change `DIV` while the ROSC is running, and the output clock will change frequency without glitching. The default divisor is 8, which ensures the output clock is in the specified range on chip startup.

The divider has two outputs, `rosclksrc` and `rosclksrc_ph`. `rosclksrc_ph` is a phase shifted version of `rosclksrc`. This is primarily intended for use during product development; the outputs are identical if the `PHASE` register is left in its default state.

8.3.5. Random Number Generator

When the system clocks are running from the XOSC, you can use the ROSC to generate random numbers. Enable the ROSC and read the `RANDOMBIT` register to get a 1-bit random number; to get an `n`-bit value, read it `n` times. This does not meet the requirements of randomness for security systems because it can be compromised, but it may be useful in less critical applications. If the cores are running from the ROSC, the value will not be random because the timing of the register read will be correlated to the phase of the ROSC.

8.3.6. ROSC Counter

The `COUNT` register provides a method of managing short software delays. To use this method:

1. Write a value to the `COUNT` register. The register automatically begins to count down to zero at the ROSC frequency.
2. Poll the register until it reaches zero.

This is preferable to using NOPs in software loops because it is independent of the core clock frequency, the compiler,

and the execution time of the compiled code.

8.3.7. DORMANT mode

In DORMANT mode (see [Section 6.3.5](#)), all of the on-chip clocks can be paused to save power. This is particularly useful in battery-powered applications. RP2350 wakes from DORMANT mode by interrupt: either from an external event, such as an edge on a GPIO pin, or from the AON-Timer. This must be configured before entering DORMANT mode. To use the AON-Timer to trigger a wake from DORMANT mode, it must be clocked from an external source.

To enter DORMANT mode:

1. Switch all internal clocks to be driven from XOSC or ROSC and stop the PLLs.
2. Choose an oscillator (XOSC or ROSC). Write a specific 32-bit value to the **DORMANT** register of the chosen oscillator to stop it.

When exiting DORMANT mode, the chosen oscillator will restart. If you chose XOSC, the frequency will be more precise, but the restart will take more time due to startup delay (>1ms on the RP2350 reference design (see [Hardware design with RP2040, Minimal Design Example](#))). If you chose ROSC, the frequency will be less precise, but the start-up time is very short (approximately 1μs).

NOTE

You must stop the PLLs before entering DORMANT mode.

Pico Extras: https://github.com/raspberrypi/pico-extras/blob/master/src/p2_common/hardware_rosc/rosc.c Lines 56 - 61

```
56 void rosc_set_dormant(void) {
57     // WARNING: This stops the rosc until woken up by an irq
58     rosc_write(&rosc_hw->dormant, ROSC_DORMANT_VALUE_DORMANT);
59     // Wait for it to become stable once woken up
60     while(!(rosc_hw->status & ROSC_STATUS_STABLE_BITS));
61 }
```

WARNING

If you do not configure IRQ before entering DORMANT mode, neither oscillator will restart.

See [Section 6.3.7.2](#) for some examples of dormant mode.

8.3.8. List of Registers

The ROSC registers start at a base address of **0x400e8000** (defined as **ROSC_BASE** in SDK).

Table 605. List of ROSC registers

Offset	Name	Info
0x00	CTRL	Ring Oscillator control
0x04	FREQA	Ring Oscillator frequency control A
0x08	FREQB	Ring Oscillator frequency control B
0x0c	RANDOM	Loads a value to the LFSR randomiser
0x10	DORMANT	Ring Oscillator pause control
0x14	DIV	Controls the output divider

Offset	Name	Info
0x18	PHASE	Controls the phase shifted output
0x1c	STATUS	Ring Oscillator Status
0x20	RANDOMBIT	Returns a 1 bit random value
0x24	COUNT	A down counter running at the ROSC frequency which counts to zero and stops.

ROSC: CTRL Register

Offset: 0x00

Description

Ring Oscillator control

Table 606. CTRL Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:12	ENABLE	On power-up this field is initialised to ENABLE The system clock must be switched to another source before setting this field to DISABLE otherwise the chip will lock up The 12-bit code is intended to give some protection against accidental writes. An invalid setting will enable the oscillator. 0xd1e → DISABLE 0xfb → ENABLE	RW	-
11:0	FREQ_RANGE	Controls the number of delay stages in the ROSC ring LOW uses stages 0 to 7 MEDIUM uses stages 2 to 7 HIGH uses stages 4 to 7 TOOHIGH uses stages 6 to 7 and should not be used because its frequency exceeds design specifications The clock output will not glitch when changing the range up one step at a time The clock output will glitch when changing the range down Note: the values here are gray coded which is why HIGH comes before TOOHIGH 0xfa4 → LOW 0xfa5 → MEDIUM 0xfa7 → HIGH 0xfa6 → TOOHIGH	RW	0xaa0

ROSC: FREQA Register

Offset: 0x04

Description

The FREQA & FREQB registers control the frequency by controlling the drive strength of each stage

The drive strength has 4 levels determined by the number of bits set

Increasing the number of bits set increases the drive strength and increases the oscillation frequency

0 bits set is the default drive strength

1 bit set doubles the drive strength

2 bits set triples drive strength

3 bits set quadruples drive strength
 For frequency randomisation set both DS0_RANDOM=1 & DS1_RANDOM=1

Table 607. FREQA Register

Bits	Name	Description	Type	Reset
31:16	PASSWD	Set to 0x9696 to apply the settings Any other value in this field will set all drive strengths to 0 0x9696 → PASS	RW	0x0000
15	Reserved.	-	-	-
14:12	DS3	Stage 3 drive strength	RW	0x0
11	Reserved.	-	-	-
10:8	DS2	Stage 2 drive strength	RW	0x0
7	DS1_RANDOM	Randomises the stage 1 drive strength	RW	0x0
6:4	DS1	Stage 1 drive strength	RW	0x0
3	DS0_RANDOM	Randomises the stage 0 drive strength	RW	0x0
2:0	DS0	Stage 0 drive strength	RW	0x0

ROSC: FREQB Register

Offset: 0x08

Description

For a detailed description see freqa register

Table 608. FREQB Register

Bits	Name	Description	Type	Reset
31:16	PASSWD	Set to 0x9696 to apply the settings Any other value in this field will set all drive strengths to 0 0x9696 → PASS	RW	0x0000
15	Reserved.	-	-	-
14:12	DS7	Stage 7 drive strength	RW	0x0
11	Reserved.	-	-	-
10:8	DS6	Stage 6 drive strength	RW	0x0
7	Reserved.	-	-	-
6:4	DS5	Stage 5 drive strength	RW	0x0
3	Reserved.	-	-	-
2:0	DS4	Stage 4 drive strength	RW	0x0

ROSC: RANDOM Register

Offset: 0x0c

Description

Loads a value to the LFSR randomiser

Table 609. RANDOM Register

Bits	Name	Description	Type	Reset
31:0	SEED		RW	0x3f04b16d

ROSC: DORMANT Register

Offset: 0x10

Description

Ring Oscillator pause control

Table 610. DORMANT Register

Bits	Description	Type	Reset
31:0	This is used to save power by pausing the ROSC On power-up this field is initialised to WAKE An invalid write will also select WAKE Warning: setup the irq before selecting dormant mode 0x636f6d61 → DORMANT 0x77616b65 → WAKE	RW	-

ROSC: DIV Register

Offset: 0x14

Description

Controls the output divider

Table 611. DIV Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	set to 0xaa00 + div where div = 0 divides by 128 div = 1-127 divides by div any other value sets div=128 this register resets to div=32 0xaa00 → PASS	RW	-

ROSC: PHASE Register

Offset: 0x18

Description

Controls the phase shifted output

Table 612. PHASE Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:4	PASSWD	set to 0xaa any other value enables the output with shift=0	RW	0x00
3	ENABLE	enable the phase-shifted output this can be changed on-the-fly	RW	0x1
2	FLIP	invert the phase-shifted output this is ignored when div=1	RW	0x0
1:0	SHIFT	phase shift the phase-shifted output by SHIFT input clocks this can be changed on-the-fly must be set to 0 before setting div=1	RW	0x0

ROSC: STATUS Register

Offset: 0x1c

Description

Ring Oscillator Status

Table 613. STATUS Register

Bits	Name	Description	Type	Reset
31	STABLE	Oscillator is running and stable	RO	0x0
30:25	Reserved.	-	-	-
24	BADWRITE	An invalid value has been written to CTRL_ENABLE or CTRL_FREQ_RANGE or FREQA or FREQB or DIV or PHASE or DORMANT	WC	0x0
23:17	Reserved.	-	-	-
16	DIV_RUNNING	post-divider is running this resets to 0 but transitions to 1 during chip startup	RO	-
15:13	Reserved.	-	-	-
12	ENABLED	Oscillator is enabled but not necessarily running and stable this resets to 0 but transitions to 1 during chip startup	RO	-
11:0	Reserved.	-	-	-

ROSC: RANDOMBIT Register

Offset: 0x20

Table 614.
RANDOMBIT Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	This just reads the state of the oscillator output so randomness is compromised if the ring oscillator is stopped or run at a harmonic of the bus frequency	RO	0x1

ROSC: COUNT Register

Offset: 0x24

Table 615. COUNT Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	A down counter running at the ROSC frequency which counts to zero and stops. To start the counter write a non-zero value. Can be used for short software pauses when setting up time sensitive hardware.	RW	0x0000

8.4. Low Power Oscillator (LPOSC)

The Low Power Oscillator (LPOSC) provides a clock signal to the always-on logic when the main crystal oscillator is powered down in a low power (P1.x) state. It operates at a nominal 32.768kHz and is an RC oscillator, requiring no

external components. The oscillator's output clock is used to sequence initial chip start up and transition to and from low-power states. It can also be used by the AON-timer, see [Section 12.10](#).

The oscillator starts up as soon as the core power supply is available and power-on reset has been released. If brownout detection is enabled, the oscillator will be disabled when a core supply brownout is detected, but will restart as soon as the core supply has recovered and brownout reset has been released. The oscillator's frequency takes around 1ms to stabilise, and the chip will be held in reset during this period.

8.4.1. Frequency Accuracy and Calibration

The low power oscillator has an initial frequency accuracy of $\pm 20\%$. However, it can be trimmed to $\pm 1.5\%$ using the [TRIM](#) field in the [LPOS](#)C register. 63 trim steps are available, each between 1% and 3% of the oscillator's initial frequency. The frequency can be trimmed down by 32 steps or up by 31 steps. See [Table 616](#) and [Section 8.4.3](#) for details.

Table 616. low power oscillator output frequency and trimming

Parameter	Description	Min	Typ	Max	Units
$F_{0,\text{initial}}$	initial output frequency	26.2144	32.768	39.3216	kHz
$\text{trim}_{\text{STEP}}$	frequency trim step	-	1	3	% of initial output frequency
$F_{0,\text{trimmed}}$	trimmed output frequency	32.27648	32.768	33.25952	kHz

Frequency drift with temperature: $\pm 14\%$.

Frequency drift with power supply voltage: $\pm 20\%$.

8.4.2. Using an External Low Power Clock

Instead of using the low power RC oscillator, an external 32.768 kHz low power clock signal can be provided on one of GPIO 12, 14, 20, or 22. Alternatively, those GPIOs can be used to provide a 1 kHz or 1 Hz tick. See [Section 12.10.5.2](#) and [Section 12.10.7](#) for more details.

8.4.3. List of Registers

The low power oscillator shares register address space with other power management subsystems in the always-on domain. The address space is referred to as POWMAN elsewhere in this document. A complete list of POWMAN registers is provided in [Section 6.6](#), but information on registers associated with the low power oscillator is repeated here.

The POWMAN registers start at a base address of [0x40100000](#) (defined as [POWMAN_BASE](#) in SDK).

- [LPOS](#)C
- [EXT_TIME_REF](#)
- [LPOS](#)C_FREQ_KHZ_INT
- [LPOS](#)C_FREQ_KHZ_FRAC

8.5. Tick Generators

8.5.1. Overview

The ticks block generates time references for several blocks:

- [TIMER0](#)
- [TIMER1](#)
- RISC-V platform timer, provided by [SIO](#)
- Arm Cortex-M33 0 timer
- Arm Cortex-M33 1 timer
- [WATCHDOG](#)

The ticks block uses `clk_ref` as a reference clock, known internally as `clk_tick` to the ticks block. Ideally, `clk_ref` will be configured to use the Crystal Oscillator ([Section 8.2](#)) to provide an accurate reference clock. The reference clock is divided internally to generate a tick (nominally 1μs). There is a tick per destination block. Each tick can be configured differently.

Using a 12MHz reference clock, a 1μs tick would be generated by setting the cycle count to 12. A 1MHz clock has a period of 1μs, so the hardware needs to count for 12 times as many clock cycles to get a 1μs tick from a reference running at $12 \times 1\text{MHz}$.

Before changing the cycle count, always stop the tick generator with the [TIMER0_CTRL.ENABLE](#) bit.

8.5.2. List of Registers

The TICKS registers start at a base address of [0x40108000](#) (defined as [TICKS_BASE](#) in SDK).

Table 617. List of TICKS registers

Offset	Name	Info
0x00	PROC0_CTRL	Controls the tick generator
0x04	PROC0_CYCLES	
0x08	PROC0_COUNT	
0x0c	PROC1_CTRL	Controls the tick generator
0x10	PROC1_CYCLES	
0x14	PROC1_COUNT	
0x18	TIMER0_CTRL	Controls the tick generator
0x1c	TIMER0_CYCLES	
0x20	TIMER0_COUNT	
0x24	TIMER1_CTRL	Controls the tick generator
0x28	TIMER1_CYCLES	
0x2c	TIMER1_COUNT	
0x30	WATCHDOG_CTRL	Controls the tick generator
0x34	WATCHDOG_CYCLES	
0x38	WATCHDOG_COUNT	
0x3c	RISCV_CTRL	Controls the tick generator
0x40	RISCV_CYCLES	
0x44	RISCV_COUNT	

TICKS: PROC0_CTRL Register

Offset: 0x00

Description

Controls the tick generator

Table 618.
PROC0_CTRL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	RUNNING	Is the tick generator running?	RO	-
0	ENABLE	start / stop tick generation	RW	0x0

TICKS: PROC0_CYCLES Register

Offset: 0x04

Table 619.
PROC0_CYCLES Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Total number of clk_tick cycles before the next tick.	RW	0x000

TICKS: PROC0_COUNT Register

Offset: 0x08

Table 620.
PROC0_COUNT Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Count down timer: the remaining number clk_tick cycles before the next tick is generated.	RO	-

TICKS: PROC1_CTRL Register

Offset: 0x0c

Description

Controls the tick generator

Table 621.
PROC1_CTRL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	RUNNING	Is the tick generator running?	RO	-
0	ENABLE	start / stop tick generation	RW	0x0

TICKS: PROC1_CYCLES Register

Offset: 0x10

Table 622.
PROC1_CYCLES
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Total number of clk_tick cycles before the next tick.	RW	0x000

TICKS: PROC1_COUNT Register

Offset: 0x14

Table 623.
PROC1_COUNT
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Count down timer: the remaining number clk_tick cycles before the next tick is generated.	RO	-

TICKS: TIMER0_CTRL Register

Offset: 0x18

Description

Controls the tick generator

Table 624.
TIMER0_CTRL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	RUNNING	Is the tick generator running?	RO	-
0	ENABLE	start / stop tick generation	RW	0x0

TICKS: TIMER0_CYCLES Register

Offset: 0x1c

Table 625.
TIMER0_CYCLES
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Total number of clk_tick cycles before the next tick.	RW	0x000

TICKS: TIMER0_COUNT Register

Offset: 0x20

Table 626.
TIMER0_COUNT
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Count down timer: the remaining number clk_tick cycles before the next tick is generated.	RO	-

TICKS: TIMER1_CTRL Register

Offset: 0x24

Description

Controls the tick generator

Table 627.
TIMER1_CTRL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
1	RUNNING	Is the tick generator running?	RO	-
0	ENABLE	start / stop tick generation	RW	0x0

TICKS: TIMER1_CYCLES Register

Offset: 0x28

Table 628.
TIMER1_CYCLES
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Total number of clk_tick cycles before the next tick.	RW	0x000

TICKS: TIMER1_COUNT Register

Offset: 0x2c

Table 629.
TIMER1_COUNT
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Count down timer: the remaining number clk_tick cycles before the next tick is generated.	RO	-

TICKS: WATCHDOG_CTRL Register

Offset: 0x30

Description

Controls the tick generator

Table 630.
WATCHDOG_CTRL
Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	RUNNING	Is the tick generator running?	RO	-
0	ENABLE	start / stop tick generation	RW	0x0

TICKS: WATCHDOG_CYCLES Register

Offset: 0x34

Table 631.
WATCHDOG_CYCLES
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Total number of clk_tick cycles before the next tick.	RW	0x000

TICKS: WATCHDOG_COUNT Register

Offset: 0x38

Table 632.
WATCHDOG_COUNT
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Count down timer: the remaining number clk_tick cycles before the next tick is generated.	RO	-

TICKS: RISCV_CTRL Register

Offset: 0x3c

Description

Controls the tick generator

Table 633.
RISCV_CTRL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	RUNNING	Is the tick generator running?	RO	-
0	ENABLE	start / stop tick generation	RW	0x0

TICKS: RISCV_CYCLES Register

Offset: 0x40

Table 634.
RISCV_CYCLES
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Total number of clk_tick cycles before the next tick.	RW	0x000

TICKS: RISCV_COUNT Register

Offset: 0x44

Table 635.
RISCV_COUNT
Register

Bits	Description	Type	Reset
31:9	Reserved.	-	-
8:0	Count down timer: the remaining number clk_tick cycles before the next tick is generated.	RO	-

8.6. PLL

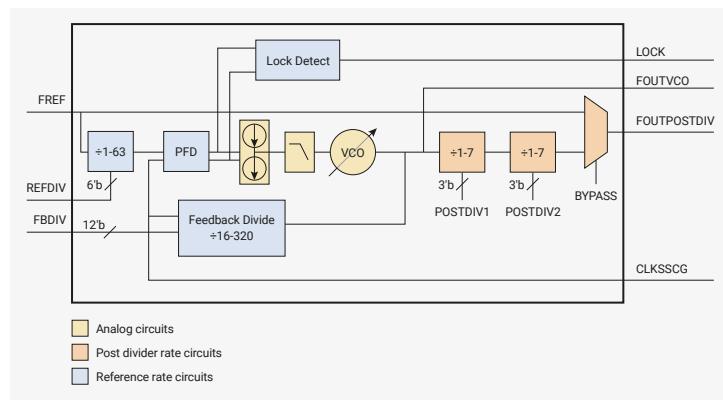
8.6.1. Overview

The PLL takes a reference clock and multiplies it using a Voltage Controlled Oscillator (VCO) with a feedback loop. The VCO runs at high frequencies: between 400 and 1600MHz. As a result, there are two **post dividers** that can divide the VCO frequency before it is distributed to the clock generators on the chip.

There are two PLLs in RP2350. They are:

- **pll_sys** - used to generate up to a 150MHz system clock
- **pll_usb** - used to generate a 48MHz USB reference clock

Figure 37. On both PLLs, the FREF (reference) input is connected to the crystal oscillator's XIN (XI) input. The PLL contains a VCO, which is locked to a constant ratio of the reference clock via the feedback loop (phase-frequency detector and loop filter). This can synthesise very high frequencies, which may be divided down by the post-dividers.



8.6.2. Changes from RP2040

- RP2350 added an interrupt that fires if the PLL loses lock. See `CS.LOCK_N`.

8.6.3. Calculating PLL parameters

To configure the PLL, you must know the frequency of the reference clock, which is routed directly from the crystal oscillator. This will often be a 12MHz crystal, for compatibility with RP2350's USB bootrom. The PLL's final output frequency `FOUTPOSTDIV` can then be calculated as $(\text{FREF} / \text{REFDIV}) \times \text{FBDIV} / (\text{POSTDIV1} \times \text{POSTDIV2})$. With a desired output frequency in mind, you must select PLL parameters according to the following constraints of the PLL design:

- minimum reference frequency (`FREF / REFDIV`) is 5MHz
- oscillator frequency (`FOUTVCO`) must be in the range 750MHz-1600MHz
- feedback divider (`FBDIV`) must be in the range 16-320
- the post dividers `POSTDIV1` and `POSTDIV2` must be in the range 1-7
- maximum input frequency (`FREF / REFDIV`) is VCO frequency divided by 16, due to minimum feedback divisor

You must also respect the maximum frequencies of the chip's clock generators (attached to `FOUTPOSTDIV`). For the system PLL this is 150MHz, and for the USB PLL, 48MHz. If using a crystal oscillator with a frequency of less than 75MHz, `REFDIV` should be 1 assuming a VCO of 1200MHz-1600MHz. If using a fast crystal with a low VCO frequency, the reference divisor may need to be increased to keep the PLL input within a suitable range.

TIP

When two different values are required for `POSTDIV1` and `POSTDIV2`, assign the higher value to `POSTDIV1` for lower power consumption.

In the RP2350 reference design (see [Hardware design with RP2040, Minimal Design Example](#)), which attaches a 12MHz crystal to the crystal oscillator, the minimum achievable VCO frequency is $12\text{MHz} \times 42 = 504\text{MHz}$, and the maximum achievable VCO frequency is $12\text{MHz} \times 133 = 1596\text{MHz}$. As a result, `FBDIV` must remain in the range 42-133. Setting `FBDIV` to 100 would synthesise a 1200MHz VCO frequency. A `POSTDIV1` value of 6 and a `POSTDIV2` value of 2 would divide this by 12 in total, producing a clean 100MHz at the PLL's final output.

8.6.3.1. Jitter vs Power Consumption

Often, several sets of PLL configuration parameters achieve the desired output frequency (or a close approximation). You decide whether to prioritise lower power consumption or lower **jitter**: cycle-to-cycle variation in the PLL's output clock period. Jitter decreases as VCO frequency increases, because you can use higher post-divide values. Consider the following scenarios:

- 1500MHz VCO / 6 / 2 = 125MHz
- 500MHz VCO / 4 / 1 = 125MHz

The 1500MHz configuration uses the most power, but produces the least jitter. The 500MHz configuration uses the least power, but produces the most jitter.

You can slightly adjust the desired output frequency to allow for a much lower VCO frequency by bringing the output to a closer rational multiple of the input. Some frequencies are not be achievable at all with a possible VCO frequency and combination of divisors.

Because RP2350's digital logic compensates for the worst possible jitter on the system clock, this doesn't affect system stability. However, applications often require a highly accurate clock for data transfers that follow the USB specification, which defines a maximum amount of allowable jitter.

SDK provides a Python script that searches for the best VCO and post divider options for a desired output frequency:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_clocks/scripts/vcocalc.py

```

1 #!/usr/bin/env python3
2
3 import argparse
4
5 # Fixed hardware parameters
6 fbdv_range = range(16, 320 + 1)
7 postdiv_range = range(1, 7 + 1)
8 ref_min = 5
9 refdiv_min = 1
10 refdiv_max = 63
11
12 def validRefdiv(string):
13     if ((int(string) < refdiv_min) or (int(string) > refdiv_max)):
14         raise ValueError("REFDIV must be in the range {} to {}".format(refdiv_min,
15                         refdiv_max))
15     return int(string)
16
17 parser = argparse.ArgumentParser(description="PLL parameter calculator")
18 parser.add_argument("--input", "-i", default=12, help="Input (reference) frequency. Default
19 12 MHz", type=float)
19 parser.add_argument("--ref-min", default=5, help="Override minimum reference frequency.
20 Default 5 MHz", type=float)
20 parser.add_argument("--vco-max", default=1600, help="Override maximum VCO frequency. Default
21 1600 MHz", type=float)
21 parser.add_argument("--vco-min", default=750, help="Override minimum VCO frequency. Default
22 750 MHz", type=float)
22 parser.add_argument("--lock-refdiv", help="Lock REFDIV to specified number in the range {} to
23 {}.".format(refdiv_min, refdiv_max), type=validRefdiv)
23 parser.add_argument("--low-vco", "-l", action="store_true", help="Use a lower VCO frequency
24 when possible. This reduces power consumption, at the cost of increased jitter")
24 parser.add_argument("output", help="Output frequency in MHz.", type=float)
25 args = parser.parse_args()
26
27 refdiv_range = range(refdiv_min, max(refdiv_min, min(refdiv_max, int(args.input / args
28 .ref_min))) + 1)
28 if args.lock_refdiv:
29     print("Locking REFDIV to", args.lock_refdiv)
30     refdiv_range = [args.lock_refdiv]
31
32 best = (0, 0, 0, 0, 0)
33 best_margin = args.output
34
35 for refdiv in refdiv_range:
36     for fbdv in (fbdv_range if args.low_vco else reversed(fbdv_range)):
37         vco = args.input / refdiv * fbdv

```

```

38     if vco < args.vco_min or vco > args.vco_max:
39         continue
40     # pd1 is inner loop so that we prefer higher ratios of pd1:pd2
41     for pd2 in postdiv_range:
42         for pd1 in postdiv_range:
43             out = vco / pd1 / pd2
44             margin = abs(out - args.output)
45             if ((vco * 1000) % (pd1 * pd2)):
46                 continue
47             elif margin < best_margin:
48                 best = (out, fbdv, pd1, pd2, refdiv)
49                 best_margin = margin
50
51 if best[0] > 0:
52     print("Requested: {} MHz".format(args.output))
53     print("Achieved: {} MHz".format(best[0]))
54     print("REFDIV: {}".format(best[4]))
55     print("FBDIV: {} (VCO = {} MHz)".format(best[1], args.input / best[4] * best[1]))
56     print("PD1: {}".format(best[2]))
57     print("PD2: {}".format(best[3]))
58     if best[4] != 1:
59         print(
60 """
61 As the requires a non-standard refdiv, this will
62 need to be set as a compile definition
63 Add the following to your CMakeLists.txt file to
64 configure the clocks as calculated above
65
66 target_compile_definitions(executable_name PRIVATE
67     PLL_SYS_REFDIV={}
68     PLL_SYS_VCO_FREQ_HZ={}
69     PLL_SYS_POSTDIV1={}
70     PLL_SYS_POSTDIV2={}
71 )\
72 """.format(best[4], int((args.input * 1_000_000) / best[4] * best[1]), best[2], best[3]))
73 else:
74     print("UNABLE TO COME UP WITH A SOLUTION....")

```

Given an input and output frequency, this script finds the best possible set of PLL parameters. When the script finds multiple equally good combinations, it returns the parameters which yield the highest VCO frequency, for the best output stability. Pass the `-l` or `--low-vco` flag to prefer lower frequencies, which reduce power consumption. Pass the `--vco-max` flag to limit the maximum VCO frequency. If the script cannot find an exact match given the provided constraints, it outputs the closest reasonable match instead.

The following example uses the script to request a 48MHz output with the best output stability:

```

$ ./vcocalc.py 48
Requested: 48.0 MHz
Achieved: 48.0 MHz
FBDIV: 120 (VCO = 1440 MHz)
PD1: 6
PD2: 5

```

The following example uses the script to request a 48MHz output with the lowest power consumption:

```

$ ./vcocalc.py -l 48
Requested: 48.0 MHz
Achieved: 48.0 MHz
FBDIV: 36 (VCO = 432 MHz)

```

```
PD1: 3
PD2: 3
```

The following example uses the script to request a 125MHz output with the lowest power consumption, but the minimum VCO frequency remains quite high:

```
$ ./vcocalc.py -l 125
Requested: 125.0 MHz
Achieved: 125.0 MHz
FBDIV: 125 (VCO = 1500 MHz)
PD1: 6
PD2: 2
```

The following example uses the script to request a 125MHz system clock, restricting the search to VCO frequencies below 600MHz. There is no exact match, so the script considers near (but not exact) frequency matches. Relaxing the search to allow nearby non-exact matches significantly reduces the minimum VCO frequency compared to the previous example:

```
$ ./vcocalc.py -l 125 --vco-max 600
Requested: 125.0 MHz
Achieved: 126.0 MHz
FBDIV: 42 (VCO = 504 MHz)
PD1: 4
PD2: 1
```

A 126MHz system clock may be a tolerable deviation from the desired 125MHz, and generating this clock consumes less power at the PLL.

8.6.4. Configuration

The SDK uses the following PLL settings:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_clocks/include/hardware/clocks.h Lines 93 - 115

```
93 //
94 // There are two PLLs in RP2040:
95 // 1. The 'SYS PLL' generates the 125MHz system clock, the frequency is defined by
96 //    'SYS_CLK_HZ'.
97 //
98 // The two PLLs use the crystal oscillator output directly as their reference frequency input;
99 //    the PLLs reference
100 // frequency cannot be reduced by the dividers present in the clocks block. The crystal
101 //    frequency is defined by 'XOSC_HZ' (or
102 //    'XOSC_KHZ' or 'XOSC_MHZ').
103 //
104 // The system's default definitions are correct for the above frequencies with a 12MHz
105 //    crystal frequency. If different frequencies are required, these must be defined in
106 //    the board configuration file together with the revised PLL settings
107 // Use 'vcocalc.py' to check and calculate new PLL settings if you change any of these
108 //    frequencies.
109 // Default PLL configuration RP2040:
110 //          REF      FBDIV  VCO          POSTDIV
111 // PLL SYS: 12 / 1 = 12MHz * 125 = 1500MHz / 6 / 2 = 125MHz
```

```

110 // PLL USB: 12 / 1 = 12MHz * 100 = 1200MHz / 5 / 5 = 48MHz
111 //
112 // Default PLL configuration RP2350:
113 //           REF      FBDIV   VCO          POSTDIV
114 // PLL SYS: 12 / 1 = 12MHz * 125 = 1500MHz / 5 / 2 = 150MHz
115 // PLL USB: 12 / 1 = 12MHz * 100 = 1200MHz / 5 / 5 = 48MHz

```

The `pll_init` function in the SDK (examined below) asserts that all of these conditions are true before attempting to configure the PLL.

The SDK defines the PLL control registers as a struct. It then maps them into memory for each instance of the PLL.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2350/hardware_structs/include/hardware/structs/pll.h Lines 24 - 74

```

24 typedef struct {
25     _REG_(PLL_CS_OFFSET) // PLL_CS
26     // Control and Status
27     // 0x80000000 [31] : LOCK (0): PLL is locked
28     // 0x40000000 [30] : LOCK_N (0): PLL is not locked
29     // 0x00000100 [8] : BYPASS (0): Passes the reference clock to the output instead of the
30     // divided VCO
31     // 0x0000003f [5:0] : REFDIV (1): Divides the PLL input reference clock
32     io_rw_32 cs;
33
34     _REG_(PLL_PWR_OFFSET) // PLL_PWR
35     // Controls the PLL power modes
36     // 0x00000020 [5] : VCOPD (1): PLL VCO powerdown
37     // 0x00000008 [3] : POSTDIVPD (1): PLL post divider powerdown
38     // 0x00000004 [2] : DSMPD (1): PLL DSM powerdown
39     // 0x00000001 [0] : PD (1): PLL powerdown
40     io_rw_32 pwr;
41
42     _REG_(PLL_FBDIV_INT_OFFSET) // PLL_FBDIV_INT
43     // Feedback divisor
44     // 0x0000ffff [11:0] : FBDIV_INT (0): see ctrl reg description for constraints
45     io_rw_32 fbddiv_int;
46
47     _REG_(PLL_PRIM_OFFSET) // PLL_PRIM
48     // Controls the PLL post dividers for the primary output
49     // 0x00070000 [18:16] : POSTDIV1 (0x7): divide by 1-7
50     // 0x00007000 [14:12] : POSTDIV2 (0x7): divide by 1-7
51     io_rw_32 prim;
52
53     _REG_(PLL_INTR_OFFSET) // PLL_INTR
54     // Raw Interrupts
55     // 0x00000001 [0] : LOCK_N_STICKY (0)
56     io_rw_32 intr;
57
58     _REG_(PLL_INTE_OFFSET) // PLL_INTE
59     // Interrupt Enable
60     // 0x00000001 [0] : LOCK_N_STICKY (0)
61     io_rw_32 inte;
62
63     _REG_(PLL_INTF_OFFSET) // PLL_INTF
64     // Interrupt Force
65     // 0x00000001 [0] : LOCK_N_STICKY (0)
66     io_rw_32 intf;
67
68     _REG_(PLL_INTS_OFFSET) // PLL_INTS
69     // Interrupt status after masking & forcing
70     // 0x00000001 [0] : LOCK_N_STICKY (0)
71     io_ro_32 ints;

```

```

71 } pll_hw_t;
72
73 #define pll_sys_hw ((pll_hw_t *)PLL_SYS_BASE)
74 #define pll_usb_hw ((pll_hw_t *)PLL_USB_BASE)

```

The SDK defines `pll_init`, which is used to configure or reconfigure a PLL. It starts by clearing any previous power state in the PLL, then calculates the appropriate feedback divider value. There are assertions to check that these values satisfy the constraints above.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_pll/pll.c Lines 13 - 21

```

13 void pll_init(PLL pll, uint refdiv, uint vco_freq, uint post_div1, uint post_div2) {
14     uint32_t ref_freq = XOSC_HZ / refdiv;
15
16     // Check vco freq is in an acceptable range
17     assert(vco_freq >= PICO_PLL_VCO_MIN_FREQ_HZ && vco_freq <= PICO_PLL_VCO_MAX_FREQ_HZ);
18
19     // What are we multiplying the reference clock by to get the vco freq
20     // (The regs are called div, because you divide the vco output and compare it to the
21     // refclk)
21     uint32_t fbddiv = vco_freq / ref_freq;

```

The programming sequence for the PLL is as follows:

1. Program the reference clock divider (is a divide by 1 in the RP2350 case).
2. Program the feedback divider.
3. Turn on the main power and VCO.
4. Wait for the VCO to achieve a stable frequency, as indicated by the `LOCK` status flag.
5. Set up post dividers and turn them on.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_pll/pll.c Lines 42 - 69

```

42     if ((pll->cs & PLL_CS_LOCK_BITS) &&
43         (refdiv == (pll->cs & PLL_CS_REFDIV_BITS)) &&
44         (fbddiv == (pll->fbddiv_int & PLL_FBDIV_INT_BITS)) &&
45         (pdiv == (pll->prim & (PLL_PRIM_POSTDIV1_BITS | PLL_PRIM_POSTDIV2_BITS))) {
46         // do not disrupt PLL that is already correctly configured and operating
47         return;
48     }
49
50     reset_unreset_block_num_wait_blocking(PLL_RESET_NUM(pll));
51
52     // Load VCO-related dividers before starting VCO
53     pll->cs = refdiv;
54     pll->fbddiv_int = fbddiv;
55
56     // Turn on PLL
57     uint32_t power = PLL_PWR_PD_BITS | // Main power
58                           PLL_PWR_VCOPD_BITS; // VCO Power
59
60     hw_clear_bits(&pll->pwr, power);
61
62     // Wait for PLL to lock
63     while (!(pll->cs & PLL_CS_LOCK_BITS)) tight_loop_contents();
64
65     // Set up post dividers
66     pll->prim = pdiv;
67

```

```

68 // Turn on post divider
69 hw_clear_bits(&pll->pwr, PLL_PWR_POSTDIVPD_BITS);

```

The VCO turns on first, followed by the post dividers, so the PLL does not output a dirty clock while the VCO is locked.

8.6.5. List of Registers

The PLL_SYS and PLL_USB registers start at base addresses of `0x40050000` and `0x40058000` respectively (defined as `PLL_SYS_BASE` and `PLL_USB_BASE` in SDK).

Table 636. List of PLL registers

Offset	Name	Info
0x00	CS	Control and Status
0x04	PWR	Controls the PLL power modes.
0x08	FBDIV_INT	Feedback divisor
0x0c	PRIM	Controls the PLL post dividers for the primary output
0x10	INTR	Raw Interrupts
0x14	INTE	Interrupt Enable
0x18	INTF	Interrupt Force
0x1c	INTS	Interrupt status after masking & forcing

PLL: CS Register

Offset: 0x00

Description

Control and Status

GENERAL CONSTRAINTS:

Reference clock frequency min=5MHz, max=800MHz

Feedback divider min=16, max=320

VCO frequency min=400MHz, max=1600MHz

Table 637. CS Register

Bits	Name	Description	Type	Reset
31	LOCK	PLL is locked	RO	0x0
30	LOCK_N	PLL is not locked Ideally this is cleared when PLL lock is seen and this should never normally be set	WC	0x0
29:9	Reserved.	-	-	-
8	BYPASS	Passes the reference clock to the output instead of the divided VCO. The VCO continues to run so the user can switch between the reference clock and the divided VCO but the output will glitch when doing so.	RW	0x0
7:6	Reserved.	-	-	-
5:0	REFDIV	Divides the PLL input reference clock. Behaviour is undefined for div=0. PLL output will be unpredictable during refdiv changes, wait for lock=1 before using it.	RW	0x01

PLL: PWR Register

Offset: 0x04

Description

Controls the PLL power modes.

Table 638. PWR Register

Bits	Name	Description	Type	Reset
31:6	Reserved.	-	-	-
5	VCOPD	PLL VCO powerdown To save power set high when PLL output not required or bypass=1.	RW	0x1
4	Reserved.	-	-	-
3	POSTDIVPD	PLL post divider powerdown To save power set high when PLL output not required or bypass=1.	RW	0x1
2	DSMPD	PLL DSM powerdown Nothing is achieved by setting this low.	RW	0x1
1	Reserved.	-	-	-
0	PD	PLL powerdown To save power set high when PLL output not required.	RW	0x1

PLL: FBDIV_INT Register

Offset: 0x08

Description

Feedback divisor

(note: this PLL does not support fractional division)

Table 639. FBDIV_INT Register

Bits	Description	Type	Reset
31:12	Reserved.	-	-
11:0	see ctrl reg description for constraints	RW	0x000

PLL: PRIM Register

Offset: 0x0c

Description

Controls the PLL post dividers for the primary output

(note: this PLL does not have a secondary output)

the primary output is driven from VCO divided by postdiv1*postdiv2

Table 640. PRIM Register

Bits	Name	Description	Type	Reset
31:19	Reserved.	-	-	-
18:16	POSTDIV1	divide by 1-7	RW	0x7
15	Reserved.	-	-	-
14:12	POSTDIV2	divide by 1-7	RW	0x7
11:0	Reserved.	-	-	-

PLL: INTR Register

Offset: 0x10

Description

Raw Interrupts

Table 641. INTR Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	LOCK_N_STICKY		WC	0x0

PLL: INTE Register

Offset: 0x14

Description

Interrupt Enable

Table 642. INTE Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	LOCK_N_STICKY		RW	0x0

PLL: INTF Register

Offset: 0x18

Description

Interrupt Force

Table 643. INTF Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	LOCK_N_STICKY		RW	0x0

PLL: INTS Register

Offset: 0x1c

Description

Interrupt status after masking & forcing

Table 644. INTS Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	LOCK_N_STICKY		RO	0x0

Chapter 9. GPIO

9.1. Overview

RP2350 has up to 54 multi-functional General Purpose Input / Output (GPIO) pins, divided into two banks:

Bank 0

30 user GPIOs in the QFN-60 package (RP2350A), or 48 user GPIOs in the QFN-80 package

Bank 1

six QSPI IOs, and the USB DP/DM pins

You can control each GPIO from software running on the processors, or by a number of other functional blocks. To meet USB rise and fall specifications, the analogue characteristics of the USB pins differ from the GPIO pads. As a result, we do not include them in the 54 GPIO total. However, you can still use them for UART or I2C.

In a typical use case, the QSPI IOs are used to execute code from an external flash device, leaving 30 or 48 Bank 0 GPIOs for the programmer to use. The QSPI pins may become available for general purpose use when booting the chip from internal OTP, or controlling the chip externally via SWD in an IO expander application.

All GPIOs support digital input and output. Several Bank 0 GPIOs can also be used as inputs to the chip's Analogue to Digital Converter (ADC):

- GPIOs 26 through 29 inclusive (four total) in the QFN-60 package
- GPIOs 40 through 47 (eight total) in the QFN-80 package

Bank 0 supports the following functions:

- Software control via Single-cycle IO (SIO) – [Section 3.1.3, "GPIO Control"](#)
- Programmable IO (PIO) – [Chapter 11, PIO](#)
- 2 × SPI – [Section 12.3, "SPI"](#)
- 2 × UART – [Section 12.1, "UART"](#)
- 2 × I2C (two-wire serial interface) – [Section 12.2, "I2C"](#)
- 8 × two-channel PWM – [Section 12.5, "PWM"](#)
- 2 × external clock inputs – [Section 8.1.1.4, "External Clocks"](#)
- 4 × general purpose clock output – [Section 8.1, "Overview"](#)
- 4 × input to ADC – [Section 12.4, "ADC and Temperature Sensor"](#)
- 1 × HSTX high-speed interface – [Section 12.11, "HSTX"](#)
- 1 × auxiliary QSPI chip select, for a second XIP device – [Section 12.14, "QSPI Memory Interface \(QMI\)"](#)
- CoreSight execution trace output – [Section 3.5.7, "Trace"](#)
- USB VBUS management – [Section 12.7.3.10, "VBUS Control"](#)
- External interrupt requests, level or edge-sensitive – [Section 9.5, "Interrupts"](#)

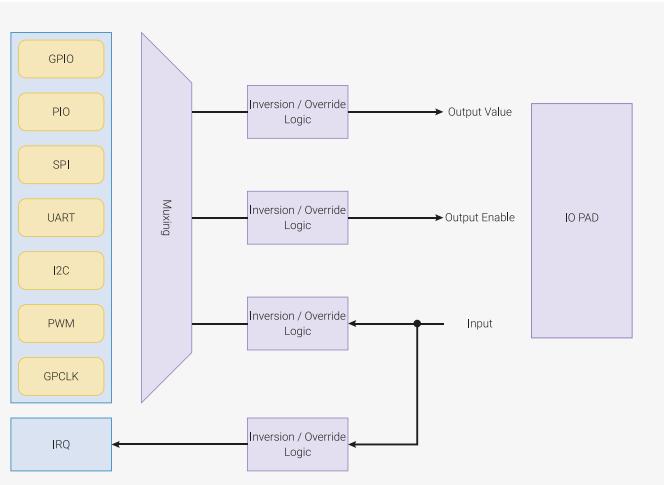
Bank 1 contains the QSPI and USB DP/DM pins and supports the following functions:

- Software control via Single-cycle IO (SIO) – [Section 3.1.3, "GPIO Control"](#)
- Flash execute in place ([Section 4.4, "External Flash and PSRAM \(XIP\)"](#)) via QSPI Memory Interface (QMI) – [Section 12.14, "QSPI Memory Interface \(QMI\)"](#)

- UART – [Section 12.1, “UART”](#)
- I2C (two-wire serial interface) – [Section 12.2, “I2C”](#)

The logical structure of an example IO is shown in Figure 38.

Figure 38. Logical structure of a GPIO.
Each GPIO can be controlled by one of a number of peripherals, or by software control registers in the SIO. The function select (FSEL) selects which peripheral output is in control of the GPIO's direction and output level, and which peripheral input can see this GPIO's input level. These three signals (output level, output enable, input level) can also be inverted or forced high or low, using the GPIO control registers.



9.2. Changes from RP2040

RP2350 GPIO differs from RP2040 in the following ways:

- 18 more GPIOs in the QFN-80 package
- Addition of a third PIO to GPIO functions
- USB DP/DM pins can be used as GPIO
- Addition of isolation register to pad registers (preserves pad state while in a low power state, cleared by software on power up)
- Changed default reset state of pad controls
- Both Secure and Non-secure access to GPIOs (see [Section 10.5](#))
- Double the number of GPIO interrupts to differentiate between Secure and Non-secure
- Interrupt summary registers added so you can quickly see which GPIOs have pending interrupts

9.3. Reset State

At first power up, Bank 0 IOs (GPIOs 0 through 29 in the QFN-60 package, and GPIOs 0 through 47 in the QFN-80 package) assume the following state:

- Output buffer is high-impedance
- Input buffer is disabled
- Pulled low
- Isolation latches are set to latched ([Section 9.7](#))

The pad output disable bit (`GPIO0.OD`) for each pad is clear at reset, but the IO muxing is reset to the null function, which ensures that the output buffer is high-impedance.

⚠️ IMPORTANT

The pad reset state is different from RP2040, which only disables digital inputs on GPIOs 26 through 29 (as of version B2) and does not have isolation latches. Applications must enable the pad input (`GPIO0.IE = 1`) and disable pad isolation latches (`GPIO0.ISO = 0`) before using the pads for digital I/O. The `gpio_set_function()` SDK function performs these tasks automatically.

Bank 1 IOs have the same reset state as Bank 0 GPIOs, except for the input enable (IE) resetting to 1, and different pull-up/pull-down states: SCK, SD0 and SD1 are pull-down, but SD2, SD3 and CSn are pull-up.

ℹ️ NOTE

To use a Bank 0 GPIO as a second chip select, you need an external pull-up to ensure the second QSPI device does not power up with its chip select asserted.

The pads return to the reset state on any of the following:

- A brownout reset
- Asserting the RUN pin low
- Setting SW-DP CDBGIRSTREQ via SWD
- Setting RP-AP rescue reset via SWD

If a pad's isolation latches are in the latched state (Section 9.7) then resetting the PADS and IO registers does not physically return the pad to its reset state. The isolation latches prevent upstream signals from propagating to the pad. Clear the ISO bit to allow signals to propagate.

9.4. Function Select

To allocate a function to a GPIO, write to the `FUNCSEL` field in the `CTRL` register corresponding to the pin. For a list of GPIOs and corresponding registers, see Table 645. For an example, see `GPIO0_CTRL`. The descriptions for the functions listed in this table can be found in Table 646.

Each GPIO can only select one function at a time. Each peripheral input (e.g. UART0 RX) should only be selected by one GPIO at a time. If you connect the same peripheral input to multiple GPIOs, the peripheral sees the logical OR of these GPIO inputs.

Table 645. General Purpose Input/Output (GPIO) Bank 0 Functions

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
0		SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2	XIP_CS1n	USB OVCUR DET	
1		SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2	TRACECLK	USB VBUS DET	
2		SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2	TRACEDATA0	USB VBUS EN	UART0 TX
3		SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	TRACEDATA1	USB OVCUR DET	UART0 RX
4		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	TRACEDATA2	USB VBUS DET	
5		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	TRACEDATA3	USB VBUS EN	
6		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART1 TX
7		SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 RX
8		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	XIP_CS1n	USB VBUS EN	
9		SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	
10		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	UART1 TX
11		SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	UART1 RX
12	HSTX	SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB OVCUR DET	
13	HSTX	SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB VBUS DET	
14	HSTX	SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO1	USB VBUS EN	UART0 TX
15	HSTX	SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT1	USB OVCUR DET	UART0 RX
16	HSTX	SPI0 RX	UART0 TX	I2C0 SDA	PWM0 A	SIO	PIO0	PIO1	PIO2		USB VBUS DET	
17	HSTX	SPI0 CSn	UART0 RX	I2C0 SCL	PWM0 B	SIO	PIO0	PIO1	PIO2		USB VBUS EN	
18	HSTX	SPI0 SCK	UART0 CTS	I2C1 SDA	PWM1 A	SIO	PIO0	PIO1	PIO2		USB OVCUR DET	UART0 TX
19	HSTX	SPI0 TX	UART0 RTS	I2C1 SCL	PWM1 B	SIO	PIO0	PIO1	PIO2	XIP_CS1n	USB VBUS DET	UART0 RX
20		SPI0 RX	UART1 TX	I2C0 SDA	PWM2 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO0	USB VBUS EN	
21		SPI0 CSn	UART1 RX	I2C0 SCL	PWM2 B	SIO	PIO0	PIO1	PIO2	CLOCK GPOUT0	USB OVCUR DET	
22		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM3 A	SIO	PIO0	PIO1	PIO2	CLOCK GPIO1	USB VBUS DET	UART1 TX

GPIO	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
23		SPI0 TX	UART1 RTS	I2C1 SCL	PWM3 B	SIO	PIO0	PIO1	PIO2	CLOCK
24		SPI1 RX	UART1 TX	I2C0 SDA	PWM4 A	SIO	PIO0	PIO1	PIO2	CLOCK
25		SPI1 CSn	UART1 RX	I2C0 SCL	PWM4 B	SIO	PIO0	PIO1	PIO2	CLOCK
26		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM5 A	SIO	PIO0	PIO1	PIO2	
27		SPI1 TX	UART1 RTS	I2C1 SCL	PWM5 B	SIO	PIO0	PIO1	PIO2	
28		SPI1 RX	UART0 TX	I2C0 SDA	PWM6 A	SIO	PIO0	PIO1	PIO2	
29		SPI1 CSn	UART0 RX	I2C0 SCL	PWM6 B	SIO	PIO0	PIO1	PIO2	

GPIOs 30 through 47 are QFN-80 only:

30		SPI1 SCK	UART0 CTS	I2C1 SDA	PWM7 A	SIO	PIO0	PIO1	PIO2	
31		SPI1 TX	UART0 RTS	I2C1 SCL	PWM7 B	SIO	PIO0	PIO1	PIO2	
32		SPI0 RX	UART0 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2	
33		SPI0 CSn	UART0 RX	I2C0 SCL	PWM8 B	SIO	PIO0	PIO1	PIO2	
34		SPI0 SCK	UART0 CTS	I2C1 SDA	PWM9 A	SIO	PIO0	PIO1	PIO2	
35		SPI0 TX	UART0 RTS	I2C1 SCL	PWM9 B	SIO	PIO0	PIO1	PIO2	
36		SPI0 RX	UART1 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2	
37		SPI0 CSn	UART1 RX	I2C0 SCL	PWM10 B	SIO	PIO0	PIO1	PIO2	
38		SPI0 SCK	UART1 CTS	I2C1 SDA	PWM11 A	SIO	PIO0	PIO1	PIO2	
39		SPI0 TX	UART1 RTS	I2C1 SCL	PWM11 B	SIO	PIO0	PIO1	PIO2	
40		SPI1 RX	UART1 TX	I2C0 SDA	PWM8 A	SIO	PIO0	PIO1	PIO2	
41		SPI1 CSn	UART1 RX	I2C0 SCL	PWM8 B	SIO	PIO0	PIO1	PIO2	
42		SPI1 SCK	UART1 CTS	I2C1 SDA	PWM9 A	SIO	PIO0	PIO1	PIO2	
43		SPI1 TX	UART1 RTS	I2C1 SCL	PWM9 B	SIO	PIO0	PIO1	PIO2	
44		SPI1 RX	UART0 TX	I2C0 SDA	PWM10 A	SIO	PIO0	PIO1	PIO2	
45		SPI1 CSn	UART0 RX	I2C0 SCL	PWM10 B	SIO	PIO0	PIO1	PIO2	
46		SPI1 SCK	UART0 CTS	I2C1 SDA	PWM11 A	SIO	PIO0	PIO1	PIO2	
47		SPI1 TX	UART0 RTS	I2C1 SCL	PWM11 B	SIO	PIO0	PIO1	PIO2	XIP_C

Table 646. GPIO User Bank function descriptions

Function Name	Description
SPIx	Connect one of the internal PL022 SPI peripherals to GPIO.
UARTx	Connect one of the internal PL011 UART peripherals to GPIO.
I2Cx	Connect one of the internal DW I2C peripherals to GPIO.
PWMx A/B	Connect a PWM slice to GPIO. There are twelve PWM slices, each with two output channels (A/B). The B pin can also be used as an input, for frequency and duty cycle measurement.
SIO	Software control of GPIO from the Single-cycle IO (SIO) block. The SIO function (F5) must be selected for the processors to drive a GPIO, but the input is always connected, so software can check the state of GPIOs at any time.
PIOx	Connect one of the programmable IO blocks (PIO) to GPIO. PIO can implement a wide variety of interfaces, and has its own internal pin mapping hardware, allowing flexible placement of digital interfaces on Bank 0 GPIOs. The PIO function (F6, F7, F8) must be selected for PIO to drive a GPIO, but the input is always connected, so the PIOs can always see the state of all pins.
HSTX	Connect the high-speed transmit peripheral (HSTX) to GPIO.
CLOCK GPINx	General purpose clock inputs. Can be routed to a number of internal clock domains on RP2350, e.g. to provide a 1Hz clock for the AON-Timer, or can be connected to an internal frequency counter.
CLOCK GPOUTx	General purpose clock outputs. Can drive a number of internal clocks (including PLL outputs) onto GPIOs, with optional integer divide.
TRACECLK, TRACEDATAx	CoreSight execution trace output from Cortex-M33 processors (Arm-only).
USB OVCUR DET/VBUS DET/VBUS EN	USB power control signals to/from the internal USB controller.
QMI CSxn	Auxiliary chip select for QSPI bus, to allow execute-in-place from an additional flash or PSRAM device.

Bank 1 function select operates identically to Bank 0, but its registers are in a different register block, starting with [USBPHY_DP_CTRL](#).

Table 647. GPIO Bank 1 Functions

Pin	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
USB DP			UART1 TX	I2C0 SDA		SIO						
USB DM			UART1 RX	I2C0 SCL		SIO						
QSPI SCK	QMI SCK		UART1 CTS	I2C1 SDA		SIO						UART1 TX
QSPI CSn	QMI CS0n		UART1 RTS	I2C1 SCL		SIO						UART1 RX
QSPI SD0	QMI SD0		UART0 TX	I2C0 SDA		SIO						
QSPI SD1	QMI SD1		UART0 RX	I2C0 SCL		SIO						
QSPI SD2	QMI SD2		UART0 CTS	I2C1 SDA		SIO						UART0 TX
QSPI SD3	QMI SD3		UART0 RTS	I2C1 SCL		SIO						UART0 RX

Table 648. GPIO bank 1 function descriptions

Function Name	Description
UARTx	Connect one of the internal PL011 UART peripherals to GPIO.
I2Cx	Connect one of the internal DW I2C peripherals to GPIO.

Function Name	Description
SIO	Software control of GPIO, from the single-cycle IO (SIO) block. The SIO function (F5) must be selected for the processors to drive a GPIO, but the input is always connected, so software can check the state of GPIOs at any time.
QMI	QSPI memory interface peripheral, used for execute-in-place from external QSPI flash or PSRAM memory devices.

The six QSPI Bank GPIO pins are typically used by the XIP peripheral to communicate with an external flash device. However, there are two scenarios where the pins can be used as software-controlled GPIOs:

- If a SPI or Dual-SPI flash device is used for execute-in-place, then the SD2 and SD3 pins are not used for flash access, and can be used for other GPIO functions on the circuit board.
- If RP2350 is used in a flashless configuration (USB and OTP boot only), then all six pins can be used for software-controlled GPIO functions.

9.5. Interrupts

An interrupt can be generated for every GPIO pin in four scenarios:

- Level High: the GPIO pin is a logical 1
- Level Low: the GPIO pin is a logical 0
- Edge High: the GPIO has transitioned from a logical 0 to a logical 1
- Edge Low: the GPIO has transitioned from a logical 1 to a logical 0

The level interrupts are not latched. This means that if the pin is a logical 1 and the level high interrupt is active, it will become inactive as soon as the pin changes to a logical 0. The edge interrupts are stored in the `INTR` register and can be cleared by writing to the `INTR` register.

There are enable, status, and force registers for three interrupt destinations: proc 0, proc 1, and dormant_wake. For proc 0 the registers are enable (`PROCO_INTE0`), status (`PROCO_INTS0`), and force (`PROCO_INTF0`). Dormant wake is used to wake the ROSC or XOSC up from dormant mode. See [Section 6.3.7.2](#) for more information on dormant mode.

There is an interrupt output for each combination of IO bank, IRQ destination, and security domain. In total there are twelve such outputs:

- IO Bank 0 to dormant wake (Secure and Non-secure)
- IO Bank 0 to proc 0 (Secure and Non-secure)
- IO Bank 0 to proc 1 (Secure and Non-secure)
- IO QSPI to dormant wake (Secure and Non-secure)
- IO QSPI to proc 0 (Secure and Non-secure)
- IO QSPI to proc 1 (Secure and Non-secure)

Each interrupt output has its own array of enable registers (INTE) which configures which GPIO events cause the interrupt to assert. The interrupt asserts when at least one enabled event occurs, and de-asserts when all enabled events have been acknowledged via the relevant INTR register.

This means the user can watch for several GPIO events at once.

Summary registers can be used to quickly check for pending GPIO interrupts. See `IRQSUMMARY_PROCO_NONSECURE0` for an example.

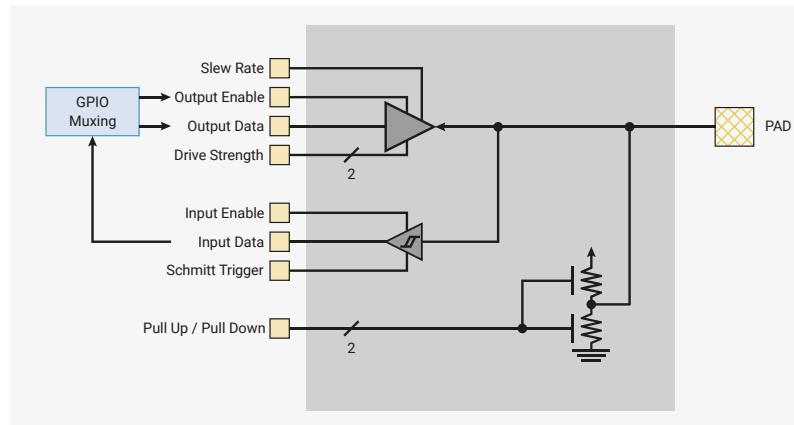
9.6. Pads

Each GPIO is connected to the off-chip world via a **pad**. Pads are the electrical interface between the chip's internal logic and external circuitry. They translate signal voltage levels, support higher currents and offer some protection against electrostatic discharge (ESD) events. You can adjust pad electrical behaviour to meet the requirements of external circuitry in the following ways:

- Output drive strength can be set to 2mA, 4mA, 8mA or 12mA.
- Output slew rate can be set to slow or fast.
- Input hysteresis (Schmitt trigger mode) can be enabled.
- A pull-up or pull-down can be enabled, to set the output signal level when the output driver is disabled.
- The input buffer can be disabled, to reduce current consumption when the pad is unused, unconnected or connected to an analogue signal.

An example pad is shown in [Figure 39](#).

Figure 39. Diagram of a single IO pad.



The pad's Output Enable, Output Data and Input Data ports connect, via the IO mux, to the function controlling the pad. All other ports are controlled from the pad control register. You can use this register to disable the pad's output driver by overriding the Output Enable signal from the function controlling the pad. See [GPIO0](#) for an example of a pad control register.

Both the output signal level and acceptable input signal level at the pad are determined by the digital IO supply (IOVDD). IOVDD can be any nominal voltage between 1.8V and 3.3V, but to meet specification when powered at 1.8V, the pad input thresholds must be adjusted by writing a 1 to the pad [VOLTAGE_SELECT](#) registers. By default, the pad input thresholds are valid for an IOVDD voltage between 2.5V and 3.3V. Using a voltage of 1.8V with the default input thresholds is a safe operating mode, but it will result in input thresholds that don't meet specification.

WARNING

Using IOVDD voltages greater than 1.8V, with the input thresholds set for 1.8V may result in damage to the chip.

Pad input threshold are adjusted on a per bank basis, with separate [VOLTAGE_SELECT](#) registers for the pads associated with the User IO bank (IO Bank 0) and the QSPI IO bank. However, both banks share the same digital IO supply (IOVDD), so both register should always be set to the same value.

Pad register details are available in [Section 9.11.3, "Pad Control - User Bank"](#) and [Section 9.11.4, "Pad Control - QSPI Bank"](#).

9.7. Pad Isolation Latches

RP2350 features extended low-power states which allow all internal logic, with the exception of POWMAN and some CoreSight debug logic, to fully power down under software control. This includes powering down all peripherals, the IO muxing, and the pad control registers, which brings with it the risk that pad signals may experience unwanted transitions when entering and exiting low-power states.

To ensure that pad states are well-defined at all times, all signals passing from the switched core power domain to the pads pass through **isolation latches**. In normal operation, the latches are transparent, so the pads are controlled fully by logic inside the switched core power domain, such as UARTs or the processors. However, when the ISO bit for each pad is set (e.g. `GPIO0.ISO`) or the switched core domain is powered down, the control signals currently presented to that pad are **latched** until the isolation is disabled. This includes the output enable state, output high/low level, and pull-up/pull-down resistor enable. The input signal from the pad back into the switched core domain is not isolated.

Consequently, when switched core logic is powered down, all Bank 0 and Bank 1 pads maintain the output state they held immediately before the power down, unless overridden by always-on logic in POWMAN. When the switched core power domain powers back up, all the GPIO ISO bits reset to 1, so the pre-power down state continues to be maintained until user software starts up and clears the ISO bit to indicate it is ready to use the pad again. Pads whose IO muxing has not yet been set up can be left isolated indefinitely, and will maintain their pre-power down state.

Once software has finished setting up the IO muxing for a given pad, and the peripheral which is to be muxed in, the ISO bit should be cleared. At this point the isolation latches will become transparent again: output signals passing through the IO muxing block are now reflected in the pad output state, so peripherals can communicate with the outside world. This process allows the switched core domain to be power cycled without causing any transitions on the pad outputs that may interfere with the operation of external hardware connected to the pads.

NOTE

Non-SDK applications ported from RP2040 must clear the ISO bit before using a GPIO, as this feature was not present on RP2040. The SDK automatically clears the ISO bit when `gpio_set_function()` is called.

The isolation latches themselves are reset by the always-on power domain reset, namely any one of:

- Power-on reset
- Brownout reset
- RUN pin being asserted low
- SW-DP CDBGRSTREQ
- RP-AP rescue reset

The latches reset to the reset value of the signal being isolated. For example, on Bank 0 GPIOs, the input enable control (`GPIO0.IE`) resets to 0 (input-disabled), so the isolation latches for these signals also take a reset value of 0. Resetting the isolation latch forces the pad to assume its reset state *even if it is currently isolated*.

The ISO control bits (e.g. `GPIO0.ISO`) are reset by the top-level switched core domain isolation signal, which is asserted by POWMAN before powering down the switched core domain and deasserted after it is powered up. This means that entering and exiting a sleep state where the switched core domain is unpowered leaves all GPIOs isolated after power up; you can then re-engage them individually. The ISO control bits are not reset by the PADS register block reset driven by the RESETS control registers: resetting the PADS register block returns non-isolated pads to their reset state, but has no effect on isolated pads.

9.8. Processor GPIO Controls (SIO)

The single-cycle IO subsystem (Section 3.1) contains memory-mapped GPIO registers. The processors can use these to perform input/output operations on GPIOs:

- The [GPIO_OUT](#) and [GPIO_HI_OUT](#) registers set the output level: 1 = high, 0 = low
- The [GPIO_OE](#) and [GPIO_HI_OE](#) registers set the output enable: 1 = output, 0 = input
- The [GPIO_IN](#) and [GPIO_HI_IN](#) registers read the GPIO inputs

These registers are all 32 bits in size. The low registers (e.g. [GPIO_OUT](#)) connect to GPIOs 0 through 31, and the high registers (e.g. [GPIO_HI_OUT](#)) connect to GPIOs 32 through 47, the QSPI pads, and the USB DM/DP pads.

For the output and output enable registers to take effect, the SIO function must be selected on each GPIO (function 5). However, the GPIO input registers read back the GPIO input values even when the SIO function is not selected, so the processor can always check the input state of any pin.

The SIO GPIO registers are shared between the two processors and between the Secure and Non-secure security domains. This avoids programming errors introduced by selecting multiple GPIO functions for access from different contexts.

Non-secure code's view of the SIO registers is restricted by the Non-secure GPIO mask defined in [GPIO_NSmask0](#) and [GPIO_NSmask1](#). Non-secure writes to Secure GPIOs are ignored. Non-secure reads of Secure GPIOs return 0.

These registers are documented in more detail in the SIO GPIO register section ([Section 3.1.3](#)).

The DMA cannot access registers in the SIO subsystem. The recommended method to DMA to GPIOs is a PIO program which continuously transfers TX FIFO data to the GPIO outputs, which provides more consistent timing than DMA directly into GPIO registers.

9.9. GPIO Coprocessor Port

Coprocessor port 0 on each Cortex-M33 processor connects to a GPIO coprocessor interface. These coprocessor instructions provide fast access to the SIO GPIO registers from Arm software:

- The equivalent of any SIO GPIO register access is a single instruction, without having to materialise a 32-bit register address beforehand
- An indexed write operation on any single GPIO is a single instruction
- 64 bits can be read/written in a single instruction

This reduces the timing impact of GPIO accesses on surrounding software, for example when GPIO tracing has been added to interrupt handlers diagnose complex timing issues.

Both Secure and Non-secure code may access the coprocessor. Non-secure code sees a restricted view of the GPIO registers, defined by ACCESSCTRL [GPIO_NSmask0/1](#).

The GPIO coprocessor instruction set is documented in [Section 3.6.1](#).

9.10. Software Examples

9.10.1. Select an IO function

An IO pin can perform many different functions and must be configured before use. For example, you may want it to be a [UART_TX](#) pin, or a [PWM](#) output. The SDK provides [gpio_set_function](#) for this purpose. Many SDK examples call [gpio_set_function](#) early on to enable printing to a UART.

The SDK starts by defining a structure to represent the registers of IO Bank 0, the User IO bank. Each IO has a status register, followed by a control register. For N IOs, the SDK instantiates the structure containing a status and control register as `io[N]` to repeat it N times.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2350/hardware_structs/include/hardware/structs/iobank0.h Lines 157 - 429

```

157 typedef struct {
158     iobank0_status_ctrl_hw_t io[NUM_BANK0_GPIOS]; // 48
159
160     uint32_t _pad0[32];
161
162     _REG_(IO_BANK0_IRQSUMMARY_PROC0_SECURE0_OFFSET) // IO_BANK0_IRQSUMMARY_PROC0_SECURE0
163     // (Description copied from array index 0 register IO_BANK0_IRQSUMMARY_PROC0_SECURE0
164     applies similarly to other array indexes)
165     //
166     // 0x80000000 [31] : GPIO31 (0)
167     // 0x40000000 [30] : GPIO30 (0)
168     // 0x20000000 [29] : GPIO29 (0)
169     // 0x10000000 [28] : GPIO28 (0)
170     // 0x08000000 [27] : GPIO27 (0)
171     // 0x04000000 [26] : GPIO26 (0)
172     // 0x02000000 [25] : GPIO25 (0)
173     // 0x01000000 [24] : GPIO24 (0)
174     // 0x00800000 [23] : GPIO23 (0)
175     // 0x00400000 [22] : GPIO22 (0)
176     // 0x00200000 [21] : GPIO21 (0)
177     // 0x00100000 [20] : GPIO20 (0)
178     // 0x00080000 [19] : GPIO19 (0)
179     // 0x00040000 [18] : GPIO18 (0)
180     // 0x00020000 [17] : GPIO17 (0)
181     // 0x00010000 [16] : GPIO16 (0)
182     // 0x00008000 [15] : GPIO15 (0)
183     // 0x00004000 [14] : GPIO14 (0)
184     // 0x00002000 [13] : GPIO13 (0)
185     // 0x00001000 [12] : GPIO12 (0)
186     // 0x00000800 [11] : GPIO11 (0)
187     // 0x00000400 [10] : GPIO10 (0)
188     // 0x00000200 [9] : GPIO9 (0)
189     // 0x00000100 [8] : GPIO8 (0)
190     // 0x00000080 [7] : GPIO7 (0)
191     // 0x00000040 [6] : GPIO6 (0)
192     // 0x00000020 [5] : GPIO5 (0)
193     // 0x00000010 [4] : GPIO4 (0)
194     // 0x00000008 [3] : GPIO3 (0)
195     // 0x00000004 [2] : GPIO2 (0)
196     // 0x00000002 [1] : GPIO1 (0)
197     // 0x00000001 [0] : GPIO0 (0)
198     io_ro_32 irqsummary_proc0_secure[2];
199
200     _REG_(IO_BANK0_IRQSUMMARY_PROC0_NONSECURE0_OFFSET) //
201     IO_BANK0_IRQSUMMARY_PROC0_NONSECURE0
202     // (Description copied from array index 0 register IO_BANK0_IRQSUMMARY_PROC0_NONSECURE0
203     applies similarly to other array indexes)
204     //
205     // 0x80000000 [31] : GPIO31 (0)
206     // 0x40000000 [30] : GPIO30 (0)
207     // 0x20000000 [29] : GPIO29 (0)
208     // 0x10000000 [28] : GPIO28 (0)
209     // 0x08000000 [27] : GPIO27 (0)
210     // 0x04000000 [26] : GPIO26 (0)
211     // 0x02000000 [25] : GPIO25 (0)
212     // 0x01000000 [24] : GPIO24 (0)
213     // 0x00800000 [23] : GPIO23 (0)
214     // 0x00400000 [22] : GPIO22 (0)
215     // 0x00200000 [21] : GPIO21 (0)
216     // 0x00100000 [20] : GPIO20 (0)
217     // 0x00080000 [19] : GPIO19 (0)

```

```

215  // 0x00004000 [18]    : GPIO18 (0)
216  // 0x00002000 [17]    : GPIO17 (0)
217  // 0x00010000 [16]    : GPIO16 (0)
218  // 0x00008000 [15]    : GPIO15 (0)
219  // 0x00004000 [14]    : GPIO14 (0)
220  // 0x00002000 [13]    : GPIO13 (0)
221  // 0x00001000 [12]    : GPIO12 (0)
222  // 0x00000800 [11]    : GPIO11 (0)
223  // 0x00000400 [10]    : GPIO10 (0)
224  // 0x00000200 [9]     : GPIO9 (0)
225  // 0x00000100 [8]     : GPIO8 (0)
226  // 0x00000080 [7]     : GPIO7 (0)
227  // 0x00000040 [6]     : GPIO6 (0)
228  // 0x00000020 [5]     : GPIO5 (0)
229  // 0x00000010 [4]     : GPIO4 (0)
230  // 0x00000008 [3]     : GPIO3 (0)
231  // 0x00000004 [2]     : GPIO2 (0)
232  // 0x00000002 [1]     : GPIO1 (0)
233  // 0x00000001 [0]     : GPIO0 (0)
234  io_ro_32 irqsummary_proc0_nonsecure[2];
235
236  _REG_(IO_BANK0_IRQSUMMARY_PROC1_SECURE0_OFFSET) // IO_BANK0_IRQSUMMARY_PROC1_SECURE0
237  // (Description copied from array index 0 register IO_BANK0_IRQSUMMARY_PROC1_SECURE0
238  // applies similarly to other array indexes)
239  //
240  // 0x80000000 [31]    : GPIO31 (0)
241  // 0x40000000 [30]    : GPIO30 (0)
242  // 0x20000000 [29]    : GPIO29 (0)
243  // 0x10000000 [28]    : GPIO28 (0)
244  // 0x08000000 [27]    : GPIO27 (0)
245  // 0x04000000 [26]    : GPIO26 (0)
246  // 0x02000000 [25]    : GPIO25 (0)
247  // 0x01000000 [24]    : GPIO24 (0)
248  // 0x00800000 [23]    : GPIO23 (0)
249  // 0x00400000 [22]    : GPIO22 (0)
250  // 0x00200000 [21]    : GPIO21 (0)
251  // 0x00100000 [20]    : GPIO20 (0)
252  // 0x00080000 [19]    : GPIO19 (0)
253  // 0x00040000 [18]    : GPIO18 (0)
254  // 0x00020000 [17]    : GPIO17 (0)
255  // 0x00010000 [16]    : GPIO16 (0)
256  // 0x00008000 [15]    : GPIO15 (0)
257  // 0x00004000 [14]    : GPIO14 (0)
258  // 0x00002000 [13]    : GPIO13 (0)
259  // 0x00001000 [12]    : GPIO12 (0)
260  // 0x00000800 [11]    : GPIO11 (0)
261  // 0x00000400 [10]    : GPIO10 (0)
262  // 0x00000200 [9]     : GPIO9 (0)
263  // 0x00000100 [8]     : GPIO8 (0)
264  // 0x00000080 [7]     : GPIO7 (0)
265  // 0x00000040 [6]     : GPIO6 (0)
266  // 0x00000020 [5]     : GPIO5 (0)
267  // 0x00000010 [4]     : GPIO4 (0)
268  // 0x00000008 [3]     : GPIO3 (0)
269  // 0x00000004 [2]     : GPIO2 (0)
270  // 0x00000002 [1]     : GPIO1 (0)
271  // 0x00000001 [0]     : GPIO0 (0)
272  io_ro_32 irqsummary_proc1_secure[2];
273
274  _REG_(IO_BANK0_IRQSUMMARY_PROC1_NONSECURE0_OFFSET) // IO_BANK0_IRQSUMMARY_PROC1_NONSECURE0
275  // (Description copied from array index 0 register IO_BANK0_IRQSUMMARY_PROC1_NONSECURE0
276  // applies similarly to other array indexes)
277  //

```

```

276 // 0x80000000 [31] : GPIO31 (0)
277 // 0x40000000 [30] : GPIO30 (0)
278 // 0x20000000 [29] : GPIO29 (0)
279 // 0x10000000 [28] : GPIO28 (0)
280 // 0x08000000 [27] : GPIO27 (0)
281 // 0x04000000 [26] : GPIO26 (0)
282 // 0x02000000 [25] : GPIO25 (0)
283 // 0x01000000 [24] : GPIO24 (0)
284 // 0x00800000 [23] : GPIO23 (0)
285 // 0x00400000 [22] : GPIO22 (0)
286 // 0x00200000 [21] : GPIO21 (0)
287 // 0x00100000 [20] : GPIO20 (0)
288 // 0x00080000 [19] : GPIO19 (0)
289 // 0x00040000 [18] : GPIO18 (0)
290 // 0x00020000 [17] : GPIO17 (0)
291 // 0x00010000 [16] : GPIO16 (0)
292 // 0x00008000 [15] : GPIO15 (0)
293 // 0x00004000 [14] : GPIO14 (0)
294 // 0x00002000 [13] : GPIO13 (0)
295 // 0x00001000 [12] : GPIO12 (0)
296 // 0x00000800 [11] : GPIO11 (0)
297 // 0x00000400 [10] : GPIO10 (0)
298 // 0x00000200 [9] : GPIO09 (0)
299 // 0x00000100 [8] : GPIO08 (0)
300 // 0x00000080 [7] : GPIO07 (0)
301 // 0x00000040 [6] : GPIO06 (0)
302 // 0x00000020 [5] : GPIO05 (0)
303 // 0x00000010 [4] : GPIO04 (0)
304 // 0x00000008 [3] : GPIO03 (0)
305 // 0x00000004 [2] : GPIO02 (0)
306 // 0x00000002 [1] : GPIO01 (0)
307 // 0x00000001 [0] : GPIO00 (0)
308 io_ro_32 irqsummary_proc1_nonscure[2];
309
310 _REG_(IO_BANK0_IRQSUMMARY_COMA_WAKE_SECURE0_OFFSET) //
    IO_BANK0_IRQSUMMARY_COMA_WAKE_SECURE0
311 // (Description copied from array index 0 register IO_BANK0_IRQSUMMARY_COMA_WAKE_SECURE0
    applies similarly to other array indexes)
312 //
313 // 0x80000000 [31] : GPIO31 (0)
314 // 0x40000000 [30] : GPIO30 (0)
315 // 0x20000000 [29] : GPIO29 (0)
316 // 0x10000000 [28] : GPIO28 (0)
317 // 0x08000000 [27] : GPIO27 (0)
318 // 0x04000000 [26] : GPIO26 (0)
319 // 0x02000000 [25] : GPIO25 (0)
320 // 0x01000000 [24] : GPIO24 (0)
321 // 0x00800000 [23] : GPIO23 (0)
322 // 0x00400000 [22] : GPIO22 (0)
323 // 0x00200000 [21] : GPIO21 (0)
324 // 0x00100000 [20] : GPIO20 (0)
325 // 0x00080000 [19] : GPIO19 (0)
326 // 0x00040000 [18] : GPIO18 (0)
327 // 0x00020000 [17] : GPIO17 (0)
328 // 0x00010000 [16] : GPIO16 (0)
329 // 0x00008000 [15] : GPIO15 (0)
330 // 0x00004000 [14] : GPIO14 (0)
331 // 0x00002000 [13] : GPIO13 (0)
332 // 0x00001000 [12] : GPIO12 (0)
333 // 0x00000800 [11] : GPIO11 (0)
334 // 0x00000400 [10] : GPIO10 (0)
335 // 0x00000200 [9] : GPIO09 (0)
336 // 0x00000100 [8] : GPIO08 (0)
337 // 0x00000080 [7] : GPIO07 (0)

```

```

338 // 0x00000040 [6] : GPIO6 (0)
339 // 0x00000020 [5] : GPIO5 (0)
340 // 0x00000010 [4] : GPIO4 (0)
341 // 0x00000008 [3] : GPIO3 (0)
342 // 0x00000004 [2] : GPIO2 (0)
343 // 0x00000002 [1] : GPIO1 (0)
344 // 0x00000001 [0] : GPIO0 (0)
345 io_ro_32_irqsummary_coma_wake_secure[2];
346
347 _REG_(IO_BANK0_IRQSUMMARY_COMA_WAKE_NONSECURE0_OFFSET) //  

    IO_BANK0_IRQSUMMARY_COMA_WAKE_NONSECURE0
348 // (Description copied from array index 0 register  

    IO_BANK0_IRQSUMMARY_COMA_WAKE_NONSECURE0 applies similarly to other array indexes)
349 //
350 // 0x80000000 [31] : GPIO31 (0)
351 // 0x40000000 [30] : GPIO30 (0)
352 // 0x20000000 [29] : GPIO29 (0)
353 // 0x10000000 [28] : GPIO28 (0)
354 // 0x08000000 [27] : GPIO27 (0)
355 // 0x04000000 [26] : GPIO26 (0)
356 // 0x02000000 [25] : GPIO25 (0)
357 // 0x01000000 [24] : GPIO24 (0)
358 // 0x00800000 [23] : GPIO23 (0)
359 // 0x00400000 [22] : GPIO22 (0)
360 // 0x00200000 [21] : GPIO21 (0)
361 // 0x00100000 [20] : GPIO20 (0)
362 // 0x00080000 [19] : GPIO19 (0)
363 // 0x00040000 [18] : GPIO18 (0)
364 // 0x00020000 [17] : GPIO17 (0)
365 // 0x00010000 [16] : GPIO16 (0)
366 // 0x00008000 [15] : GPIO15 (0)
367 // 0x00004000 [14] : GPIO14 (0)
368 // 0x00002000 [13] : GPIO13 (0)
369 // 0x00001000 [12] : GPIO12 (0)
370 // 0x00000800 [11] : GPIO11 (0)
371 // 0x00000400 [10] : GPIO10 (0)
372 // 0x00000200 [9] : GPIO9 (0)
373 // 0x00000100 [8] : GPIO8 (0)
374 // 0x00000080 [7] : GPIO7 (0)
375 // 0x00000040 [6] : GPIO6 (0)
376 // 0x00000020 [5] : GPIO5 (0)
377 // 0x00000010 [4] : GPIO4 (0)
378 // 0x00000008 [3] : GPIO3 (0)
379 // 0x00000004 [2] : GPIO2 (0)
380 // 0x00000002 [1] : GPIO1 (0)
381 // 0x00000001 [0] : GPIO0 (0)
382 io_ro_32_irqsummary_coma_wake_nonsecure[2];
383
384 _REG_(IO_BANK0_INTR0_OFFSET) // IO_BANK0_INTR0
385 // (Description copied from array index 0 register IO_BANK0_INTR0 applies similarly to  

    other array indexes)
386 //
387 // Raw Interrupts
388 // 0x80000000 [31] : GPIO7_EDGE_HIGH (0)
389 // 0x40000000 [30] : GPIO7_EDGE_LOW (0)
390 // 0x20000000 [29] : GPIO7_LEVEL_HIGH (0)
391 // 0x10000000 [28] : GPIO7_LEVEL_LOW (0)
392 // 0x08000000 [27] : GPIO6_EDGE_HIGH (0)
393 // 0x04000000 [26] : GPIO6_EDGE_LOW (0)
394 // 0x02000000 [25] : GPIO6_LEVEL_HIGH (0)
395 // 0x01000000 [24] : GPIO6_LEVEL_LOW (0)
396 // 0x00800000 [23] : GPIO5_EDGE_HIGH (0)
397 // 0x00400000 [22] : GPIO5_EDGE_LOW (0)
398 // 0x00200000 [21] : GPIO5_LEVEL_HIGH (0)

```

```

399 // 0x00100000 [20] : GPIO5_LEVEL_LOW (0)
400 // 0x00080000 [19] : GPIO4_EDGE_HIGH (0)
401 // 0x00040000 [18] : GPIO4_EDGE_LOW (0)
402 // 0x00020000 [17] : GPIO4_LEVEL_HIGH (0)
403 // 0x00010000 [16] : GPIO4_LEVEL_LOW (0)
404 // 0x00008000 [15] : GPIO3_EDGE_HIGH (0)
405 // 0x00004000 [14] : GPIO3_EDGE_LOW (0)
406 // 0x00002000 [13] : GPIO3_LEVEL_HIGH (0)
407 // 0x00001000 [12] : GPIO3_LEVEL_LOW (0)
408 // 0x00000800 [11] : GPIO2_EDGE_HIGH (0)
409 // 0x00000400 [10] : GPIO2_EDGE_LOW (0)
410 // 0x00000200 [9] : GPIO2_LEVEL_HIGH (0)
411 // 0x00000100 [8] : GPIO2_LEVEL_LOW (0)
412 // 0x00000080 [7] : GPIO1_EDGE_HIGH (0)
413 // 0x00000040 [6] : GPIO1_EDGE_LOW (0)
414 // 0x00000020 [5] : GPIO1_LEVEL_HIGH (0)
415 // 0x00000010 [4] : GPIO1_LEVEL_LOW (0)
416 // 0x00000008 [3] : GPIO0_EDGE_HIGH (0)
417 // 0x00000004 [2] : GPIO0_EDGE_LOW (0)
418 // 0x00000002 [1] : GPIO0_LEVEL_HIGH (0)
419 // 0x00000001 [0] : GPIO0_LEVEL_LOW (0)
420 io_rw_32 intr[6];
421
422 io_irq_ctrl_hw_t proc0_irq_ctrl;
423
424 io_irq_ctrl_hw_t proc1_irq_ctrl;
425
426 io_irq_ctrl_hw_t dormant_wake_irq_ctrl;
427 } iobank0_hw_t;
428
429 #define iobank0_hw ((iobank0_hw_t *)IO_BANK0_BASE)

```

A similar structure is defined for the pad control registers for IO bank 1. By default, all pads come out of reset ready to use, with input enabled and output disable set to 0. Regardless, `gpio_set_function` in the SDK sets the input enable and clears the output disable to engage the pad's IO buffers and connect internal signals to the outside world. Finally, the desired function select is written to the IO control register (see `GPIO0_CTRL` for an example of an IO control register).

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_gpio/gpio.c Lines 36 - 53

```

36 // Select function for this GPIO, and ensure input/output are enabled at the pad.
37 // This also clears the input/output/irq override bits.
38 void gpio_set_function(uint gpio, enum gpio_function fn) {
39     check_gpio_param(gpio);
40     invalid_params_if(HARDWARE_GPIO, ((uint32_t)fn << IO_BANK0_GPIO0_CTRL_FUNCSEL_LSB) &
41         ~IO_BANK0_GPIO0_CTRL_FUNCSEL_BITS);
42     // Set input enable on, output disable off
43     hw_write_masked(&padsbank0_hw->io[gpio],
44                     PADS_BANK0_GPIO0_IE_BITS,
45                     PADS_BANK0_GPIO0_IE_BITS | PADS_BANK0_GPIO0_OD_BITS
46     );
47     // Zero all fields apart from fsel; we want this IO to do what the peripheral tells it.
48     // This doesn't affect e.g. pullup/pulldown, as these are in pad controls.
49     iobank0_hw->io[gpio].ctrl = fn << IO_BANK0_GPIO0_CTRL_FUNCSEL_LSB;
50     // Remove pad isolation now that the correct peripheral is in control of the pad
51     hw_clear_bits(&padsbank0_hw->io[gpio], PADS_BANK0_GPIO0_ISO_BITS);
52 #endif
53 }

```

9.10.2. Enable a GPIO interrupt

The SDK provides a method of being interrupted when a GPIO pin changes state:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_gpio/gpio.c Lines 186 - 199

```

186 void gpio_set_irq_enabled(uint gpio, uint32_t events, bool enabled) {
187     // either this call disables the interrupt
188     // or callback should already be set (raw or using gpio_set_irq_callback)
189     // this protects against enabling the interrupt without callback set
190     assert(!enabled
191            || (raw_irq_mask[get_core_num()] & (1u<<gpio))
192            || callbacks[get_core_num()]);
193
194     // Separate mask/force/status per-core, so check which core called, and
195     // set the relevant IRQ controls.
196     io_irq_ctrl_hw_t *irq_ctrl_base = get_core_num() ?
197                                         &iobank0_hw->proc1_irq_ctrl : &iobank0_hw-
198                                         >proc0_irq_ctrl;
199     _gpio_set_irq_enabled(gpio, events, enabled, irq_ctrl_base);
200 }
```

`gpio_set_irq_enabled` uses a lower level function `_gpio_set_irq_enabled`:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_gpio/gpio.c Lines 173 - 184

```

173 static void _gpio_set_irq_enabled(uint gpio, uint32_t events, bool enabled, io_irq_ctrl_hw_t
174 *irq_ctrl_base) {
175     // Clear stale events which might cause immediate spurious handler entry
176     gpio_acknowledge_irq(gpio, events);
177
178     io_rw_32 *en_reg = &irq_ctrl_base->inte[gpio / 8];
179     events <= 4 * (gpio % 8);
180
181     if (enabled)
182         hw_set_bits(en_reg, events);
183     else
184         hw_clear_bits(en_reg, events);
185 }
```

The user provides a pointer to a callback function that is called when the GPIO event happens. An example application that uses this system is `hello_gpio_irq`:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/gpio/hello_gpio_irq/hello_gpio_irq.c

```

1 /**
2  * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
3  *
4  * SPDX-License-Identifier: BSD-3-Clause
5  */
6
7 #include <stdio.h>
8 #include "pico/stl.h"
9 #include "hardware/gpio.h"
10
11 static char event_str[128];
12
13 void gpio_event_string(char *buf, uint32_t events);
14
15 void gpio_callback(uint gpio, uint32_t events) {
```

```

16  // Put the GPIO event(s) that just happened into event_str
17  // so we can print it
18  gpio_event_string(event_str, events);
19  printf("GPIO %d %s\n", gpio, event_str);
20 }
21
22 int main() {
23     stdio_init_all();
24
25     printf("Hello GPIO IRQ\n");
26     gpio_set_irq_enabled_with_callback(2, GPIO_IRQ_EDGE_RISE | GPIO_IRQ_EDGE_FALL, true,
27                                     &gpio_callback);
28
29     // Wait forever
30     while (1);
31
32
33 static const char *gpio_irq_str[] = {
34     "LEVEL_LOW", // 0x1
35     "LEVEL_HIGH", // 0x2
36     "EDGE_FALL", // 0x4
37     "EDGE_RISE" // 0x8
38 };
39
40 void gpio_event_string(char *buf, uint32_t events) {
41     for (uint i = 0; i < 4; i++) {
42         uint mask = (1 << i);
43         if (events & mask) {
44             // Copy this event string into the user string
45             const char *event_str = gpio_irq_str[i];
46             while (*event_str != '\0') {
47                 *buf++ = *event_str++;
48             }
49             events &= ~mask;
50
51             // If more events add ","
52             if (events) {
53                 *buf++ = ',';
54                 *buf++ = ' ';
55             }
56         }
57     }
58     *buf++ = '\0';
59 }

```

9.11. List of Registers

9.11.1. IO - User Bank

The User Bank IO registers start at a base address of **0x40028000** (defined as **IO_BANK0_BASE** in SDK).

Table 649. List of IO_BANK0 registers

Offset	Name	Info
0x000	GPIO0_STATUS	
0x004	GPIO0_CTRL	
0x008	GPIO1_STATUS	

Offset	Name	Info
0x00c	GPIO1_CTRL	
0x010	GPIO2_STATUS	
0x014	GPIO2_CTRL	
0x018	GPIO3_STATUS	
0x01c	GPIO3_CTRL	
0x020	GPIO4_STATUS	
0x024	GPIO4_CTRL	
0x028	GPIO5_STATUS	
0x02c	GPIO5_CTRL	
0x030	GPIO6_STATUS	
0x034	GPIO6_CTRL	
0x038	GPIO7_STATUS	
0x03c	GPIO7_CTRL	
0x040	GPIO8_STATUS	
0x044	GPIO8_CTRL	
0x048	GPIO9_STATUS	
0x04c	GPIO9_CTRL	
0x050	GPIO10_STATUS	
0x054	GPIO10_CTRL	
0x058	GPIO11_STATUS	
0x05c	GPIO11_CTRL	
0x060	GPIO12_STATUS	
0x064	GPIO12_CTRL	
0x068	GPIO13_STATUS	
0x06c	GPIO13_CTRL	
0x070	GPIO14_STATUS	
0x074	GPIO14_CTRL	
0x078	GPIO15_STATUS	
0x07c	GPIO15_CTRL	
0x080	GPIO16_STATUS	
0x084	GPIO16_CTRL	
0x088	GPIO17_STATUS	
0x08c	GPIO17_CTRL	
0x090	GPIO18_STATUS	
0x094	GPIO18_CTRL	
0x098	GPIO19_STATUS	

Offset	Name	Info
0x09c	GPIO19_CTRL	
0x0a0	GPIO20_STATUS	
0x0a4	GPIO20_CTRL	
0x0a8	GPIO21_STATUS	
0x0ac	GPIO21_CTRL	
0x0b0	GPIO22_STATUS	
0x0b4	GPIO22_CTRL	
0x0b8	GPIO23_STATUS	
0x0bc	GPIO23_CTRL	
0x0c0	GPIO24_STATUS	
0x0c4	GPIO24_CTRL	
0x0c8	GPIO25_STATUS	
0x0cc	GPIO25_CTRL	
0x0d0	GPIO26_STATUS	
0x0d4	GPIO26_CTRL	
0x0d8	GPIO27_STATUS	
0x0dc	GPIO27_CTRL	
0x0e0	GPIO28_STATUS	
0x0e4	GPIO28_CTRL	
0x0e8	GPIO29_STATUS	
0x0ec	GPIO29_CTRL	
0x0f0	GPIO30_STATUS	
0x0f4	GPIO30_CTRL	
0x0f8	GPIO31_STATUS	
0x0fc	GPIO31_CTRL	
0x100	GPIO32_STATUS	
0x104	GPIO32_CTRL	
0x108	GPIO33_STATUS	
0x10c	GPIO33_CTRL	
0x110	GPIO34_STATUS	
0x114	GPIO34_CTRL	
0x118	GPIO35_STATUS	
0x11c	GPIO35_CTRL	
0x120	GPIO36_STATUS	
0x124	GPIO36_CTRL	
0x128	GPIO37_STATUS	

Offset	Name	Info
0x12c	GPIO37_CTRL	
0x130	GPIO38_STATUS	
0x134	GPIO38_CTRL	
0x138	GPIO39_STATUS	
0x13c	GPIO39_CTRL	
0x140	GPIO40_STATUS	
0x144	GPIO40_CTRL	
0x148	GPIO41_STATUS	
0x14c	GPIO41_CTRL	
0x150	GPIO42_STATUS	
0x154	GPIO42_CTRL	
0x158	GPIO43_STATUS	
0x15c	GPIO43_CTRL	
0x160	GPIO44_STATUS	
0x164	GPIO44_CTRL	
0x168	GPIO45_STATUS	
0x16c	GPIO45_CTRL	
0x170	GPIO46_STATUS	
0x174	GPIO46_CTRL	
0x178	GPIO47_STATUS	
0x17c	GPIO47_CTRL	
0x200	IRQSUMMARY_PROC0_SECURE0	
0x204	IRQSUMMARY_PROC0_SECURE1	
0x208	IRQSUMMARY_PROC0_NONSECURE0	
0x20c	IRQSUMMARY_PROC0_NONSECURE1	
0x210	IRQSUMMARY_PROC1_SECURE0	
0x214	IRQSUMMARY_PROC1_SECURE1	
0x218	IRQSUMMARY_PROC1_NONSECURE0	
0x21c	IRQSUMMARY_PROC1_NONSECURE1	
0x220	IRQSUMMARY_COMA_WAKE_SECURE0	
0x224	IRQSUMMARY_COMA_WAKE_SECURE1	
0x228	IRQSUMMARY_COMA_WAKE_NONSECURE0	
0x22c	IRQSUMMARY_COMA_WAKE_NONSECURE1	

Offset	Name	Info
0x230	INTR0	Raw Interrupts
0x234	INTR1	Raw Interrupts
0x238	INTR2	Raw Interrupts
0x23c	INTR3	Raw Interrupts
0x240	INTR4	Raw Interrupts
0x244	INTR5	Raw Interrupts
0x248	PROC0_INTE0	Interrupt Enable for proc0
0x24c	PROC0_INTE1	Interrupt Enable for proc0
0x250	PROC0_INTE2	Interrupt Enable for proc0
0x254	PROC0_INTE3	Interrupt Enable for proc0
0x258	PROC0_INTE4	Interrupt Enable for proc0
0x25c	PROC0_INTE5	Interrupt Enable for proc0
0x260	PROC0_INTF0	Interrupt Force for proc0
0x264	PROC0_INTF1	Interrupt Force for proc0
0x268	PROC0_INTF2	Interrupt Force for proc0
0x26c	PROC0_INTF3	Interrupt Force for proc0
0x270	PROC0_INTF4	Interrupt Force for proc0
0x274	PROC0_INTF5	Interrupt Force for proc0
0x278	PROC0_INTS0	Interrupt status after masking & forcing for proc0
0x27c	PROC0_INTS1	Interrupt status after masking & forcing for proc0
0x280	PROC0_INTS2	Interrupt status after masking & forcing for proc0
0x284	PROC0_INTS3	Interrupt status after masking & forcing for proc0
0x288	PROC0_INTS4	Interrupt status after masking & forcing for proc0
0x28c	PROC0_INTS5	Interrupt status after masking & forcing for proc0
0x290	PROC1_INTE0	Interrupt Enable for proc1
0x294	PROC1_INTE1	Interrupt Enable for proc1
0x298	PROC1_INTE2	Interrupt Enable for proc1
0x29c	PROC1_INTE3	Interrupt Enable for proc1
0x2a0	PROC1_INTE4	Interrupt Enable for proc1
0x2a4	PROC1_INTE5	Interrupt Enable for proc1
0x2a8	PROC1_INTF0	Interrupt Force for proc1
0x2ac	PROC1_INTF1	Interrupt Force for proc1
0x2b0	PROC1_INTF2	Interrupt Force for proc1
0x2b4	PROC1_INTF3	Interrupt Force for proc1
0x2b8	PROC1_INTF4	Interrupt Force for proc1
0x2bc	PROC1_INTF5	Interrupt Force for proc1

Offset	Name	Info
0x2c0	PROC1_INTS0	Interrupt status after masking & forcing for proc1
0x2c4	PROC1_INTS1	Interrupt status after masking & forcing for proc1
0x2c8	PROC1_INTS2	Interrupt status after masking & forcing for proc1
0x2cc	PROC1_INTS3	Interrupt status after masking & forcing for proc1
0x2d0	PROC1_INTS4	Interrupt status after masking & forcing for proc1
0x2d4	PROC1_INTS5	Interrupt status after masking & forcing for proc1
0x2d8	DORMANT_WAKE_INTE0	Interrupt Enable for dormant_wake
0x2dc	DORMANT_WAKE_INTE1	Interrupt Enable for dormant_wake
0x2e0	DORMANT_WAKE_INTE2	Interrupt Enable for dormant_wake
0x2e4	DORMANT_WAKE_INTE3	Interrupt Enable for dormant_wake
0x2e8	DORMANT_WAKE_INTE4	Interrupt Enable for dormant_wake
0x2ec	DORMANT_WAKE_INTE5	Interrupt Enable for dormant_wake
0x2f0	DORMANT_WAKE_INTF0	Interrupt Force for dormant_wake
0x2f4	DORMANT_WAKE_INTF1	Interrupt Force for dormant_wake
0x2f8	DORMANT_WAKE_INTF2	Interrupt Force for dormant_wake
0x2fc	DORMANT_WAKE_INTF3	Interrupt Force for dormant_wake
0x300	DORMANT_WAKE_INTF4	Interrupt Force for dormant_wake
0x304	DORMANT_WAKE_INTF5	Interrupt Force for dormant_wake
0x308	DORMANT_WAKE_INTS0	Interrupt status after masking & forcing for dormant_wake
0x30c	DORMANT_WAKE_INTS1	Interrupt status after masking & forcing for dormant_wake
0x310	DORMANT_WAKE_INTS2	Interrupt status after masking & forcing for dormant_wake
0x314	DORMANT_WAKE_INTS3	Interrupt status after masking & forcing for dormant_wake
0x318	DORMANT_WAKE_INTS4	Interrupt status after masking & forcing for dormant_wake
0x31c	DORMANT_WAKE_INTS5	Interrupt status after masking & forcing for dormant_wake

IO_BANK0: GPIO0_STATUS Register

Offset: 0x000

Table 650.
GPIO0_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO0_CTRL Register

Offset: 0x004

Table 651.
GPIO0_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → jtag_tck 0x01 → spi0_rx 0x02 → uart0_tx 0x03 → i2c0_sda 0x04 → pwm_a_0 0x05 → sio_0 0x06 → pio0_0 0x07 → pio1_0 0x08 → pio2_0 0x09 → xip_ss_n_1 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO1_STATUS Register

Offset: 0x008

Table 652.
GPIO1_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO1_CTRL Register

Offset: 0x00c

Table 653.
GPIO1_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → jtag_tms 0x01 → spi0_ss_n 0x02 → uart0_rx 0x03 → i2c0_scl 0x04 → pwm_b_0 0x05 → sio_1 0x06 → pio0_1 0x07 → pio1_1 0x08 → pio2_1 0x09 → coresight_traceclk 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO2_STATUS Register

Offset: 0x010

Table 654.
GPIO2_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO2_CTRL Register

Offset: 0x014

Table 655.
GPIO2_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → jtag_tdi 0x01 → spi0_sclk 0x02 → uart0_cts 0x03 → i2c1_sda 0x04 → pwm_a_1 0x05 → sio_2 0x06 → pio0_2 0x07 → pio1_2 0x08 → pio2_2 0x09 → coresight_tracedata_0 0x0a → usb_muxing_vbus_en 0x0b → uart0_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO3_STATUS Register

Offset: 0x018

Table 656.
GPIO3_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO3_CTRL Register

Offset: 0x01c

Table 657.
GPIO3_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → jtag_tdo 0x01 → spi0_tx 0x02 → uart0_rts 0x03 → i2c1_scl 0x04 → pwm_b_1 0x05 → sio_3 0x06 → pio0_3 0x07 → pio1_3 0x08 → pio2_3 0x09 → coresight_tracedata_1 0x0a → usb_muxing_overcurr_detect 0x0b → uart0_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO4_STATUS Register

Offset: 0x020

Table 658.
GPIO4_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0

Bits	Name	Description	Type	Reset
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO4_CTRL Register

Offset: 0x024

Table 659.
GPIO4_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_rx 0x02 → uart1_tx 0x03 → i2c0_sda 0x04 → pwm_a_2 0x05 → sio_4 0x06 → pio0_4 0x07 → pio1_4 0x08 → pio2_4 0x09 → coresight_tracedata_2 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO5_STATUS Register

Offset: 0x028

Table 660.
GPIO5_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO5_CTRL Register

Offset: 0x02c

Table 661.
GPIO5_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_ss_n 0x02 → uart1_rx 0x03 → i2c0_scl 0x04 → pwm_b_2 0x05 → sio_5 0x06 → pio0_5 0x07 → pio1_5 0x08 → pio2_5 0x09 → coresight_tracedata_3 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO6_STATUS Register

Offset: 0x030

Table 662.
GPIO6_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO6_CTRL Register

Offset: 0x034

Table 663.
GPIO6_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_sclk 0x02 → uart1_cts 0x03 → i2c1_sda 0x04 → pwm_a_3 0x05 → sio_6 0x06 → pio0_6 0x07 → pio1_6 0x08 → pio2_6 0x0a → usb_muxing_overcurr_detect 0x0b → uart1_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO7_STATUS Register

Offset: 0x038

Table 664.
GPIO7_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO7_CTRL Register

Offset: 0x03c

Table 665.
GPIO7_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_tx 0x02 → uart1_rts 0x03 → i2c1_scl 0x04 → pwm_b_3 0x05 → sio_7 0x06 → pio0_7 0x07 → pio1_7 0x08 → pio2_7 0x0a → usb_muxing_vbus_detect 0x0b → uart1_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO8_STATUS Register

Offset: 0x040

Table 666.
GPIO8_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO8_CTRL Register

Offset: 0x044

Table 667.
GPIO8_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_rx 0x02 → uart1_tx 0x03 → i2c0_sda 0x04 → pwm_a_4 0x05 → sio_8 0x06 → pio0_8 0x07 → pio1_8 0x08 → pio2_8 0x09 → xip_ss_n_1 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO9_STATUS Register

Offset: 0x048

Table 668.
GPIO9_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO9_CTRL Register

Offset: 0x04c

Table 669.
GPIO9_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_ss_n 0x02 → uart1_rx 0x03 → i2c0_scl 0x04 → pwm_b_4 0x05 → sio_9 0x06 → pio0_9 0x07 → pio1_9 0x08 → pio2_9 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO10_STATUS Register

Offset: 0x050

Table 670.
GPIO10_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO10_CTRL Register

Offset: 0x054

Table 671.
GPIO10_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_sclk 0x02 → uart1_cts 0x03 → i2c1_sda 0x04 → pwm_a_5 0x05 → sio_10 0x06 → pio0_10 0x07 → pio1_10 0x08 → pio2_10 0x0a → usb_muxing_vbus_detect 0x0b → uart1_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO11_STATUS Register

Offset: 0x058

Table 672.
GPIO11_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO11_CTRL Register

Offset: 0x05c

Table 673.
GPIO11_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_tx 0x02 → uart1_rts 0x03 → i2c1_scl 0x04 → pwm_b_5 0x05 → sio_11 0x06 → pio0_11 0x07 → pio1_11 0x08 → pio2_11 0x0a → usb_muxing_vbus_en 0x0b → uart1_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO12_STATUS Register

Offset: 0x060

Table 674.
GPIO12_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO12_CTRL Register

Offset: 0x064

Table 675.
GPIO12_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_0 0x01 → spi1_rx 0x02 → uart0_tx 0x03 → i2c0_sda 0x04 → pwm_a_6 0x05 → sio_12 0x06 → pio0_12 0x07 → pio1_12 0x08 → pio2_12 0x09 → clocks_gpin_0 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO13_STATUS Register

Offset: 0x068

Table 676.
GPIO13_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO13_CTRL Register

Offset: 0x06c

Table 677.
GPIO13_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_1 0x01 → spi1_ss_n 0x02 → uart0_rx 0x03 → i2c0_scl 0x04 → pwm_b_6 0x05 → sio_13 0x06 → pio0_13 0x07 → pio1_13 0x08 → pio2_13 0x09 → clocks_gpout_0 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO14_STATUS Register

Offset: 0x070

Table 678.
GPIO14_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO14_CTRL Register

Offset: 0x074

Table 679.
GPIO14_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_2 0x01 → spi1_sclk 0x02 → uart0_cts 0x03 → i2c1_sda 0x04 → pwm_a_7 0x05 → sio_14 0x06 → pio0_14 0x07 → pio1_14 0x08 → pio2_14 0x09 → clocks_gpin_1 0x0a → usb_muxing_vbus_en 0x0b → uart0_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO15_STATUS Register

Offset: 0x078

Table 680.
GPIO15_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO15_CTRL Register

Offset: 0x07c

Table 681.
GPIO15_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_3 0x01 → spi1_tx 0x02 → uart0_rts 0x03 → i2c1_scl 0x04 → pwm_b_7 0x05 → sio_15 0x06 → pio0_15 0x07 → pio1_15 0x08 → pio2_15 0x09 → clocks_gpout_1 0x0a → usb_muxing_overcurr_detect 0x0b → uart0_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO16_STATUS Register

Offset: 0x080

Table 682.
GPIO16_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0

Bits	Name	Description	Type	Reset
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO16_CTRL Register

Offset: 0x084

Table 683.
GPIO16_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_4 0x01 → spi0_rx 0x02 → uart0_tx 0x03 → i2c0_sda 0x04 → pwm_a_0 0x05 → sio_16 0x06 → pio0_16 0x07 → pio1_16 0x08 → pio2_16 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO17_STATUS Register

Offset: 0x088

Table 684.
GPIO17_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO17_CTRL Register

Offset: 0x08c

Table 685.
GPIO17_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_5 0x01 → spi0_ss_n 0x02 → uart0_rx 0x03 → i2c0_scl 0x04 → pwm_b_0 0x05 → sio_17 0x06 → pio0_17 0x07 → pio1_17 0x08 → pio2_17 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO18_STATUS Register

Offset: 0x090

Table 686.
GPIO18_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO18_CTRL Register

Offset: 0x094

Table 687.
GPIO18_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_6 0x01 → spi0_sclk 0x02 → uart0_cts 0x03 → i2c1_sda 0x04 → pwm_a_1 0x05 → sio_18 0x06 → pio0_18 0x07 → pio1_18 0x08 → pio2_18 0x0a → usb_muxing_overcurr_detect 0x0b → uart0_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO19_STATUS Register

Offset: 0x098

Table 688.
GPIO19_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO19_CTRL Register

Offset: 0x09c

Table 689.
GPIO19_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → hstx_7 0x01 → spi0_tx 0x02 → uart0_rts 0x03 → i2c1_scl 0x04 → pwm_b_1 0x05 → sio_19 0x06 → pio0_19 0x07 → pio1_19 0x08 → pio2_19 0x09 → xip_ss_n_1 0x0a → usb_muxing_vbus_detect 0x0b → uart0_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO20_STATUS Register

Offset: 0x0a0

Table 690.
GPIO20_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0

Bits	Name	Description	Type	Reset
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO20_CTRL Register

Offset: 0x0a4

Table 691.
GPIO20_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_rx 0x02 → uart1_tx 0x03 → i2c0_sda 0x04 → pwm_a_2 0x05 → sio_20 0x06 → pio0_20 0x07 → pio1_20 0x08 → pio2_20 0x09 → clocks_gpin_0 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO21_STATUS Register

Offset: 0x0a8

Table 692.
GPIO21_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO21_CTRL Register

Offset: 0x0ac

Table 693.
GPIO21_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_ss_n 0x02 → uart1_rx 0x03 → i2c0_scl 0x04 → pwm_b_2 0x05 → sio_21 0x06 → pio0_21 0x07 → pio1_21 0x08 → pio2_21 0x09 → clocks_gpout_0 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO22_STATUS Register

Offset: 0x0b0

Table 694.
GPIO22_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO22_CTRL Register

Offset: 0x0b4

Table 695.
GPIO22_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_sclk 0x02 → uart1_cts 0x03 → i2c1_sda 0x04 → pwm_a_3 0x05 → sio_22 0x06 → pio0_22 0x07 → pio1_22 0x08 → pio2_22 0x09 → clocks_gpin_1 0x0a → usb_muxing_vbus_detect 0x0b → uart1_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO23_STATUS Register

Offset: 0x0b8

Table 696.
GPIO23_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO23_CTRL Register

Offset: 0x0bc

Table 697.
GPIO23_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_tx 0x02 → uart1_rts 0x03 → i2c1_scl 0x04 → pwm_b_3 0x05 → sio_23 0x06 → pio0_23 0x07 → pio1_23 0x08 → pio2_23 0x09 → clocks_gpout_1 0x0a → usb_muxing_vbus_en 0x0b → uart1_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO24_STATUS Register

Offset: 0x0c0

Table 698.
GPIO24_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0

Bits	Name	Description	Type	Reset
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO24_CTRL Register

Offset: 0x0c4

Table 699.
GPIO24_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_rx 0x02 → uart1_tx 0x03 → i2c0_sda 0x04 → pwm_a_4 0x05 → sio_24 0x06 → pio0_24 0x07 → pio1_24 0x08 → pio2_24 0x09 → clocks_gpout_2 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO25_STATUS Register

Offset: 0x0c8

Table 700.
GPIO25_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO25_CTRL Register

Offset: 0x0cc

Table 701.
GPIO25_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_ss_n 0x02 → uart1_rx 0x03 → i2c0_scl 0x04 → pwm_b_4 0x05 → sio_25 0x06 → pio0_25 0x07 → pio1_25 0x08 → pio2_25 0x09 → clocks_gpout_3 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO26_STATUS Register

Offset: 0x0d0

Table 702.
GPIO26_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO26_CTRL Register

Offset: 0x0d4

Table 703.
GPIO26_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_sclk 0x02 → uart1_cts 0x03 → i2c1_sda 0x04 → pwm_a_5 0x05 → sio_26 0x06 → pio0_26 0x07 → pio1_26 0x08 → pio2_26 0x0a → usb_muxing_vbus_en 0x0b → uart1_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO27_STATUS Register

Offset: 0x0d8

Table 704.
GPIO27_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO27_CTRL Register

Offset: 0x0dc

Table 705.
GPIO27_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_tx 0x02 → uart1_rts 0x03 → i2c1_scl 0x04 → pwm_b_5 0x05 → sio_27 0x06 → pio0_27 0x07 → pio1_27 0x08 → pio2_27 0x0a → usb_muxing_overcurr_detect 0x0b → uart1_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO28_STATUS Register

Offset: 0x0e0

Table 706.
GPIO28_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO28_CTRL Register

Offset: 0x0e4

Table 707.
GPIO28_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_rx 0x02 → uart0_tx 0x03 → i2c0_sda 0x04 → pwm_a_6 0x05 → sio_28 0x06 → pio0_28 0x07 → pio1_28 0x08 → pio2_28 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO29_STATUS Register

Offset: 0x0e8

Table 708.
GPIO29_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO29_CTRL Register

Offset: 0x0ec

Table 709.
GPIO29_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_ss_n 0x02 → uart0_rx 0x03 → i2c0_scl 0x04 → pwm_b_6 0x05 → sio_29 0x06 → pio0_29 0x07 → pio1_29 0x08 → pio2_29 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO30_STATUS Register

Offset: 0x0f0

Table 710.
GPIO30_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO30_CTRL Register

Offset: 0x0f4

Table 711.
GPIO30_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_sclk 0x02 → uart0_cts 0x03 → i2c1_sda 0x04 → pwm_a_7 0x05 → sio_30 0x06 → pio0_30 0x07 → pio1_30 0x08 → pio2_30 0x0a → usb_muxing_overcurr_detect 0x0b → uart0_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO31_STATUS Register

Offset: 0x0f8

Table 712.
GPIO31_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO31_CTRL Register

Offset: 0x0fc

Table 713.
GPIO31_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_tx 0x02 → uart0_rts 0x03 → i2c1_scl 0x04 → pwm_b_7 0x05 → sio_31 0x06 → pio0_31 0x07 → pio1_31 0x08 → pio2_31 0x0a → usb_muxing_vbus_detect 0x0b → uart0_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO32_STATUS Register

Offset: 0x100

Table 714.
GPIO32_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO32_CTRL Register

Offset: 0x104

Table 715.
GPIO32_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_rx 0x02 → uart0_tx 0x03 → i2c0_sda 0x04 → pwm_a_8 0x05 → sio_32 0x06 → pio0_32 0x07 → pio1_32 0x08 → pio2_32 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO33_STATUS Register

Offset: 0x108

Table 716.
GPIO33_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO33_CTRL Register

Offset: 0x10c

Table 717.
GPIO33_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_ss_n 0x02 → uart0_rx 0x03 → i2c0_scl 0x04 → pwm_b_8 0x05 → sio_33 0x06 → pio0_33 0x07 → pio1_33 0x08 → pio2_33 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO34_STATUS Register

Offset: 0x110

Table 718.
GPIO34_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO34_CTRL Register

Offset: 0x114

Table 719.
GPIO34_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_sclk 0x02 → uart0_cts 0x03 → i2c1_sda 0x04 → pwm_a_9 0x05 → sio_34 0x06 → pio0_34 0x07 → pio1_34 0x08 → pio2_34 0x0a → usb_muxing_vbus_detect 0x0b → uart0_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO35_STATUS Register

Offset: 0x118

Table 720.
GPIO35_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO35_CTRL Register

Offset: 0x11c

Table 721.
GPIO35_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_tx 0x02 → uart0_rts 0x03 → i2c1_scl 0x04 → pwm_b_9 0x05 → sio_35 0x06 → pio0_35 0x07 → pio1_35 0x08 → pio2_35 0x0a → usb_muxing_vbus_en 0x0b → uart0_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO36_STATUS Register

Offset: 0x120

Table 722.
GPIO36_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO36_CTRL Register

Offset: 0x124

Table 723.
GPIO36_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_rx 0x02 → uart1_tx 0x03 → i2c0_sda 0x04 → pwm_a_10 0x05 → sio_36 0x06 → pio0_36 0x07 → pio1_36 0x08 → pio2_36 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO37_STATUS Register

Offset: 0x128

Table 724.
GPIO37_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO37_CTRL Register

Offset: 0x12c

Table 725.
GPIO37_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_ss_n 0x02 → uart1_rx 0x03 → i2c0_scl 0x04 → pwm_b_10 0x05 → sio_37 0x06 → pio0_37 0x07 → pio1_37 0x08 → pio2_37 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO38_STATUS Register

Offset: 0x130

Table 726.
GPIO38_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO38_CTRL Register

Offset: 0x134

Table 727.
GPIO38_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_sclk 0x02 → uart1_cts 0x03 → i2c1_sda 0x04 → pwm_a_11 0x05 → sio_38 0x06 → pio0_38 0x07 → pio1_38 0x08 → pio2_38 0x0a → usb_muxing_vbus_en 0x0b → uart1_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO39_STATUS Register

Offset: 0x138

Table 728.
GPIO39_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO39_CTRL Register

Offset: 0x13c

Table 729.
GPIO39_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi0_tx 0x02 → uart1_rts 0x03 → i2c1_scl 0x04 → pwm_b_11 0x05 → sio_39 0x06 → pio0_39 0x07 → pio1_39 0x08 → pio2_39 0x0a → usb_muxing_overcurr_detect 0x0b → uart1_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO40_STATUS Register

Offset: 0x140

Table 730.
GPIO40_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO40_CTRL Register

Offset: 0x144

Table 731.
GPIO40_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_rx 0x02 → uart1_tx 0x03 → i2c0_sda 0x04 → pwm_a_8 0x05 → sio_40 0x06 → pio0_40 0x07 → pio1_40 0x08 → pio2_40 0x0a → usb_muxing_vbus_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO41_STATUS Register

Offset: 0x148

Table 732.
GPIO41_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO41_CTRL Register

Offset: 0x14c

Table 733.
GPIO41_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_ss_n 0x02 → uart1_rx 0x03 → i2c0_scl 0x04 → pwm_b_8 0x05 → sio_41 0x06 → pio0_41 0x07 → pio1_41 0x08 → pio2_41 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO42_STATUS Register

Offset: 0x150

Table 734.
GPIO42_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO42_CTRL Register

Offset: 0x154

Table 735.
GPIO42_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_sclk 0x02 → uart1_cts 0x03 → i2c1_sda 0x04 → pwm_a_9 0x05 → sio_42 0x06 → pio0_42 0x07 → pio1_42 0x08 → pio2_42 0x0a → usb_muxing_overcurr_detect 0x0b → uart1_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO43_STATUS Register

Offset: 0x158

Table 736.
GPIO43_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO43_CTRL Register

Offset: 0x15c

Table 737.
GPIO43_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_tx 0x02 → uart1_rts 0x03 → i2c1_scl 0x04 → pwm_b_9 0x05 → sio_43 0x06 → pio0_43 0x07 → pio1_43 0x08 → pio2_43 0x0a → usb_muxing_vbus_detect 0x0b → uart1_rx 0x1f → null	RW	0x1f

IO_BANK0: GPIO44_STATUS Register

Offset: 0x160

Table 738.
GPIO44_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO44_CTRL Register

Offset: 0x164

Table 739.
GPIO44_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_rx 0x02 → uart0_tx 0x03 → i2c0_sda 0x04 → pwm_a_10 0x05 → sio_44 0x06 → pio0_44 0x07 → pio1_44 0x08 → pio2_44 0x0a → usb_muxing_vbus_en 0x1f → null	RW	0x1f

IO_BANK0: GPIO45_STATUS Register

Offset: 0x168

Table 740.
GPIO45_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO45_CTRL Register

Offset: 0x16c

Table 741.
GPIO45_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_ss_n 0x02 → uart0_rx 0x03 → i2c0_scl 0x04 → pwm_b_10 0x05 → sio_45 0x06 → pio0_45 0x07 → pio1_45 0x08 → pio2_45 0x0a → usb_muxing_overcurr_detect 0x1f → null	RW	0x1f

IO_BANK0: GPIO46_STATUS Register

Offset: 0x170

Table 742.
GPIO46_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO46_CTRL Register

Offset: 0x174

Table 743.
GPIO46_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_sclk 0x02 → uart0_cts 0x03 → i2c1_sda 0x04 → pwm_a_11 0x05 → sio_46 0x06 → pio0_46 0x07 → pio1_46 0x08 → pio2_46 0x0a → usb_muxing_vbus_detect 0x0b → uart0_tx 0x1f → null	RW	0x1f

IO_BANK0: GPIO47_STATUS Register

Offset: 0x178

Table 744.
GPIO47_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_BANK0: GPIO47_CTRL Register

Offset: 0x17c

Table 745.
GPIO47_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x01 → spi1_tx 0x02 → uart0_rts 0x03 → i2c1_scl 0x04 → pwm_b_11 0x05 → sio_47 0x06 → pio0_47 0x07 → pio1_47 0x08 → pio2_47 0x09 → xip_ss_n_1 0x0a → usb_muxing_vbus_en 0x0b → uart0_rx 0x1f → null	RW	0x1f

IO_BANK0: IRQSUMMARY_PROC0_SECURE0 Register

Offset: 0x200

Table 746.
IRQSUMMARY_PROC0_SECURE0 Register

Bits	Name	Description	Type	Reset
31	GPIO31		RO	0x0
30	GPIO30		RO	0x0
29	GPIO29		RO	0x0
28	GPIO28		RO	0x0
27	GPIO27		RO	0x0

Bits	Name	Description	Type	Reset
26	GPIO26		RO	0x0
25	GPIO25		RO	0x0
24	GPIO24		RO	0x0
23	GPIO23		RO	0x0
22	GPIO22		RO	0x0
21	GPIO21		RO	0x0
20	GPIO20		RO	0x0
19	GPIO19		RO	0x0
18	GPIO18		RO	0x0
17	GPIO17		RO	0x0
16	GPIO16		RO	0x0
15	GPIO15		RO	0x0
14	GPIO14		RO	0x0
13	GPIO13		RO	0x0
12	GPIO12		RO	0x0
11	GPIO11		RO	0x0
10	GPIO10		RO	0x0
9	GPIO9		RO	0x0
8	GPIO8		RO	0x0
7	GPIO7		RO	0x0
6	GPIO6		RO	0x0
5	GPIO5		RO	0x0
4	GPIO4		RO	0x0
3	GPIO3		RO	0x0
2	GPIO2		RO	0x0
1	GPIO1		RO	0x0
0	GPIO0		RO	0x0

IO_BANK0: IRQSUMMARY_PROCO_SECURE1 Register

Offset: 0x204

Table 747.
IRQSUMMARY_PROCO_SECURE1 Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	GPIO47		RO	0x0
14	GPIO46		RO	0x0
13	GPIO45		RO	0x0
12	GPIO44		RO	0x0

Bits	Name	Description	Type	Reset
11	GPIO43		RO	0x0
10	GPIO42		RO	0x0
9	GPIO41		RO	0x0
8	GPIO40		RO	0x0
7	GPIO39		RO	0x0
6	GPIO38		RO	0x0
5	GPIO37		RO	0x0
4	GPIO36		RO	0x0
3	GPIO35		RO	0x0
2	GPIO34		RO	0x0
1	GPIO33		RO	0x0
0	GPIO32		RO	0x0

IO_BANK0: IRQSUMMARY_PROC0_NONSECURE0 Register

Offset: 0x208

Table 748.
IRQSUMMARY_PROC0
_NONSECURE0
Register

Bits	Name	Description	Type	Reset
31	GPIO31		RO	0x0
30	GPIO30		RO	0x0
29	GPIO29		RO	0x0
28	GPIO28		RO	0x0
27	GPIO27		RO	0x0
26	GPIO26		RO	0x0
25	GPIO25		RO	0x0
24	GPIO24		RO	0x0
23	GPIO23		RO	0x0
22	GPIO22		RO	0x0
21	GPIO21		RO	0x0
20	GPIO20		RO	0x0
19	GPIO19		RO	0x0
18	GPIO18		RO	0x0
17	GPIO17		RO	0x0
16	GPIO16		RO	0x0
15	GPIO15		RO	0x0
14	GPIO14		RO	0x0
13	GPIO13		RO	0x0
12	GPIO12		RO	0x0

Bits	Name	Description	Type	Reset
11	GPIO11		RO	0x0
10	GPIO10		RO	0x0
9	GPIO9		RO	0x0
8	GPIO8		RO	0x0
7	GPIO7		RO	0x0
6	GPIO6		RO	0x0
5	GPIO5		RO	0x0
4	GPIO4		RO	0x0
3	GPIO3		RO	0x0
2	GPIO2		RO	0x0
1	GPIO1		RO	0x0
0	GPIO0		RO	0x0

IO_BANK0: IRQSUMMARY_PROC0_NONSECURE1 Register

Offset: 0x20c

Table 749.
IRQSUMMARY_PROC0
_NONSECURE1
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	GPIO47		RO	0x0
14	GPIO46		RO	0x0
13	GPIO45		RO	0x0
12	GPIO44		RO	0x0
11	GPIO43		RO	0x0
10	GPIO42		RO	0x0
9	GPIO41		RO	0x0
8	GPIO40		RO	0x0
7	GPIO39		RO	0x0
6	GPIO38		RO	0x0
5	GPIO37		RO	0x0
4	GPIO36		RO	0x0
3	GPIO35		RO	0x0
2	GPIO34		RO	0x0
1	GPIO33		RO	0x0
0	GPIO32		RO	0x0

IO_BANK0: IRQSUMMARY_PROC1_SECURE0 Register

Offset: 0x210

Table 750.
`IRQSUMMARY_PROC1_SECURE0 Register`

Bits	Name	Description	Type	Reset
31	GPIO31		RO	0x0
30	GPIO30		RO	0x0
29	GPIO29		RO	0x0
28	GPIO28		RO	0x0
27	GPIO27		RO	0x0
26	GPIO26		RO	0x0
25	GPIO25		RO	0x0
24	GPIO24		RO	0x0
23	GPIO23		RO	0x0
22	GPIO22		RO	0x0
21	GPIO21		RO	0x0
20	GPIO20		RO	0x0
19	GPIO19		RO	0x0
18	GPIO18		RO	0x0
17	GPIO17		RO	0x0
16	GPIO16		RO	0x0
15	GPIO15		RO	0x0
14	GPIO14		RO	0x0
13	GPIO13		RO	0x0
12	GPIO12		RO	0x0
11	GPIO11		RO	0x0
10	GPIO10		RO	0x0
9	GPIO9		RO	0x0
8	GPIO8		RO	0x0
7	GPIO7		RO	0x0
6	GPIO6		RO	0x0
5	GPIO5		RO	0x0
4	GPIO4		RO	0x0
3	GPIO3		RO	0x0
2	GPIO2		RO	0x0
1	GPIO1		RO	0x0
0	GPIO0		RO	0x0

IO_BANK0: IRQSUMMARY_PROC1_SECURE1 Register

Offset: 0x214

Table 751.
`IRQSUMMARY_PROC1`
`_SECURE1 Register`

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	GPIO47		RO	0x0
14	GPIO46		RO	0x0
13	GPIO45		RO	0x0
12	GPIO44		RO	0x0
11	GPIO43		RO	0x0
10	GPIO42		RO	0x0
9	GPIO41		RO	0x0
8	GPIO40		RO	0x0
7	GPIO39		RO	0x0
6	GPIO38		RO	0x0
5	GPIO37		RO	0x0
4	GPIO36		RO	0x0
3	GPIO35		RO	0x0
2	GPIO34		RO	0x0
1	GPIO33		RO	0x0
0	GPIO32		RO	0x0

IO_BANK0: IRQSUMMARY_PROC1_NONSECURE0 Register

Offset: 0x218

Table 752.
`IRQSUMMARY_PROC1`
`_NONSECURE0`
`Register`

Bits	Name	Description	Type	Reset
31	GPIO31		RO	0x0
30	GPIO30		RO	0x0
29	GPIO29		RO	0x0
28	GPIO28		RO	0x0
27	GPIO27		RO	0x0
26	GPIO26		RO	0x0
25	GPIO25		RO	0x0
24	GPIO24		RO	0x0
23	GPIO23		RO	0x0
22	GPIO22		RO	0x0
21	GPIO21		RO	0x0
20	GPIO20		RO	0x0
19	GPIO19		RO	0x0
18	GPIO18		RO	0x0
17	GPIO17		RO	0x0

Bits	Name	Description	Type	Reset
16	GPIO16		RO	0x0
15	GPIO15		RO	0x0
14	GPIO14		RO	0x0
13	GPIO13		RO	0x0
12	GPIO12		RO	0x0
11	GPIO11		RO	0x0
10	GPIO10		RO	0x0
9	GPIO9		RO	0x0
8	GPIO8		RO	0x0
7	GPIO7		RO	0x0
6	GPIO6		RO	0x0
5	GPIO5		RO	0x0
4	GPIO4		RO	0x0
3	GPIO3		RO	0x0
2	GPIO2		RO	0x0
1	GPIO1		RO	0x0
0	GPIO0		RO	0x0

IO_BANK0: IRQSUMMARY_PROC1_NONSECURE1 Register

Offset: 0x21c

Table 753.
IRQSUMMARY_PROC1
_NONSECURE1
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	GPIO47		RO	0x0
14	GPIO46		RO	0x0
13	GPIO45		RO	0x0
12	GPIO44		RO	0x0
11	GPIO43		RO	0x0
10	GPIO42		RO	0x0
9	GPIO41		RO	0x0
8	GPIO40		RO	0x0
7	GPIO39		RO	0x0
6	GPIO38		RO	0x0
5	GPIO37		RO	0x0
4	GPIO36		RO	0x0
3	GPIO35		RO	0x0
2	GPIO34		RO	0x0

Bits	Name	Description	Type	Reset
1	GPIO33		RO	0x0
0	GPIO32		RO	0x0

IO_BANK0: IRQSUMMARY_COMA_WAKE_SECURE0 Register

Offset: 0x220

Table 754.
IRQSUMMARY_COMA_WAKE_SECURE0 Register

Bits	Name	Description	Type	Reset
31	GPIO31		RO	0x0
30	GPIO30		RO	0x0
29	GPIO29		RO	0x0
28	GPIO28		RO	0x0
27	GPIO27		RO	0x0
26	GPIO26		RO	0x0
25	GPIO25		RO	0x0
24	GPIO24		RO	0x0
23	GPIO23		RO	0x0
22	GPIO22		RO	0x0
21	GPIO21		RO	0x0
20	GPIO20		RO	0x0
19	GPIO19		RO	0x0
18	GPIO18		RO	0x0
17	GPIO17		RO	0x0
16	GPIO16		RO	0x0
15	GPIO15		RO	0x0
14	GPIO14		RO	0x0
13	GPIO13		RO	0x0
12	GPIO12		RO	0x0
11	GPIO11		RO	0x0
10	GPIO10		RO	0x0
9	GPIO9		RO	0x0
8	GPIO8		RO	0x0
7	GPIO7		RO	0x0
6	GPIO6		RO	0x0
5	GPIO5		RO	0x0
4	GPIO4		RO	0x0
3	GPIO3		RO	0x0
2	GPIO2		RO	0x0

Bits	Name	Description	Type	Reset
1	GPIO1		RO	0x0
0	GPIO0		RO	0x0

IO_BANK0: IRQSUMMARY_COMA_WAKE_SECURE1 Register

Offset: 0x224

Table 755.
IRQSUMMARY_COMA_WAKE_SECURE1 Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	GPIO47		RO	0x0
14	GPIO46		RO	0x0
13	GPIO45		RO	0x0
12	GPIO44		RO	0x0
11	GPIO43		RO	0x0
10	GPIO42		RO	0x0
9	GPIO41		RO	0x0
8	GPIO40		RO	0x0
7	GPIO39		RO	0x0
6	GPIO38		RO	0x0
5	GPIO37		RO	0x0
4	GPIO36		RO	0x0
3	GPIO35		RO	0x0
2	GPIO34		RO	0x0
1	GPIO33		RO	0x0
0	GPIO32		RO	0x0

IO_BANK0: IRQSUMMARY_COMA_WAKE_NONSECURE0 Register

Offset: 0x228

Table 756.
IRQSUMMARY_COMA_WAKE_NONSECURE0 Register

Bits	Name	Description	Type	Reset
31	GPIO31		RO	0x0
30	GPIO30		RO	0x0
29	GPIO29		RO	0x0
28	GPIO28		RO	0x0
27	GPIO27		RO	0x0
26	GPIO26		RO	0x0
25	GPIO25		RO	0x0
24	GPIO24		RO	0x0
23	GPIO23		RO	0x0

Bits	Name	Description	Type	Reset
22	GPIO22		RO	0x0
21	GPIO21		RO	0x0
20	GPIO20		RO	0x0
19	GPIO19		RO	0x0
18	GPIO18		RO	0x0
17	GPIO17		RO	0x0
16	GPIO16		RO	0x0
15	GPIO15		RO	0x0
14	GPIO14		RO	0x0
13	GPIO13		RO	0x0
12	GPIO12		RO	0x0
11	GPIO11		RO	0x0
10	GPIO10		RO	0x0
9	GPIO9		RO	0x0
8	GPIO8		RO	0x0
7	GPIO7		RO	0x0
6	GPIO6		RO	0x0
5	GPIO5		RO	0x0
4	GPIO4		RO	0x0
3	GPIO3		RO	0x0
2	GPIO2		RO	0x0
1	GPIO1		RO	0x0
0	GPIO0		RO	0x0

IO_BANK0: IRQSUMMARY_COMA_WAKE_NONSECURE1 Register

Offset: 0x22c

Table 757.
IRQSUMMARY_COMA_WAKE_NONSECURE1 Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	GPIO47		RO	0x0
14	GPIO46		RO	0x0
13	GPIO45		RO	0x0
12	GPIO44		RO	0x0
11	GPIO43		RO	0x0
10	GPIO42		RO	0x0
9	GPIO41		RO	0x0
8	GPIO40		RO	0x0

Bits	Name	Description	Type	Reset
7	GPIO39		RO	0x0
6	GPIO38		RO	0x0
5	GPIO37		RO	0x0
4	GPIO36		RO	0x0
3	GPIO35		RO	0x0
2	GPIO34		RO	0x0
1	GPIO33		RO	0x0
0	GPIO32		RO	0x0

IO_BANK0: INTR0 Register

Offset: 0x230

Description

Raw Interrupts

Table 758. INTR0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		WC	0x0
30	GPIO7_EDGE_LOW		WC	0x0
29	GPIO7_LEVEL_HIGH		RO	0x0
28	GPIO7_LEVEL_LOW		RO	0x0
27	GPIO6_EDGE_HIGH		WC	0x0
26	GPIO6_EDGE_LOW		WC	0x0
25	GPIO6_LEVEL_HIGH		RO	0x0
24	GPIO6_LEVEL_LOW		RO	0x0
23	GPIO5_EDGE_HIGH		WC	0x0
22	GPIO5_EDGE_LOW		WC	0x0
21	GPIO5_LEVEL_HIGH		RO	0x0
20	GPIO5_LEVEL_LOW		RO	0x0
19	GPIO4_EDGE_HIGH		WC	0x0
18	GPIO4_EDGE_LOW		WC	0x0
17	GPIO4_LEVEL_HIGH		RO	0x0
16	GPIO4_LEVEL_LOW		RO	0x0
15	GPIO3_EDGE_HIGH		WC	0x0
14	GPIO3_EDGE_LOW		WC	0x0
13	GPIO3_LEVEL_HIGH		RO	0x0
12	GPIO3_LEVEL_LOW		RO	0x0
11	GPIO2_EDGE_HIGH		WC	0x0
10	GPIO2_EDGE_LOW		WC	0x0

Bits	Name	Description	Type	Reset
9	GPIO2_LEVEL_HIGH		RO	0x0
8	GPIO2_LEVEL_LOW		RO	0x0
7	GPIO1_EDGE_HIGH		WC	0x0
6	GPIO1_EDGE_LOW		WC	0x0
5	GPIO1_LEVEL_HIGH		RO	0x0
4	GPIO1_LEVEL_LOW		RO	0x0
3	GPIO0_EDGE_HIGH		WC	0x0
2	GPIO0_EDGE_LOW		WC	0x0
1	GPIO0_LEVEL_HIGH		RO	0x0
0	GPIO0_LEVEL_LOW		RO	0x0

IO_BANK0: INTR1 Register

Offset: 0x234

Description

Raw Interrupts

Table 759. INTR1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		WC	0x0
30	GPIO15_EDGE_LOW		WC	0x0
29	GPIO15_LEVEL_HIGH		RO	0x0
28	GPIO15_LEVEL_LOW		RO	0x0
27	GPIO14_EDGE_HIGH		WC	0x0
26	GPIO14_EDGE_LOW		WC	0x0
25	GPIO14_LEVEL_HIGH		RO	0x0
24	GPIO14_LEVEL_LOW		RO	0x0
23	GPIO13_EDGE_HIGH		WC	0x0
22	GPIO13_EDGE_LOW		WC	0x0
21	GPIO13_LEVEL_HIGH		RO	0x0
20	GPIO13_LEVEL_LOW		RO	0x0
19	GPIO12_EDGE_HIGH		WC	0x0
18	GPIO12_EDGE_LOW		WC	0x0
17	GPIO12_LEVEL_HIGH		RO	0x0
16	GPIO12_LEVEL_LOW		RO	0x0
15	GPIO11_EDGE_HIGH		WC	0x0
14	GPIO11_EDGE_LOW		WC	0x0
13	GPIO11_LEVEL_HIGH		RO	0x0
12	GPIO11_LEVEL_LOW		RO	0x0

Bits	Name	Description	Type	Reset
11	GPIO10_EDGE_HIGH		WC	0x0
10	GPIO10_EDGE_LOW		WC	0x0
9	GPIO10_LEVEL_HIGH		RO	0x0
8	GPIO10_LEVEL_LOW		RO	0x0
7	GPIO9_EDGE_HIGH		WC	0x0
6	GPIO9_EDGE_LOW		WC	0x0
5	GPIO9_LEVEL_HIGH		RO	0x0
4	GPIO9_LEVEL_LOW		RO	0x0
3	GPIO8_EDGE_HIGH		WC	0x0
2	GPIO8_EDGE_LOW		WC	0x0
1	GPIO8_LEVEL_HIGH		RO	0x0
0	GPIO8_LEVEL_LOW		RO	0x0

IO_BANK0: INTR2 Register

Offset: 0x238

Description

Raw Interrupts

Table 760. INTR2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		WC	0x0
30	GPIO23_EDGE_LOW		WC	0x0
29	GPIO23_LEVEL_HIGH		RO	0x0
28	GPIO23_LEVEL_LOW		RO	0x0
27	GPIO22_EDGE_HIGH		WC	0x0
26	GPIO22_EDGE_LOW		WC	0x0
25	GPIO22_LEVEL_HIGH		RO	0x0
24	GPIO22_LEVEL_LOW		RO	0x0
23	GPIO21_EDGE_HIGH		WC	0x0
22	GPIO21_EDGE_LOW		WC	0x0
21	GPIO21_LEVEL_HIGH		RO	0x0
20	GPIO21_LEVEL_LOW		RO	0x0
19	GPIO20_EDGE_HIGH		WC	0x0
18	GPIO20_EDGE_LOW		WC	0x0
17	GPIO20_LEVEL_HIGH		RO	0x0
16	GPIO20_LEVEL_LOW		RO	0x0
15	GPIO19_EDGE_HIGH		WC	0x0
14	GPIO19_EDGE_LOW		WC	0x0

Bits	Name	Description	Type	Reset
13	GPIO19_LEVEL_HIGH		RO	0x0
12	GPIO19_LEVEL_LOW		RO	0x0
11	GPIO18_EDGE_HIGH		WC	0x0
10	GPIO18_EDGE_LOW		WC	0x0
9	GPIO18_LEVEL_HIGH		RO	0x0
8	GPIO18_LEVEL_LOW		RO	0x0
7	GPIO17_EDGE_HIGH		WC	0x0
6	GPIO17_EDGE_LOW		WC	0x0
5	GPIO17_LEVEL_HIGH		RO	0x0
4	GPIO17_LEVEL_LOW		RO	0x0
3	GPIO16_EDGE_HIGH		WC	0x0
2	GPIO16_EDGE_LOW		WC	0x0
1	GPIO16_LEVEL_HIGH		RO	0x0
0	GPIO16_LEVEL_LOW		RO	0x0

IO_BANK0: INTR3 Register

Offset: 0x23c

Description

Raw Interrupts

Table 761. INTR3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		WC	0x0
30	GPIO31_EDGE_LOW		WC	0x0
29	GPIO31_LEVEL_HIGH		RO	0x0
28	GPIO31_LEVEL_LOW		RO	0x0
27	GPIO30_EDGE_HIGH		WC	0x0
26	GPIO30_EDGE_LOW		WC	0x0
25	GPIO30_LEVEL_HIGH		RO	0x0
24	GPIO30_LEVEL_LOW		RO	0x0
23	GPIO29_EDGE_HIGH		WC	0x0
22	GPIO29_EDGE_LOW		WC	0x0
21	GPIO29_LEVEL_HIGH		RO	0x0
20	GPIO29_LEVEL_LOW		RO	0x0
19	GPIO28_EDGE_HIGH		WC	0x0
18	GPIO28_EDGE_LOW		WC	0x0
17	GPIO28_LEVEL_HIGH		RO	0x0
16	GPIO28_LEVEL_LOW		RO	0x0

Bits	Name	Description	Type	Reset
15	GPIO27_EDGE_HIGH		WC	0x0
14	GPIO27_EDGE_LOW		WC	0x0
13	GPIO27_LEVEL_HIGH		RO	0x0
12	GPIO27_LEVEL_LOW		RO	0x0
11	GPIO26_EDGE_HIGH		WC	0x0
10	GPIO26_EDGE_LOW		WC	0x0
9	GPIO26_LEVEL_HIGH		RO	0x0
8	GPIO26_LEVEL_LOW		RO	0x0
7	GPIO25_EDGE_HIGH		WC	0x0
6	GPIO25_EDGE_LOW		WC	0x0
5	GPIO25_LEVEL_HIGH		RO	0x0
4	GPIO25_LEVEL_LOW		RO	0x0
3	GPIO24_EDGE_HIGH		WC	0x0
2	GPIO24_EDGE_LOW		WC	0x0
1	GPIO24_LEVEL_HIGH		RO	0x0
0	GPIO24_LEVEL_LOW		RO	0x0

IO_BANK0: INTR4 Register

Offset: 0x240

Description

Raw Interrupts

Table 762. INTR4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		WC	0x0
30	GPIO39_EDGE_LOW		WC	0x0
29	GPIO39_LEVEL_HIGH		RO	0x0
28	GPIO39_LEVEL_LOW		RO	0x0
27	GPIO38_EDGE_HIGH		WC	0x0
26	GPIO38_EDGE_LOW		WC	0x0
25	GPIO38_LEVEL_HIGH		RO	0x0
24	GPIO38_LEVEL_LOW		RO	0x0
23	GPIO37_EDGE_HIGH		WC	0x0
22	GPIO37_EDGE_LOW		WC	0x0
21	GPIO37_LEVEL_HIGH		RO	0x0
20	GPIO37_LEVEL_LOW		RO	0x0
19	GPIO36_EDGE_HIGH		WC	0x0
18	GPIO36_EDGE_LOW		WC	0x0

Bits	Name	Description	Type	Reset
17	GPIO36_LEVEL_HIGH		RO	0x0
16	GPIO36_LEVEL_LOW		RO	0x0
15	GPIO35_EDGE_HIGH		WC	0x0
14	GPIO35_EDGE_LOW		WC	0x0
13	GPIO35_LEVEL_HIGH		RO	0x0
12	GPIO35_LEVEL_LOW		RO	0x0
11	GPIO34_EDGE_HIGH		WC	0x0
10	GPIO34_EDGE_LOW		WC	0x0
9	GPIO34_LEVEL_HIGH		RO	0x0
8	GPIO34_LEVEL_LOW		RO	0x0
7	GPIO33_EDGE_HIGH		WC	0x0
6	GPIO33_EDGE_LOW		WC	0x0
5	GPIO33_LEVEL_HIGH		RO	0x0
4	GPIO33_LEVEL_LOW		RO	0x0
3	GPIO32_EDGE_HIGH		WC	0x0
2	GPIO32_EDGE_LOW		WC	0x0
1	GPIO32_LEVEL_HIGH		RO	0x0
0	GPIO32_LEVEL_LOW		RO	0x0

IO_BANK0: INTR5 Register

Offset: 0x244

Description

Raw Interrupts

Table 763. INTR5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		WC	0x0
30	GPIO47_EDGE_LOW		WC	0x0
29	GPIO47_LEVEL_HIGH		RO	0x0
28	GPIO47_LEVEL_LOW		RO	0x0
27	GPIO46_EDGE_HIGH		WC	0x0
26	GPIO46_EDGE_LOW		WC	0x0
25	GPIO46_LEVEL_HIGH		RO	0x0
24	GPIO46_LEVEL_LOW		RO	0x0
23	GPIO45_EDGE_HIGH		WC	0x0
22	GPIO45_EDGE_LOW		WC	0x0
21	GPIO45_LEVEL_HIGH		RO	0x0
20	GPIO45_LEVEL_LOW		RO	0x0

Bits	Name	Description	Type	Reset
19	GPIO44_EDGE_HIGH		WC	0x0
18	GPIO44_EDGE_LOW		WC	0x0
17	GPIO44_LEVEL_HIGH		RO	0x0
16	GPIO44_LEVEL_LOW		RO	0x0
15	GPIO43_EDGE_HIGH		WC	0x0
14	GPIO43_EDGE_LOW		WC	0x0
13	GPIO43_LEVEL_HIGH		RO	0x0
12	GPIO43_LEVEL_LOW		RO	0x0
11	GPIO42_EDGE_HIGH		WC	0x0
10	GPIO42_EDGE_LOW		WC	0x0
9	GPIO42_LEVEL_HIGH		RO	0x0
8	GPIO42_LEVEL_LOW		RO	0x0
7	GPIO41_EDGE_HIGH		WC	0x0
6	GPIO41_EDGE_LOW		WC	0x0
5	GPIO41_LEVEL_HIGH		RO	0x0
4	GPIO41_LEVEL_LOW		RO	0x0
3	GPIO40_EDGE_HIGH		WC	0x0
2	GPIO40_EDGE_LOW		WC	0x0
1	GPIO40_LEVEL_HIGH		RO	0x0
0	GPIO40_LEVEL_LOW		RO	0x0

IO_BANK0: PROCO_INTE0 Register

Offset: 0x248

Description

Interrupt Enable for proc0

Table 764.
PROCO_INTE0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RW	0x0
30	GPIO7_EDGE_LOW		RW	0x0
29	GPIO7_LEVEL_HIGH		RW	0x0
28	GPIO7_LEVEL_LOW		RW	0x0
27	GPIO6_EDGE_HIGH		RW	0x0
26	GPIO6_EDGE_LOW		RW	0x0
25	GPIO6_LEVEL_HIGH		RW	0x0
24	GPIO6_LEVEL_LOW		RW	0x0
23	GPIO5_EDGE_HIGH		RW	0x0
22	GPIO5_EDGE_LOW		RW	0x0

Bits	Name	Description	Type	Reset
21	GPIO5_LEVEL_HIGH		RW	0x0
20	GPIO5_LEVEL_LOW		RW	0x0
19	GPIO4_EDGE_HIGH		RW	0x0
18	GPIO4_EDGE_LOW		RW	0x0
17	GPIO4_LEVEL_HIGH		RW	0x0
16	GPIO4_LEVEL_LOW		RW	0x0
15	GPIO3_EDGE_HIGH		RW	0x0
14	GPIO3_EDGE_LOW		RW	0x0
13	GPIO3_LEVEL_HIGH		RW	0x0
12	GPIO3_LEVEL_LOW		RW	0x0
11	GPIO2_EDGE_HIGH		RW	0x0
10	GPIO2_EDGE_LOW		RW	0x0
9	GPIO2_LEVEL_HIGH		RW	0x0
8	GPIO2_LEVEL_LOW		RW	0x0
7	GPIO1_EDGE_HIGH		RW	0x0
6	GPIO1_EDGE_LOW		RW	0x0
5	GPIO1_LEVEL_HIGH		RW	0x0
4	GPIO1_LEVEL_LOW		RW	0x0
3	GPIO0_EDGE_HIGH		RW	0x0
2	GPIO0_EDGE_LOW		RW	0x0
1	GPIO0_LEVEL_HIGH		RW	0x0
0	GPIO0_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTE1 Register

Offset: 0x24c

Description

Interrupt Enable for proc0

Table 765.
PROC0_INTE1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RW	0x0
30	GPIO15_EDGE_LOW		RW	0x0
29	GPIO15_LEVEL_HIGH		RW	0x0
28	GPIO15_LEVEL_LOW		RW	0x0
27	GPIO14_EDGE_HIGH		RW	0x0
26	GPIO14_EDGE_LOW		RW	0x0
25	GPIO14_LEVEL_HIGH		RW	0x0
24	GPIO14_LEVEL_LOW		RW	0x0

Bits	Name	Description	Type	Reset
23	GPIO13_EDGE_HIGH		RW	0x0
22	GPIO13_EDGE_LOW		RW	0x0
21	GPIO13_LEVEL_HIGH		RW	0x0
20	GPIO13_LEVEL_LOW		RW	0x0
19	GPIO12_EDGE_HIGH		RW	0x0
18	GPIO12_EDGE_LOW		RW	0x0
17	GPIO12_LEVEL_HIGH		RW	0x0
16	GPIO12_LEVEL_LOW		RW	0x0
15	GPIO11_EDGE_HIGH		RW	0x0
14	GPIO11_EDGE_LOW		RW	0x0
13	GPIO11_LEVEL_HIGH		RW	0x0
12	GPIO11_LEVEL_LOW		RW	0x0
11	GPIO10_EDGE_HIGH		RW	0x0
10	GPIO10_EDGE_LOW		RW	0x0
9	GPIO10_LEVEL_HIGH		RW	0x0
8	GPIO10_LEVEL_LOW		RW	0x0
7	GPIO9_EDGE_HIGH		RW	0x0
6	GPIO9_EDGE_LOW		RW	0x0
5	GPIO9_LEVEL_HIGH		RW	0x0
4	GPIO9_LEVEL_LOW		RW	0x0
3	GPIO8_EDGE_HIGH		RW	0x0
2	GPIO8_EDGE_LOW		RW	0x0
1	GPIO8_LEVEL_HIGH		RW	0x0
0	GPIO8_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTE2 Register

Offset: 0x250

Description

Interrupt Enable for proc0

Table 766.
PROC0_INTE2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RW	0x0
30	GPIO23_EDGE_LOW		RW	0x0
29	GPIO23_LEVEL_HIGH		RW	0x0
28	GPIO23_LEVEL_LOW		RW	0x0
27	GPIO22_EDGE_HIGH		RW	0x0
26	GPIO22_EDGE_LOW		RW	0x0

Bits	Name	Description	Type	Reset
25	GPIO22_LEVEL_HIGH		RW	0x0
24	GPIO22_LEVEL_LOW		RW	0x0
23	GPIO21_EDGE_HIGH		RW	0x0
22	GPIO21_EDGE_LOW		RW	0x0
21	GPIO21_LEVEL_HIGH		RW	0x0
20	GPIO21_LEVEL_LOW		RW	0x0
19	GPIO20_EDGE_HIGH		RW	0x0
18	GPIO20_EDGE_LOW		RW	0x0
17	GPIO20_LEVEL_HIGH		RW	0x0
16	GPIO20_LEVEL_LOW		RW	0x0
15	GPIO19_EDGE_HIGH		RW	0x0
14	GPIO19_EDGE_LOW		RW	0x0
13	GPIO19_LEVEL_HIGH		RW	0x0
12	GPIO19_LEVEL_LOW		RW	0x0
11	GPIO18_EDGE_HIGH		RW	0x0
10	GPIO18_EDGE_LOW		RW	0x0
9	GPIO18_LEVEL_HIGH		RW	0x0
8	GPIO18_LEVEL_LOW		RW	0x0
7	GPIO17_EDGE_HIGH		RW	0x0
6	GPIO17_EDGE_LOW		RW	0x0
5	GPIO17_LEVEL_HIGH		RW	0x0
4	GPIO17_LEVEL_LOW		RW	0x0
3	GPIO16_EDGE_HIGH		RW	0x0
2	GPIO16_EDGE_LOW		RW	0x0
1	GPIO16_LEVEL_HIGH		RW	0x0
0	GPIO16_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTE3 Register

Offset: 0x254

Description

Interrupt Enable for proc0

Table 767.
PROC0_INTE3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RW	0x0
30	GPIO31_EDGE_LOW		RW	0x0
29	GPIO31_LEVEL_HIGH		RW	0x0
28	GPIO31_LEVEL_LOW		RW	0x0

Bits	Name	Description	Type	Reset
27	GPIO30_EDGE_HIGH		RW	0x0
26	GPIO30_EDGE_LOW		RW	0x0
25	GPIO30_LEVEL_HIGH		RW	0x0
24	GPIO30_LEVEL_LOW		RW	0x0
23	GPIO29_EDGE_HIGH		RW	0x0
22	GPIO29_EDGE_LOW		RW	0x0
21	GPIO29_LEVEL_HIGH		RW	0x0
20	GPIO29_LEVEL_LOW		RW	0x0
19	GPIO28_EDGE_HIGH		RW	0x0
18	GPIO28_EDGE_LOW		RW	0x0
17	GPIO28_LEVEL_HIGH		RW	0x0
16	GPIO28_LEVEL_LOW		RW	0x0
15	GPIO27_EDGE_HIGH		RW	0x0
14	GPIO27_EDGE_LOW		RW	0x0
13	GPIO27_LEVEL_HIGH		RW	0x0
12	GPIO27_LEVEL_LOW		RW	0x0
11	GPIO26_EDGE_HIGH		RW	0x0
10	GPIO26_EDGE_LOW		RW	0x0
9	GPIO26_LEVEL_HIGH		RW	0x0
8	GPIO26_LEVEL_LOW		RW	0x0
7	GPIO25_EDGE_HIGH		RW	0x0
6	GPIO25_EDGE_LOW		RW	0x0
5	GPIO25_LEVEL_HIGH		RW	0x0
4	GPIO25_LEVEL_LOW		RW	0x0
3	GPIO24_EDGE_HIGH		RW	0x0
2	GPIO24_EDGE_LOW		RW	0x0
1	GPIO24_LEVEL_HIGH		RW	0x0
0	GPIO24_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTE4 Register

Offset: 0x258

Description

Interrupt Enable for proc0

Table 768.
PROC0_INTE4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RW	0x0
30	GPIO39_EDGE_LOW		RW	0x0

Bits	Name	Description	Type	Reset
29	GPIO39_LEVEL_HIGH		RW	0x0
28	GPIO39_LEVEL_LOW		RW	0x0
27	GPIO38_EDGE_HIGH		RW	0x0
26	GPIO38_EDGE_LOW		RW	0x0
25	GPIO38_LEVEL_HIGH		RW	0x0
24	GPIO38_LEVEL_LOW		RW	0x0
23	GPIO37_EDGE_HIGH		RW	0x0
22	GPIO37_EDGE_LOW		RW	0x0
21	GPIO37_LEVEL_HIGH		RW	0x0
20	GPIO37_LEVEL_LOW		RW	0x0
19	GPIO36_EDGE_HIGH		RW	0x0
18	GPIO36_EDGE_LOW		RW	0x0
17	GPIO36_LEVEL_HIGH		RW	0x0
16	GPIO36_LEVEL_LOW		RW	0x0
15	GPIO35_EDGE_HIGH		RW	0x0
14	GPIO35_EDGE_LOW		RW	0x0
13	GPIO35_LEVEL_HIGH		RW	0x0
12	GPIO35_LEVEL_LOW		RW	0x0
11	GPIO34_EDGE_HIGH		RW	0x0
10	GPIO34_EDGE_LOW		RW	0x0
9	GPIO34_LEVEL_HIGH		RW	0x0
8	GPIO34_LEVEL_LOW		RW	0x0
7	GPIO33_EDGE_HIGH		RW	0x0
6	GPIO33_EDGE_LOW		RW	0x0
5	GPIO33_LEVEL_HIGH		RW	0x0
4	GPIO33_LEVEL_LOW		RW	0x0
3	GPIO32_EDGE_HIGH		RW	0x0
2	GPIO32_EDGE_LOW		RW	0x0
1	GPIO32_LEVEL_HIGH		RW	0x0
0	GPIO32_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTE5 Register

Offset: 0x25c

Description

Interrupt Enable for proc0

Table 769.
PROCO_INTE5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RW	0x0
30	GPIO47_EDGE_LOW		RW	0x0
29	GPIO47_LEVEL_HIGH		RW	0x0
28	GPIO47_LEVEL_LOW		RW	0x0
27	GPIO46_EDGE_HIGH		RW	0x0
26	GPIO46_EDGE_LOW		RW	0x0
25	GPIO46_LEVEL_HIGH		RW	0x0
24	GPIO46_LEVEL_LOW		RW	0x0
23	GPIO45_EDGE_HIGH		RW	0x0
22	GPIO45_EDGE_LOW		RW	0x0
21	GPIO45_LEVEL_HIGH		RW	0x0
20	GPIO45_LEVEL_LOW		RW	0x0
19	GPIO44_EDGE_HIGH		RW	0x0
18	GPIO44_EDGE_LOW		RW	0x0
17	GPIO44_LEVEL_HIGH		RW	0x0
16	GPIO44_LEVEL_LOW		RW	0x0
15	GPIO43_EDGE_HIGH		RW	0x0
14	GPIO43_EDGE_LOW		RW	0x0
13	GPIO43_LEVEL_HIGH		RW	0x0
12	GPIO43_LEVEL_LOW		RW	0x0
11	GPIO42_EDGE_HIGH		RW	0x0
10	GPIO42_EDGE_LOW		RW	0x0
9	GPIO42_LEVEL_HIGH		RW	0x0
8	GPIO42_LEVEL_LOW		RW	0x0
7	GPIO41_EDGE_HIGH		RW	0x0
6	GPIO41_EDGE_LOW		RW	0x0
5	GPIO41_LEVEL_HIGH		RW	0x0
4	GPIO41_LEVEL_LOW		RW	0x0
3	GPIO40_EDGE_HIGH		RW	0x0
2	GPIO40_EDGE_LOW		RW	0x0
1	GPIO40_LEVEL_HIGH		RW	0x0
0	GPIO40_LEVEL_LOW		RW	0x0

IO_BANK0: PROCO_INTF0 Register

Offset: 0x260

Description

Interrupt Force for proc0

Table 770.
PROC0_INTF0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RW	0x0
30	GPIO7_EDGE_LOW		RW	0x0
29	GPIO7_LEVEL_HIGH		RW	0x0
28	GPIO7_LEVEL_LOW		RW	0x0
27	GPIO6_EDGE_HIGH		RW	0x0
26	GPIO6_EDGE_LOW		RW	0x0
25	GPIO6_LEVEL_HIGH		RW	0x0
24	GPIO6_LEVEL_LOW		RW	0x0
23	GPIO5_EDGE_HIGH		RW	0x0
22	GPIO5_EDGE_LOW		RW	0x0
21	GPIO5_LEVEL_HIGH		RW	0x0
20	GPIO5_LEVEL_LOW		RW	0x0
19	GPIO4_EDGE_HIGH		RW	0x0
18	GPIO4_EDGE_LOW		RW	0x0
17	GPIO4_LEVEL_HIGH		RW	0x0
16	GPIO4_LEVEL_LOW		RW	0x0
15	GPIO3_EDGE_HIGH		RW	0x0
14	GPIO3_EDGE_LOW		RW	0x0
13	GPIO3_LEVEL_HIGH		RW	0x0
12	GPIO3_LEVEL_LOW		RW	0x0
11	GPIO2_EDGE_HIGH		RW	0x0
10	GPIO2_EDGE_LOW		RW	0x0
9	GPIO2_LEVEL_HIGH		RW	0x0
8	GPIO2_LEVEL_LOW		RW	0x0
7	GPIO1_EDGE_HIGH		RW	0x0
6	GPIO1_EDGE_LOW		RW	0x0
5	GPIO1_LEVEL_HIGH		RW	0x0
4	GPIO1_LEVEL_LOW		RW	0x0
3	GPIO0_EDGE_HIGH		RW	0x0
2	GPIO0_EDGE_LOW		RW	0x0
1	GPIO0_LEVEL_HIGH		RW	0x0
0	GPIO0_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTF1 Register

Offset: 0x264

Description

Interrupt Force for proc0

Table 771.
PROC0_INTF1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RW	0x0
30	GPIO15_EDGE_LOW		RW	0x0
29	GPIO15_LEVEL_HIGH		RW	0x0
28	GPIO15_LEVEL_LOW		RW	0x0
27	GPIO14_EDGE_HIGH		RW	0x0
26	GPIO14_EDGE_LOW		RW	0x0
25	GPIO14_LEVEL_HIGH		RW	0x0
24	GPIO14_LEVEL_LOW		RW	0x0
23	GPIO13_EDGE_HIGH		RW	0x0
22	GPIO13_EDGE_LOW		RW	0x0
21	GPIO13_LEVEL_HIGH		RW	0x0
20	GPIO13_LEVEL_LOW		RW	0x0
19	GPIO12_EDGE_HIGH		RW	0x0
18	GPIO12_EDGE_LOW		RW	0x0
17	GPIO12_LEVEL_HIGH		RW	0x0
16	GPIO12_LEVEL_LOW		RW	0x0
15	GPIO11_EDGE_HIGH		RW	0x0
14	GPIO11_EDGE_LOW		RW	0x0
13	GPIO11_LEVEL_HIGH		RW	0x0
12	GPIO11_LEVEL_LOW		RW	0x0
11	GPIO10_EDGE_HIGH		RW	0x0
10	GPIO10_EDGE_LOW		RW	0x0
9	GPIO10_LEVEL_HIGH		RW	0x0
8	GPIO10_LEVEL_LOW		RW	0x0
7	GPIO9_EDGE_HIGH		RW	0x0
6	GPIO9_EDGE_LOW		RW	0x0
5	GPIO9_LEVEL_HIGH		RW	0x0
4	GPIO9_LEVEL_LOW		RW	0x0
3	GPIO8_EDGE_HIGH		RW	0x0
2	GPIO8_EDGE_LOW		RW	0x0
1	GPIO8_LEVEL_HIGH		RW	0x0
0	GPIO8_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTF2 Register

Offset: 0x268

Description

Interrupt Force for proc0

Table 772.
PROC0_INTF2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RW	0x0
30	GPIO23_EDGE_LOW		RW	0x0
29	GPIO23_LEVEL_HIGH		RW	0x0
28	GPIO23_LEVEL_LOW		RW	0x0
27	GPIO22_EDGE_HIGH		RW	0x0
26	GPIO22_EDGE_LOW		RW	0x0
25	GPIO22_LEVEL_HIGH		RW	0x0
24	GPIO22_LEVEL_LOW		RW	0x0
23	GPIO21_EDGE_HIGH		RW	0x0
22	GPIO21_EDGE_LOW		RW	0x0
21	GPIO21_LEVEL_HIGH		RW	0x0
20	GPIO21_LEVEL_LOW		RW	0x0
19	GPIO20_EDGE_HIGH		RW	0x0
18	GPIO20_EDGE_LOW		RW	0x0
17	GPIO20_LEVEL_HIGH		RW	0x0
16	GPIO20_LEVEL_LOW		RW	0x0
15	GPIO19_EDGE_HIGH		RW	0x0
14	GPIO19_EDGE_LOW		RW	0x0
13	GPIO19_LEVEL_HIGH		RW	0x0
12	GPIO19_LEVEL_LOW		RW	0x0
11	GPIO18_EDGE_HIGH		RW	0x0
10	GPIO18_EDGE_LOW		RW	0x0
9	GPIO18_LEVEL_HIGH		RW	0x0
8	GPIO18_LEVEL_LOW		RW	0x0
7	GPIO17_EDGE_HIGH		RW	0x0
6	GPIO17_EDGE_LOW		RW	0x0
5	GPIO17_LEVEL_HIGH		RW	0x0
4	GPIO17_LEVEL_LOW		RW	0x0
3	GPIO16_EDGE_HIGH		RW	0x0
2	GPIO16_EDGE_LOW		RW	0x0
1	GPIO16_LEVEL_HIGH		RW	0x0
0	GPIO16_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTF3 Register

Offset: 0x26c

Description

Interrupt Force for proc0

Table 773.
PROC0_INTF3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RW	0x0
30	GPIO31_EDGE_LOW		RW	0x0
29	GPIO31_LEVEL_HIGH		RW	0x0
28	GPIO31_LEVEL_LOW		RW	0x0
27	GPIO30_EDGE_HIGH		RW	0x0
26	GPIO30_EDGE_LOW		RW	0x0
25	GPIO30_LEVEL_HIGH		RW	0x0
24	GPIO30_LEVEL_LOW		RW	0x0
23	GPIO29_EDGE_HIGH		RW	0x0
22	GPIO29_EDGE_LOW		RW	0x0
21	GPIO29_LEVEL_HIGH		RW	0x0
20	GPIO29_LEVEL_LOW		RW	0x0
19	GPIO28_EDGE_HIGH		RW	0x0
18	GPIO28_EDGE_LOW		RW	0x0
17	GPIO28_LEVEL_HIGH		RW	0x0
16	GPIO28_LEVEL_LOW		RW	0x0
15	GPIO27_EDGE_HIGH		RW	0x0
14	GPIO27_EDGE_LOW		RW	0x0
13	GPIO27_LEVEL_HIGH		RW	0x0
12	GPIO27_LEVEL_LOW		RW	0x0
11	GPIO26_EDGE_HIGH		RW	0x0
10	GPIO26_EDGE_LOW		RW	0x0
9	GPIO26_LEVEL_HIGH		RW	0x0
8	GPIO26_LEVEL_LOW		RW	0x0
7	GPIO25_EDGE_HIGH		RW	0x0
6	GPIO25_EDGE_LOW		RW	0x0
5	GPIO25_LEVEL_HIGH		RW	0x0
4	GPIO25_LEVEL_LOW		RW	0x0
3	GPIO24_EDGE_HIGH		RW	0x0
2	GPIO24_EDGE_LOW		RW	0x0
1	GPIO24_LEVEL_HIGH		RW	0x0
0	GPIO24_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTF4 Register

Offset: 0x270

Description

Interrupt Force for proc0

Table 774.
PROC0_INTF4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RW	0x0
30	GPIO39_EDGE_LOW		RW	0x0
29	GPIO39_LEVEL_HIGH		RW	0x0
28	GPIO39_LEVEL_LOW		RW	0x0
27	GPIO38_EDGE_HIGH		RW	0x0
26	GPIO38_EDGE_LOW		RW	0x0
25	GPIO38_LEVEL_HIGH		RW	0x0
24	GPIO38_LEVEL_LOW		RW	0x0
23	GPIO37_EDGE_HIGH		RW	0x0
22	GPIO37_EDGE_LOW		RW	0x0
21	GPIO37_LEVEL_HIGH		RW	0x0
20	GPIO37_LEVEL_LOW		RW	0x0
19	GPIO36_EDGE_HIGH		RW	0x0
18	GPIO36_EDGE_LOW		RW	0x0
17	GPIO36_LEVEL_HIGH		RW	0x0
16	GPIO36_LEVEL_LOW		RW	0x0
15	GPIO35_EDGE_HIGH		RW	0x0
14	GPIO35_EDGE_LOW		RW	0x0
13	GPIO35_LEVEL_HIGH		RW	0x0
12	GPIO35_LEVEL_LOW		RW	0x0
11	GPIO34_EDGE_HIGH		RW	0x0
10	GPIO34_EDGE_LOW		RW	0x0
9	GPIO34_LEVEL_HIGH		RW	0x0
8	GPIO34_LEVEL_LOW		RW	0x0
7	GPIO33_EDGE_HIGH		RW	0x0
6	GPIO33_EDGE_LOW		RW	0x0
5	GPIO33_LEVEL_HIGH		RW	0x0
4	GPIO33_LEVEL_LOW		RW	0x0
3	GPIO32_EDGE_HIGH		RW	0x0
2	GPIO32_EDGE_LOW		RW	0x0
1	GPIO32_LEVEL_HIGH		RW	0x0
0	GPIO32_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTF5 Register

Offset: 0x274

Description

Interrupt Force for proc0

Table 775.
PROC0_INTE5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RW	0x0
30	GPIO47_EDGE_LOW		RW	0x0
29	GPIO47_LEVEL_HIGH		RW	0x0
28	GPIO47_LEVEL_LOW		RW	0x0
27	GPIO46_EDGE_HIGH		RW	0x0
26	GPIO46_EDGE_LOW		RW	0x0
25	GPIO46_LEVEL_HIGH		RW	0x0
24	GPIO46_LEVEL_LOW		RW	0x0
23	GPIO45_EDGE_HIGH		RW	0x0
22	GPIO45_EDGE_LOW		RW	0x0
21	GPIO45_LEVEL_HIGH		RW	0x0
20	GPIO45_LEVEL_LOW		RW	0x0
19	GPIO44_EDGE_HIGH		RW	0x0
18	GPIO44_EDGE_LOW		RW	0x0
17	GPIO44_LEVEL_HIGH		RW	0x0
16	GPIO44_LEVEL_LOW		RW	0x0
15	GPIO43_EDGE_HIGH		RW	0x0
14	GPIO43_EDGE_LOW		RW	0x0
13	GPIO43_LEVEL_HIGH		RW	0x0
12	GPIO43_LEVEL_LOW		RW	0x0
11	GPIO42_EDGE_HIGH		RW	0x0
10	GPIO42_EDGE_LOW		RW	0x0
9	GPIO42_LEVEL_HIGH		RW	0x0
8	GPIO42_LEVEL_LOW		RW	0x0
7	GPIO41_EDGE_HIGH		RW	0x0
6	GPIO41_EDGE_LOW		RW	0x0
5	GPIO41_LEVEL_HIGH		RW	0x0
4	GPIO41_LEVEL_LOW		RW	0x0
3	GPIO40_EDGE_HIGH		RW	0x0
2	GPIO40_EDGE_LOW		RW	0x0
1	GPIO40_LEVEL_HIGH		RW	0x0
0	GPIO40_LEVEL_LOW		RW	0x0

IO_BANK0: PROC0_INTS0 Register

Offset: 0x278

Description

Interrupt status after masking & forcing for proc0

Table 776.
PROC0_INTS0
Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RO	0x0
30	GPIO7_EDGE_LOW		RO	0x0
29	GPIO7_LEVEL_HIGH		RO	0x0
28	GPIO7_LEVEL_LOW		RO	0x0
27	GPIO6_EDGE_HIGH		RO	0x0
26	GPIO6_EDGE_LOW		RO	0x0
25	GPIO6_LEVEL_HIGH		RO	0x0
24	GPIO6_LEVEL_LOW		RO	0x0
23	GPIO5_EDGE_HIGH		RO	0x0
22	GPIO5_EDGE_LOW		RO	0x0
21	GPIO5_LEVEL_HIGH		RO	0x0
20	GPIO5_LEVEL_LOW		RO	0x0
19	GPIO4_EDGE_HIGH		RO	0x0
18	GPIO4_EDGE_LOW		RO	0x0
17	GPIO4_LEVEL_HIGH		RO	0x0
16	GPIO4_LEVEL_LOW		RO	0x0
15	GPIO3_EDGE_HIGH		RO	0x0
14	GPIO3_EDGE_LOW		RO	0x0
13	GPIO3_LEVEL_HIGH		RO	0x0
12	GPIO3_LEVEL_LOW		RO	0x0
11	GPIO2_EDGE_HIGH		RO	0x0
10	GPIO2_EDGE_LOW		RO	0x0
9	GPIO2_LEVEL_HIGH		RO	0x0
8	GPIO2_LEVEL_LOW		RO	0x0
7	GPIO1_EDGE_HIGH		RO	0x0
6	GPIO1_EDGE_LOW		RO	0x0
5	GPIO1_LEVEL_HIGH		RO	0x0
4	GPIO1_LEVEL_LOW		RO	0x0
3	GPIO0_EDGE_HIGH		RO	0x0
2	GPIO0_EDGE_LOW		RO	0x0
1	GPIO0_LEVEL_HIGH		RO	0x0
0	GPIO0_LEVEL_LOW		RO	0x0

IO_BANK0: PROC0_INTS1 Register

Offset: 0x27c

Description

Interrupt status after masking & forcing for proc0

Table 777.
PROC0_INTS1
Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RO	0x0
30	GPIO15_EDGE_LOW		RO	0x0
29	GPIO15_LEVEL_HIGH		RO	0x0
28	GPIO15_LEVEL_LOW		RO	0x0
27	GPIO14_EDGE_HIGH		RO	0x0
26	GPIO14_EDGE_LOW		RO	0x0
25	GPIO14_LEVEL_HIGH		RO	0x0
24	GPIO14_LEVEL_LOW		RO	0x0
23	GPIO13_EDGE_HIGH		RO	0x0
22	GPIO13_EDGE_LOW		RO	0x0
21	GPIO13_LEVEL_HIGH		RO	0x0
20	GPIO13_LEVEL_LOW		RO	0x0
19	GPIO12_EDGE_HIGH		RO	0x0
18	GPIO12_EDGE_LOW		RO	0x0
17	GPIO12_LEVEL_HIGH		RO	0x0
16	GPIO12_LEVEL_LOW		RO	0x0
15	GPIO11_EDGE_HIGH		RO	0x0
14	GPIO11_EDGE_LOW		RO	0x0
13	GPIO11_LEVEL_HIGH		RO	0x0
12	GPIO11_LEVEL_LOW		RO	0x0
11	GPIO10_EDGE_HIGH		RO	0x0
10	GPIO10_EDGE_LOW		RO	0x0
9	GPIO10_LEVEL_HIGH		RO	0x0
8	GPIO10_LEVEL_LOW		RO	0x0
7	GPIO9_EDGE_HIGH		RO	0x0
6	GPIO9_EDGE_LOW		RO	0x0
5	GPIO9_LEVEL_HIGH		RO	0x0
4	GPIO9_LEVEL_LOW		RO	0x0
3	GPIO8_EDGE_HIGH		RO	0x0
2	GPIO8_EDGE_LOW		RO	0x0
1	GPIO8_LEVEL_HIGH		RO	0x0
0	GPIO8_LEVEL_LOW		RO	0x0

IO_BANK0: PROC0_INTS2 Register

Offset: 0x280

Description

Interrupt status after masking & forcing for proc0

Table 778.
PROC0_INTS2
Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RO	0x0
30	GPIO23_EDGE_LOW		RO	0x0
29	GPIO23_LEVEL_HIGH		RO	0x0
28	GPIO23_LEVEL_LOW		RO	0x0
27	GPIO22_EDGE_HIGH		RO	0x0
26	GPIO22_EDGE_LOW		RO	0x0
25	GPIO22_LEVEL_HIGH		RO	0x0
24	GPIO22_LEVEL_LOW		RO	0x0
23	GPIO21_EDGE_HIGH		RO	0x0
22	GPIO21_EDGE_LOW		RO	0x0
21	GPIO21_LEVEL_HIGH		RO	0x0
20	GPIO21_LEVEL_LOW		RO	0x0
19	GPIO20_EDGE_HIGH		RO	0x0
18	GPIO20_EDGE_LOW		RO	0x0
17	GPIO20_LEVEL_HIGH		RO	0x0
16	GPIO20_LEVEL_LOW		RO	0x0
15	GPIO19_EDGE_HIGH		RO	0x0
14	GPIO19_EDGE_LOW		RO	0x0
13	GPIO19_LEVEL_HIGH		RO	0x0
12	GPIO19_LEVEL_LOW		RO	0x0
11	GPIO18_EDGE_HIGH		RO	0x0
10	GPIO18_EDGE_LOW		RO	0x0
9	GPIO18_LEVEL_HIGH		RO	0x0
8	GPIO18_LEVEL_LOW		RO	0x0
7	GPIO17_EDGE_HIGH		RO	0x0
6	GPIO17_EDGE_LOW		RO	0x0
5	GPIO17_LEVEL_HIGH		RO	0x0
4	GPIO17_LEVEL_LOW		RO	0x0
3	GPIO16_EDGE_HIGH		RO	0x0
2	GPIO16_EDGE_LOW		RO	0x0
1	GPIO16_LEVEL_HIGH		RO	0x0
0	GPIO16_LEVEL_LOW		RO	0x0

IO_BANK0: PROC0_INTS3 Register

Offset: 0x284

Description

Interrupt status after masking & forcing for proc0

Table 779.
PROC0_INTS3
Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RO	0x0
30	GPIO31_EDGE_LOW		RO	0x0
29	GPIO31_LEVEL_HIGH		RO	0x0
28	GPIO31_LEVEL_LOW		RO	0x0
27	GPIO30_EDGE_HIGH		RO	0x0
26	GPIO30_EDGE_LOW		RO	0x0
25	GPIO30_LEVEL_HIGH		RO	0x0
24	GPIO30_LEVEL_LOW		RO	0x0
23	GPIO29_EDGE_HIGH		RO	0x0
22	GPIO29_EDGE_LOW		RO	0x0
21	GPIO29_LEVEL_HIGH		RO	0x0
20	GPIO29_LEVEL_LOW		RO	0x0
19	GPIO28_EDGE_HIGH		RO	0x0
18	GPIO28_EDGE_LOW		RO	0x0
17	GPIO28_LEVEL_HIGH		RO	0x0
16	GPIO28_LEVEL_LOW		RO	0x0
15	GPIO27_EDGE_HIGH		RO	0x0
14	GPIO27_EDGE_LOW		RO	0x0
13	GPIO27_LEVEL_HIGH		RO	0x0
12	GPIO27_LEVEL_LOW		RO	0x0
11	GPIO26_EDGE_HIGH		RO	0x0
10	GPIO26_EDGE_LOW		RO	0x0
9	GPIO26_LEVEL_HIGH		RO	0x0
8	GPIO26_LEVEL_LOW		RO	0x0
7	GPIO25_EDGE_HIGH		RO	0x0
6	GPIO25_EDGE_LOW		RO	0x0
5	GPIO25_LEVEL_HIGH		RO	0x0
4	GPIO25_LEVEL_LOW		RO	0x0
3	GPIO24_EDGE_HIGH		RO	0x0
2	GPIO24_EDGE_LOW		RO	0x0
1	GPIO24_LEVEL_HIGH		RO	0x0
0	GPIO24_LEVEL_LOW		RO	0x0

IO_BANK0: PROC0_INTS4 Register

Offset: 0x288

Description

Interrupt status after masking & forcing for proc0

Table 780.
PROC0_INTS4
Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RO	0x0
30	GPIO39_EDGE_LOW		RO	0x0
29	GPIO39_LEVEL_HIGH		RO	0x0
28	GPIO39_LEVEL_LOW		RO	0x0
27	GPIO38_EDGE_HIGH		RO	0x0
26	GPIO38_EDGE_LOW		RO	0x0
25	GPIO38_LEVEL_HIGH		RO	0x0
24	GPIO38_LEVEL_LOW		RO	0x0
23	GPIO37_EDGE_HIGH		RO	0x0
22	GPIO37_EDGE_LOW		RO	0x0
21	GPIO37_LEVEL_HIGH		RO	0x0
20	GPIO37_LEVEL_LOW		RO	0x0
19	GPIO36_EDGE_HIGH		RO	0x0
18	GPIO36_EDGE_LOW		RO	0x0
17	GPIO36_LEVEL_HIGH		RO	0x0
16	GPIO36_LEVEL_LOW		RO	0x0
15	GPIO35_EDGE_HIGH		RO	0x0
14	GPIO35_EDGE_LOW		RO	0x0
13	GPIO35_LEVEL_HIGH		RO	0x0
12	GPIO35_LEVEL_LOW		RO	0x0
11	GPIO34_EDGE_HIGH		RO	0x0
10	GPIO34_EDGE_LOW		RO	0x0
9	GPIO34_LEVEL_HIGH		RO	0x0
8	GPIO34_LEVEL_LOW		RO	0x0
7	GPIO33_EDGE_HIGH		RO	0x0
6	GPIO33_EDGE_LOW		RO	0x0
5	GPIO33_LEVEL_HIGH		RO	0x0
4	GPIO33_LEVEL_LOW		RO	0x0
3	GPIO32_EDGE_HIGH		RO	0x0
2	GPIO32_EDGE_LOW		RO	0x0
1	GPIO32_LEVEL_HIGH		RO	0x0
0	GPIO32_LEVEL_LOW		RO	0x0

IO_BANK0: PROC0_INTS5 Register

Offset: 0x28c

Description

Interrupt status after masking & forcing for proc0

Table 781.
PROC0_INTS5
Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RO	0x0
30	GPIO47_EDGE_LOW		RO	0x0
29	GPIO47_LEVEL_HIGH		RO	0x0
28	GPIO47_LEVEL_LOW		RO	0x0
27	GPIO46_EDGE_HIGH		RO	0x0
26	GPIO46_EDGE_LOW		RO	0x0
25	GPIO46_LEVEL_HIGH		RO	0x0
24	GPIO46_LEVEL_LOW		RO	0x0
23	GPIO45_EDGE_HIGH		RO	0x0
22	GPIO45_EDGE_LOW		RO	0x0
21	GPIO45_LEVEL_HIGH		RO	0x0
20	GPIO45_LEVEL_LOW		RO	0x0
19	GPIO44_EDGE_HIGH		RO	0x0
18	GPIO44_EDGE_LOW		RO	0x0
17	GPIO44_LEVEL_HIGH		RO	0x0
16	GPIO44_LEVEL_LOW		RO	0x0
15	GPIO43_EDGE_HIGH		RO	0x0
14	GPIO43_EDGE_LOW		RO	0x0
13	GPIO43_LEVEL_HIGH		RO	0x0
12	GPIO43_LEVEL_LOW		RO	0x0
11	GPIO42_EDGE_HIGH		RO	0x0
10	GPIO42_EDGE_LOW		RO	0x0
9	GPIO42_LEVEL_HIGH		RO	0x0
8	GPIO42_LEVEL_LOW		RO	0x0
7	GPIO41_EDGE_HIGH		RO	0x0
6	GPIO41_EDGE_LOW		RO	0x0
5	GPIO41_LEVEL_HIGH		RO	0x0
4	GPIO41_LEVEL_LOW		RO	0x0
3	GPIO40_EDGE_HIGH		RO	0x0
2	GPIO40_EDGE_LOW		RO	0x0
1	GPIO40_LEVEL_HIGH		RO	0x0
0	GPIO40_LEVEL_LOW		RO	0x0

IO_BANK0: PROC1_INTE0 Register

Offset: 0x290

Description

Interrupt Enable for proc1

Table 782.
PROC1_INTE0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RW	0x0
30	GPIO7_EDGE_LOW		RW	0x0
29	GPIO7_LEVEL_HIGH		RW	0x0
28	GPIO7_LEVEL_LOW		RW	0x0
27	GPIO6_EDGE_HIGH		RW	0x0
26	GPIO6_EDGE_LOW		RW	0x0
25	GPIO6_LEVEL_HIGH		RW	0x0
24	GPIO6_LEVEL_LOW		RW	0x0
23	GPIO5_EDGE_HIGH		RW	0x0
22	GPIO5_EDGE_LOW		RW	0x0
21	GPIO5_LEVEL_HIGH		RW	0x0
20	GPIO5_LEVEL_LOW		RW	0x0
19	GPIO4_EDGE_HIGH		RW	0x0
18	GPIO4_EDGE_LOW		RW	0x0
17	GPIO4_LEVEL_HIGH		RW	0x0
16	GPIO4_LEVEL_LOW		RW	0x0
15	GPIO3_EDGE_HIGH		RW	0x0
14	GPIO3_EDGE_LOW		RW	0x0
13	GPIO3_LEVEL_HIGH		RW	0x0
12	GPIO3_LEVEL_LOW		RW	0x0
11	GPIO2_EDGE_HIGH		RW	0x0
10	GPIO2_EDGE_LOW		RW	0x0
9	GPIO2_LEVEL_HIGH		RW	0x0
8	GPIO2_LEVEL_LOW		RW	0x0
7	GPIO1_EDGE_HIGH		RW	0x0
6	GPIO1_EDGE_LOW		RW	0x0
5	GPIO1_LEVEL_HIGH		RW	0x0
4	GPIO1_LEVEL_LOW		RW	0x0
3	GPIO0_EDGE_HIGH		RW	0x0
2	GPIO0_EDGE_LOW		RW	0x0
1	GPIO0_LEVEL_HIGH		RW	0x0
0	GPIO0_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTE1 Register

Offset: 0x294

Description

Interrupt Enable for proc1

Table 783.
PROC1_INTE1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RW	0x0
30	GPIO15_EDGE_LOW		RW	0x0
29	GPIO15_LEVEL_HIGH		RW	0x0
28	GPIO15_LEVEL_LOW		RW	0x0
27	GPIO14_EDGE_HIGH		RW	0x0
26	GPIO14_EDGE_LOW		RW	0x0
25	GPIO14_LEVEL_HIGH		RW	0x0
24	GPIO14_LEVEL_LOW		RW	0x0
23	GPIO13_EDGE_HIGH		RW	0x0
22	GPIO13_EDGE_LOW		RW	0x0
21	GPIO13_LEVEL_HIGH		RW	0x0
20	GPIO13_LEVEL_LOW		RW	0x0
19	GPIO12_EDGE_HIGH		RW	0x0
18	GPIO12_EDGE_LOW		RW	0x0
17	GPIO12_LEVEL_HIGH		RW	0x0
16	GPIO12_LEVEL_LOW		RW	0x0
15	GPIO11_EDGE_HIGH		RW	0x0
14	GPIO11_EDGE_LOW		RW	0x0
13	GPIO11_LEVEL_HIGH		RW	0x0
12	GPIO11_LEVEL_LOW		RW	0x0
11	GPIO10_EDGE_HIGH		RW	0x0
10	GPIO10_EDGE_LOW		RW	0x0
9	GPIO10_LEVEL_HIGH		RW	0x0
8	GPIO10_LEVEL_LOW		RW	0x0
7	GPIO9_EDGE_HIGH		RW	0x0
6	GPIO9_EDGE_LOW		RW	0x0
5	GPIO9_LEVEL_HIGH		RW	0x0
4	GPIO9_LEVEL_LOW		RW	0x0
3	GPIO8_EDGE_HIGH		RW	0x0
2	GPIO8_EDGE_LOW		RW	0x0
1	GPIO8_LEVEL_HIGH		RW	0x0
0	GPIO8_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTE2 Register

Offset: 0x298

Description

Interrupt Enable for proc1

Table 784.
PROC1_INTE2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RW	0x0
30	GPIO23_EDGE_LOW		RW	0x0
29	GPIO23_LEVEL_HIGH		RW	0x0
28	GPIO23_LEVEL_LOW		RW	0x0
27	GPIO22_EDGE_HIGH		RW	0x0
26	GPIO22_EDGE_LOW		RW	0x0
25	GPIO22_LEVEL_HIGH		RW	0x0
24	GPIO22_LEVEL_LOW		RW	0x0
23	GPIO21_EDGE_HIGH		RW	0x0
22	GPIO21_EDGE_LOW		RW	0x0
21	GPIO21_LEVEL_HIGH		RW	0x0
20	GPIO21_LEVEL_LOW		RW	0x0
19	GPIO20_EDGE_HIGH		RW	0x0
18	GPIO20_EDGE_LOW		RW	0x0
17	GPIO20_LEVEL_HIGH		RW	0x0
16	GPIO20_LEVEL_LOW		RW	0x0
15	GPIO19_EDGE_HIGH		RW	0x0
14	GPIO19_EDGE_LOW		RW	0x0
13	GPIO19_LEVEL_HIGH		RW	0x0
12	GPIO19_LEVEL_LOW		RW	0x0
11	GPIO18_EDGE_HIGH		RW	0x0
10	GPIO18_EDGE_LOW		RW	0x0
9	GPIO18_LEVEL_HIGH		RW	0x0
8	GPIO18_LEVEL_LOW		RW	0x0
7	GPIO17_EDGE_HIGH		RW	0x0
6	GPIO17_EDGE_LOW		RW	0x0
5	GPIO17_LEVEL_HIGH		RW	0x0
4	GPIO17_LEVEL_LOW		RW	0x0
3	GPIO16_EDGE_HIGH		RW	0x0
2	GPIO16_EDGE_LOW		RW	0x0
1	GPIO16_LEVEL_HIGH		RW	0x0
0	GPIO16_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTE3 Register

Offset: 0x29c

Description

Interrupt Enable for proc1

Table 785.
PROC1_INTE3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RW	0x0
30	GPIO31_EDGE_LOW		RW	0x0
29	GPIO31_LEVEL_HIGH		RW	0x0
28	GPIO31_LEVEL_LOW		RW	0x0
27	GPIO30_EDGE_HIGH		RW	0x0
26	GPIO30_EDGE_LOW		RW	0x0
25	GPIO30_LEVEL_HIGH		RW	0x0
24	GPIO30_LEVEL_LOW		RW	0x0
23	GPIO29_EDGE_HIGH		RW	0x0
22	GPIO29_EDGE_LOW		RW	0x0
21	GPIO29_LEVEL_HIGH		RW	0x0
20	GPIO29_LEVEL_LOW		RW	0x0
19	GPIO28_EDGE_HIGH		RW	0x0
18	GPIO28_EDGE_LOW		RW	0x0
17	GPIO28_LEVEL_HIGH		RW	0x0
16	GPIO28_LEVEL_LOW		RW	0x0
15	GPIO27_EDGE_HIGH		RW	0x0
14	GPIO27_EDGE_LOW		RW	0x0
13	GPIO27_LEVEL_HIGH		RW	0x0
12	GPIO27_LEVEL_LOW		RW	0x0
11	GPIO26_EDGE_HIGH		RW	0x0
10	GPIO26_EDGE_LOW		RW	0x0
9	GPIO26_LEVEL_HIGH		RW	0x0
8	GPIO26_LEVEL_LOW		RW	0x0
7	GPIO25_EDGE_HIGH		RW	0x0
6	GPIO25_EDGE_LOW		RW	0x0
5	GPIO25_LEVEL_HIGH		RW	0x0
4	GPIO25_LEVEL_LOW		RW	0x0
3	GPIO24_EDGE_HIGH		RW	0x0
2	GPIO24_EDGE_LOW		RW	0x0
1	GPIO24_LEVEL_HIGH		RW	0x0
0	GPIO24_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTE4 Register

Offset: 0x2a0

Description

Interrupt Enable for proc1

Table 786.
PROC1_INTE4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RW	0x0
30	GPIO39_EDGE_LOW		RW	0x0
29	GPIO39_LEVEL_HIGH		RW	0x0
28	GPIO39_LEVEL_LOW		RW	0x0
27	GPIO38_EDGE_HIGH		RW	0x0
26	GPIO38_EDGE_LOW		RW	0x0
25	GPIO38_LEVEL_HIGH		RW	0x0
24	GPIO38_LEVEL_LOW		RW	0x0
23	GPIO37_EDGE_HIGH		RW	0x0
22	GPIO37_EDGE_LOW		RW	0x0
21	GPIO37_LEVEL_HIGH		RW	0x0
20	GPIO37_LEVEL_LOW		RW	0x0
19	GPIO36_EDGE_HIGH		RW	0x0
18	GPIO36_EDGE_LOW		RW	0x0
17	GPIO36_LEVEL_HIGH		RW	0x0
16	GPIO36_LEVEL_LOW		RW	0x0
15	GPIO35_EDGE_HIGH		RW	0x0
14	GPIO35_EDGE_LOW		RW	0x0
13	GPIO35_LEVEL_HIGH		RW	0x0
12	GPIO35_LEVEL_LOW		RW	0x0
11	GPIO34_EDGE_HIGH		RW	0x0
10	GPIO34_EDGE_LOW		RW	0x0
9	GPIO34_LEVEL_HIGH		RW	0x0
8	GPIO34_LEVEL_LOW		RW	0x0
7	GPIO33_EDGE_HIGH		RW	0x0
6	GPIO33_EDGE_LOW		RW	0x0
5	GPIO33_LEVEL_HIGH		RW	0x0
4	GPIO33_LEVEL_LOW		RW	0x0
3	GPIO32_EDGE_HIGH		RW	0x0
2	GPIO32_EDGE_LOW		RW	0x0
1	GPIO32_LEVEL_HIGH		RW	0x0
0	GPIO32_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTE5 Register

Offset: 0x2a4

Description

Interrupt Enable for proc1

Table 787.
PROC1_INTE5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RW	0x0
30	GPIO47_EDGE_LOW		RW	0x0
29	GPIO47_LEVEL_HIGH		RW	0x0
28	GPIO47_LEVEL_LOW		RW	0x0
27	GPIO46_EDGE_HIGH		RW	0x0
26	GPIO46_EDGE_LOW		RW	0x0
25	GPIO46_LEVEL_HIGH		RW	0x0
24	GPIO46_LEVEL_LOW		RW	0x0
23	GPIO45_EDGE_HIGH		RW	0x0
22	GPIO45_EDGE_LOW		RW	0x0
21	GPIO45_LEVEL_HIGH		RW	0x0
20	GPIO45_LEVEL_LOW		RW	0x0
19	GPIO44_EDGE_HIGH		RW	0x0
18	GPIO44_EDGE_LOW		RW	0x0
17	GPIO44_LEVEL_HIGH		RW	0x0
16	GPIO44_LEVEL_LOW		RW	0x0
15	GPIO43_EDGE_HIGH		RW	0x0
14	GPIO43_EDGE_LOW		RW	0x0
13	GPIO43_LEVEL_HIGH		RW	0x0
12	GPIO43_LEVEL_LOW		RW	0x0
11	GPIO42_EDGE_HIGH		RW	0x0
10	GPIO42_EDGE_LOW		RW	0x0
9	GPIO42_LEVEL_HIGH		RW	0x0
8	GPIO42_LEVEL_LOW		RW	0x0
7	GPIO41_EDGE_HIGH		RW	0x0
6	GPIO41_EDGE_LOW		RW	0x0
5	GPIO41_LEVEL_HIGH		RW	0x0
4	GPIO41_LEVEL_LOW		RW	0x0
3	GPIO40_EDGE_HIGH		RW	0x0
2	GPIO40_EDGE_LOW		RW	0x0
1	GPIO40_LEVEL_HIGH		RW	0x0
0	GPIO40_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTF0 Register

Offset: 0x2a8

Description

Interrupt Force for proc1

Table 788.
PROC1_INTF0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RW	0x0
30	GPIO7_EDGE_LOW		RW	0x0
29	GPIO7_LEVEL_HIGH		RW	0x0
28	GPIO7_LEVEL_LOW		RW	0x0
27	GPIO6_EDGE_HIGH		RW	0x0
26	GPIO6_EDGE_LOW		RW	0x0
25	GPIO6_LEVEL_HIGH		RW	0x0
24	GPIO6_LEVEL_LOW		RW	0x0
23	GPIO5_EDGE_HIGH		RW	0x0
22	GPIO5_EDGE_LOW		RW	0x0
21	GPIO5_LEVEL_HIGH		RW	0x0
20	GPIO5_LEVEL_LOW		RW	0x0
19	GPIO4_EDGE_HIGH		RW	0x0
18	GPIO4_EDGE_LOW		RW	0x0
17	GPIO4_LEVEL_HIGH		RW	0x0
16	GPIO4_LEVEL_LOW		RW	0x0
15	GPIO3_EDGE_HIGH		RW	0x0
14	GPIO3_EDGE_LOW		RW	0x0
13	GPIO3_LEVEL_HIGH		RW	0x0
12	GPIO3_LEVEL_LOW		RW	0x0
11	GPIO2_EDGE_HIGH		RW	0x0
10	GPIO2_EDGE_LOW		RW	0x0
9	GPIO2_LEVEL_HIGH		RW	0x0
8	GPIO2_LEVEL_LOW		RW	0x0
7	GPIO1_EDGE_HIGH		RW	0x0
6	GPIO1_EDGE_LOW		RW	0x0
5	GPIO1_LEVEL_HIGH		RW	0x0
4	GPIO1_LEVEL_LOW		RW	0x0
3	GPIO0_EDGE_HIGH		RW	0x0
2	GPIO0_EDGE_LOW		RW	0x0
1	GPIO0_LEVEL_HIGH		RW	0x0
0	GPIO0_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTF1 Register

Offset: 0x2ac

Description

Interrupt Force for proc1

Table 789.
PROC1_INTF1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RW	0x0
30	GPIO15_EDGE_LOW		RW	0x0
29	GPIO15_LEVEL_HIGH		RW	0x0
28	GPIO15_LEVEL_LOW		RW	0x0
27	GPIO14_EDGE_HIGH		RW	0x0
26	GPIO14_EDGE_LOW		RW	0x0
25	GPIO14_LEVEL_HIGH		RW	0x0
24	GPIO14_LEVEL_LOW		RW	0x0
23	GPIO13_EDGE_HIGH		RW	0x0
22	GPIO13_EDGE_LOW		RW	0x0
21	GPIO13_LEVEL_HIGH		RW	0x0
20	GPIO13_LEVEL_LOW		RW	0x0
19	GPIO12_EDGE_HIGH		RW	0x0
18	GPIO12_EDGE_LOW		RW	0x0
17	GPIO12_LEVEL_HIGH		RW	0x0
16	GPIO12_LEVEL_LOW		RW	0x0
15	GPIO11_EDGE_HIGH		RW	0x0
14	GPIO11_EDGE_LOW		RW	0x0
13	GPIO11_LEVEL_HIGH		RW	0x0
12	GPIO11_LEVEL_LOW		RW	0x0
11	GPIO10_EDGE_HIGH		RW	0x0
10	GPIO10_EDGE_LOW		RW	0x0
9	GPIO10_LEVEL_HIGH		RW	0x0
8	GPIO10_LEVEL_LOW		RW	0x0
7	GPIO9_EDGE_HIGH		RW	0x0
6	GPIO9_EDGE_LOW		RW	0x0
5	GPIO9_LEVEL_HIGH		RW	0x0
4	GPIO9_LEVEL_LOW		RW	0x0
3	GPIO8_EDGE_HIGH		RW	0x0
2	GPIO8_EDGE_LOW		RW	0x0
1	GPIO8_LEVEL_HIGH		RW	0x0
0	GPIO8_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTF2 Register

Offset: 0x2b0

Description

Interrupt Force for proc1

Table 790.
PROC1_INTF2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RW	0x0
30	GPIO23_EDGE_LOW		RW	0x0
29	GPIO23_LEVEL_HIGH		RW	0x0
28	GPIO23_LEVEL_LOW		RW	0x0
27	GPIO22_EDGE_HIGH		RW	0x0
26	GPIO22_EDGE_LOW		RW	0x0
25	GPIO22_LEVEL_HIGH		RW	0x0
24	GPIO22_LEVEL_LOW		RW	0x0
23	GPIO21_EDGE_HIGH		RW	0x0
22	GPIO21_EDGE_LOW		RW	0x0
21	GPIO21_LEVEL_HIGH		RW	0x0
20	GPIO21_LEVEL_LOW		RW	0x0
19	GPIO20_EDGE_HIGH		RW	0x0
18	GPIO20_EDGE_LOW		RW	0x0
17	GPIO20_LEVEL_HIGH		RW	0x0
16	GPIO20_LEVEL_LOW		RW	0x0
15	GPIO19_EDGE_HIGH		RW	0x0
14	GPIO19_EDGE_LOW		RW	0x0
13	GPIO19_LEVEL_HIGH		RW	0x0
12	GPIO19_LEVEL_LOW		RW	0x0
11	GPIO18_EDGE_HIGH		RW	0x0
10	GPIO18_EDGE_LOW		RW	0x0
9	GPIO18_LEVEL_HIGH		RW	0x0
8	GPIO18_LEVEL_LOW		RW	0x0
7	GPIO17_EDGE_HIGH		RW	0x0
6	GPIO17_EDGE_LOW		RW	0x0
5	GPIO17_LEVEL_HIGH		RW	0x0
4	GPIO17_LEVEL_LOW		RW	0x0
3	GPIO16_EDGE_HIGH		RW	0x0
2	GPIO16_EDGE_LOW		RW	0x0
1	GPIO16_LEVEL_HIGH		RW	0x0
0	GPIO16_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTF3 Register

Offset: 0x2b4

Description

Interrupt Force for proc1

Table 791.
PROC1_INTF3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RW	0x0
30	GPIO31_EDGE_LOW		RW	0x0
29	GPIO31_LEVEL_HIGH		RW	0x0
28	GPIO31_LEVEL_LOW		RW	0x0
27	GPIO30_EDGE_HIGH		RW	0x0
26	GPIO30_EDGE_LOW		RW	0x0
25	GPIO30_LEVEL_HIGH		RW	0x0
24	GPIO30_LEVEL_LOW		RW	0x0
23	GPIO29_EDGE_HIGH		RW	0x0
22	GPIO29_EDGE_LOW		RW	0x0
21	GPIO29_LEVEL_HIGH		RW	0x0
20	GPIO29_LEVEL_LOW		RW	0x0
19	GPIO28_EDGE_HIGH		RW	0x0
18	GPIO28_EDGE_LOW		RW	0x0
17	GPIO28_LEVEL_HIGH		RW	0x0
16	GPIO28_LEVEL_LOW		RW	0x0
15	GPIO27_EDGE_HIGH		RW	0x0
14	GPIO27_EDGE_LOW		RW	0x0
13	GPIO27_LEVEL_HIGH		RW	0x0
12	GPIO27_LEVEL_LOW		RW	0x0
11	GPIO26_EDGE_HIGH		RW	0x0
10	GPIO26_EDGE_LOW		RW	0x0
9	GPIO26_LEVEL_HIGH		RW	0x0
8	GPIO26_LEVEL_LOW		RW	0x0
7	GPIO25_EDGE_HIGH		RW	0x0
6	GPIO25_EDGE_LOW		RW	0x0
5	GPIO25_LEVEL_HIGH		RW	0x0
4	GPIO25_LEVEL_LOW		RW	0x0
3	GPIO24_EDGE_HIGH		RW	0x0
2	GPIO24_EDGE_LOW		RW	0x0
1	GPIO24_LEVEL_HIGH		RW	0x0
0	GPIO24_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTF4 Register

Offset: 0x2b8

Description

Interrupt Force for proc1

Table 792.
PROC1_INTF4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RW	0x0
30	GPIO39_EDGE_LOW		RW	0x0
29	GPIO39_LEVEL_HIGH		RW	0x0
28	GPIO39_LEVEL_LOW		RW	0x0
27	GPIO38_EDGE_HIGH		RW	0x0
26	GPIO38_EDGE_LOW		RW	0x0
25	GPIO38_LEVEL_HIGH		RW	0x0
24	GPIO38_LEVEL_LOW		RW	0x0
23	GPIO37_EDGE_HIGH		RW	0x0
22	GPIO37_EDGE_LOW		RW	0x0
21	GPIO37_LEVEL_HIGH		RW	0x0
20	GPIO37_LEVEL_LOW		RW	0x0
19	GPIO36_EDGE_HIGH		RW	0x0
18	GPIO36_EDGE_LOW		RW	0x0
17	GPIO36_LEVEL_HIGH		RW	0x0
16	GPIO36_LEVEL_LOW		RW	0x0
15	GPIO35_EDGE_HIGH		RW	0x0
14	GPIO35_EDGE_LOW		RW	0x0
13	GPIO35_LEVEL_HIGH		RW	0x0
12	GPIO35_LEVEL_LOW		RW	0x0
11	GPIO34_EDGE_HIGH		RW	0x0
10	GPIO34_EDGE_LOW		RW	0x0
9	GPIO34_LEVEL_HIGH		RW	0x0
8	GPIO34_LEVEL_LOW		RW	0x0
7	GPIO33_EDGE_HIGH		RW	0x0
6	GPIO33_EDGE_LOW		RW	0x0
5	GPIO33_LEVEL_HIGH		RW	0x0
4	GPIO33_LEVEL_LOW		RW	0x0
3	GPIO32_EDGE_HIGH		RW	0x0
2	GPIO32_EDGE_LOW		RW	0x0
1	GPIO32_LEVEL_HIGH		RW	0x0
0	GPIO32_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTF5 Register

Offset: 0x2bc

Description

Interrupt Force for proc1

Table 793.
PROC1_INTE5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RW	0x0
30	GPIO47_EDGE_LOW		RW	0x0
29	GPIO47_LEVEL_HIGH		RW	0x0
28	GPIO47_LEVEL_LOW		RW	0x0
27	GPIO46_EDGE_HIGH		RW	0x0
26	GPIO46_EDGE_LOW		RW	0x0
25	GPIO46_LEVEL_HIGH		RW	0x0
24	GPIO46_LEVEL_LOW		RW	0x0
23	GPIO45_EDGE_HIGH		RW	0x0
22	GPIO45_EDGE_LOW		RW	0x0
21	GPIO45_LEVEL_HIGH		RW	0x0
20	GPIO45_LEVEL_LOW		RW	0x0
19	GPIO44_EDGE_HIGH		RW	0x0
18	GPIO44_EDGE_LOW		RW	0x0
17	GPIO44_LEVEL_HIGH		RW	0x0
16	GPIO44_LEVEL_LOW		RW	0x0
15	GPIO43_EDGE_HIGH		RW	0x0
14	GPIO43_EDGE_LOW		RW	0x0
13	GPIO43_LEVEL_HIGH		RW	0x0
12	GPIO43_LEVEL_LOW		RW	0x0
11	GPIO42_EDGE_HIGH		RW	0x0
10	GPIO42_EDGE_LOW		RW	0x0
9	GPIO42_LEVEL_HIGH		RW	0x0
8	GPIO42_LEVEL_LOW		RW	0x0
7	GPIO41_EDGE_HIGH		RW	0x0
6	GPIO41_EDGE_LOW		RW	0x0
5	GPIO41_LEVEL_HIGH		RW	0x0
4	GPIO41_LEVEL_LOW		RW	0x0
3	GPIO40_EDGE_HIGH		RW	0x0
2	GPIO40_EDGE_LOW		RW	0x0
1	GPIO40_LEVEL_HIGH		RW	0x0
0	GPIO40_LEVEL_LOW		RW	0x0

IO_BANK0: PROC1_INTS0 Register

Offset: 0x2c0

Description

Interrupt status after masking & forcing for proc1

Table 794.
PROC1_INTS0
Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RO	0x0
30	GPIO7_EDGE_LOW		RO	0x0
29	GPIO7_LEVEL_HIGH		RO	0x0
28	GPIO7_LEVEL_LOW		RO	0x0
27	GPIO6_EDGE_HIGH		RO	0x0
26	GPIO6_EDGE_LOW		RO	0x0
25	GPIO6_LEVEL_HIGH		RO	0x0
24	GPIO6_LEVEL_LOW		RO	0x0
23	GPIO5_EDGE_HIGH		RO	0x0
22	GPIO5_EDGE_LOW		RO	0x0
21	GPIO5_LEVEL_HIGH		RO	0x0
20	GPIO5_LEVEL_LOW		RO	0x0
19	GPIO4_EDGE_HIGH		RO	0x0
18	GPIO4_EDGE_LOW		RO	0x0
17	GPIO4_LEVEL_HIGH		RO	0x0
16	GPIO4_LEVEL_LOW		RO	0x0
15	GPIO3_EDGE_HIGH		RO	0x0
14	GPIO3_EDGE_LOW		RO	0x0
13	GPIO3_LEVEL_HIGH		RO	0x0
12	GPIO3_LEVEL_LOW		RO	0x0
11	GPIO2_EDGE_HIGH		RO	0x0
10	GPIO2_EDGE_LOW		RO	0x0
9	GPIO2_LEVEL_HIGH		RO	0x0
8	GPIO2_LEVEL_LOW		RO	0x0
7	GPIO1_EDGE_HIGH		RO	0x0
6	GPIO1_EDGE_LOW		RO	0x0
5	GPIO1_LEVEL_HIGH		RO	0x0
4	GPIO1_LEVEL_LOW		RO	0x0
3	GPIO0_EDGE_HIGH		RO	0x0
2	GPIO0_EDGE_LOW		RO	0x0
1	GPIO0_LEVEL_HIGH		RO	0x0
0	GPIO0_LEVEL_LOW		RO	0x0

IO_BANK0: PROC1_INTS1 Register

Offset: 0x2c4

Description

Interrupt status after masking & forcing for proc1

Table 795.
PROC1_INTS1
Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RO	0x0
30	GPIO15_EDGE_LOW		RO	0x0
29	GPIO15_LEVEL_HIGH		RO	0x0
28	GPIO15_LEVEL_LOW		RO	0x0
27	GPIO14_EDGE_HIGH		RO	0x0
26	GPIO14_EDGE_LOW		RO	0x0
25	GPIO14_LEVEL_HIGH		RO	0x0
24	GPIO14_LEVEL_LOW		RO	0x0
23	GPIO13_EDGE_HIGH		RO	0x0
22	GPIO13_EDGE_LOW		RO	0x0
21	GPIO13_LEVEL_HIGH		RO	0x0
20	GPIO13_LEVEL_LOW		RO	0x0
19	GPIO12_EDGE_HIGH		RO	0x0
18	GPIO12_EDGE_LOW		RO	0x0
17	GPIO12_LEVEL_HIGH		RO	0x0
16	GPIO12_LEVEL_LOW		RO	0x0
15	GPIO11_EDGE_HIGH		RO	0x0
14	GPIO11_EDGE_LOW		RO	0x0
13	GPIO11_LEVEL_HIGH		RO	0x0
12	GPIO11_LEVEL_LOW		RO	0x0
11	GPIO10_EDGE_HIGH		RO	0x0
10	GPIO10_EDGE_LOW		RO	0x0
9	GPIO10_LEVEL_HIGH		RO	0x0
8	GPIO10_LEVEL_LOW		RO	0x0
7	GPIO9_EDGE_HIGH		RO	0x0
6	GPIO9_EDGE_LOW		RO	0x0
5	GPIO9_LEVEL_HIGH		RO	0x0
4	GPIO9_LEVEL_LOW		RO	0x0
3	GPIO8_EDGE_HIGH		RO	0x0
2	GPIO8_EDGE_LOW		RO	0x0
1	GPIO8_LEVEL_HIGH		RO	0x0
0	GPIO8_LEVEL_LOW		RO	0x0

IO_BANK0: PROC1_INTS2 Register

Offset: 0x2c8

Description

Interrupt status after masking & forcing for proc1

Table 796.
PROC1_INTS2
Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RO	0x0
30	GPIO23_EDGE_LOW		RO	0x0
29	GPIO23_LEVEL_HIGH		RO	0x0
28	GPIO23_LEVEL_LOW		RO	0x0
27	GPIO22_EDGE_HIGH		RO	0x0
26	GPIO22_EDGE_LOW		RO	0x0
25	GPIO22_LEVEL_HIGH		RO	0x0
24	GPIO22_LEVEL_LOW		RO	0x0
23	GPIO21_EDGE_HIGH		RO	0x0
22	GPIO21_EDGE_LOW		RO	0x0
21	GPIO21_LEVEL_HIGH		RO	0x0
20	GPIO21_LEVEL_LOW		RO	0x0
19	GPIO20_EDGE_HIGH		RO	0x0
18	GPIO20_EDGE_LOW		RO	0x0
17	GPIO20_LEVEL_HIGH		RO	0x0
16	GPIO20_LEVEL_LOW		RO	0x0
15	GPIO19_EDGE_HIGH		RO	0x0
14	GPIO19_EDGE_LOW		RO	0x0
13	GPIO19_LEVEL_HIGH		RO	0x0
12	GPIO19_LEVEL_LOW		RO	0x0
11	GPIO18_EDGE_HIGH		RO	0x0
10	GPIO18_EDGE_LOW		RO	0x0
9	GPIO18_LEVEL_HIGH		RO	0x0
8	GPIO18_LEVEL_LOW		RO	0x0
7	GPIO17_EDGE_HIGH		RO	0x0
6	GPIO17_EDGE_LOW		RO	0x0
5	GPIO17_LEVEL_HIGH		RO	0x0
4	GPIO17_LEVEL_LOW		RO	0x0
3	GPIO16_EDGE_HIGH		RO	0x0
2	GPIO16_EDGE_LOW		RO	0x0
1	GPIO16_LEVEL_HIGH		RO	0x0
0	GPIO16_LEVEL_LOW		RO	0x0

IO_BANK0: PROC1_INTS3 Register

Offset: 0x2cc

Description

Interrupt status after masking & forcing for proc1

Table 797.
PROC1_INTS3
Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RO	0x0
30	GPIO31_EDGE_LOW		RO	0x0
29	GPIO31_LEVEL_HIGH		RO	0x0
28	GPIO31_LEVEL_LOW		RO	0x0
27	GPIO30_EDGE_HIGH		RO	0x0
26	GPIO30_EDGE_LOW		RO	0x0
25	GPIO30_LEVEL_HIGH		RO	0x0
24	GPIO30_LEVEL_LOW		RO	0x0
23	GPIO29_EDGE_HIGH		RO	0x0
22	GPIO29_EDGE_LOW		RO	0x0
21	GPIO29_LEVEL_HIGH		RO	0x0
20	GPIO29_LEVEL_LOW		RO	0x0
19	GPIO28_EDGE_HIGH		RO	0x0
18	GPIO28_EDGE_LOW		RO	0x0
17	GPIO28_LEVEL_HIGH		RO	0x0
16	GPIO28_LEVEL_LOW		RO	0x0
15	GPIO27_EDGE_HIGH		RO	0x0
14	GPIO27_EDGE_LOW		RO	0x0
13	GPIO27_LEVEL_HIGH		RO	0x0
12	GPIO27_LEVEL_LOW		RO	0x0
11	GPIO26_EDGE_HIGH		RO	0x0
10	GPIO26_EDGE_LOW		RO	0x0
9	GPIO26_LEVEL_HIGH		RO	0x0
8	GPIO26_LEVEL_LOW		RO	0x0
7	GPIO25_EDGE_HIGH		RO	0x0
6	GPIO25_EDGE_LOW		RO	0x0
5	GPIO25_LEVEL_HIGH		RO	0x0
4	GPIO25_LEVEL_LOW		RO	0x0
3	GPIO24_EDGE_HIGH		RO	0x0
2	GPIO24_EDGE_LOW		RO	0x0
1	GPIO24_LEVEL_HIGH		RO	0x0
0	GPIO24_LEVEL_LOW		RO	0x0

IO_BANK0: PROC1_INTS4 Register

Offset: 0x2d0

Description

Interrupt status after masking & forcing for proc1

Table 798.
PROC1_INTS4
Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RO	0x0
30	GPIO39_EDGE_LOW		RO	0x0
29	GPIO39_LEVEL_HIGH		RO	0x0
28	GPIO39_LEVEL_LOW		RO	0x0
27	GPIO38_EDGE_HIGH		RO	0x0
26	GPIO38_EDGE_LOW		RO	0x0
25	GPIO38_LEVEL_HIGH		RO	0x0
24	GPIO38_LEVEL_LOW		RO	0x0
23	GPIO37_EDGE_HIGH		RO	0x0
22	GPIO37_EDGE_LOW		RO	0x0
21	GPIO37_LEVEL_HIGH		RO	0x0
20	GPIO37_LEVEL_LOW		RO	0x0
19	GPIO36_EDGE_HIGH		RO	0x0
18	GPIO36_EDGE_LOW		RO	0x0
17	GPIO36_LEVEL_HIGH		RO	0x0
16	GPIO36_LEVEL_LOW		RO	0x0
15	GPIO35_EDGE_HIGH		RO	0x0
14	GPIO35_EDGE_LOW		RO	0x0
13	GPIO35_LEVEL_HIGH		RO	0x0
12	GPIO35_LEVEL_LOW		RO	0x0
11	GPIO34_EDGE_HIGH		RO	0x0
10	GPIO34_EDGE_LOW		RO	0x0
9	GPIO34_LEVEL_HIGH		RO	0x0
8	GPIO34_LEVEL_LOW		RO	0x0
7	GPIO33_EDGE_HIGH		RO	0x0
6	GPIO33_EDGE_LOW		RO	0x0
5	GPIO33_LEVEL_HIGH		RO	0x0
4	GPIO33_LEVEL_LOW		RO	0x0
3	GPIO32_EDGE_HIGH		RO	0x0
2	GPIO32_EDGE_LOW		RO	0x0
1	GPIO32_LEVEL_HIGH		RO	0x0
0	GPIO32_LEVEL_LOW		RO	0x0

IO_BANK0: PROC1_INTS5 Register

Offset: 0x2d4

Description

Interrupt status after masking & forcing for proc1

Table 799.
PROC1_INTS5
Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RO	0x0
30	GPIO47_EDGE_LOW		RO	0x0
29	GPIO47_LEVEL_HIGH		RO	0x0
28	GPIO47_LEVEL_LOW		RO	0x0
27	GPIO46_EDGE_HIGH		RO	0x0
26	GPIO46_EDGE_LOW		RO	0x0
25	GPIO46_LEVEL_HIGH		RO	0x0
24	GPIO46_LEVEL_LOW		RO	0x0
23	GPIO45_EDGE_HIGH		RO	0x0
22	GPIO45_EDGE_LOW		RO	0x0
21	GPIO45_LEVEL_HIGH		RO	0x0
20	GPIO45_LEVEL_LOW		RO	0x0
19	GPIO44_EDGE_HIGH		RO	0x0
18	GPIO44_EDGE_LOW		RO	0x0
17	GPIO44_LEVEL_HIGH		RO	0x0
16	GPIO44_LEVEL_LOW		RO	0x0
15	GPIO43_EDGE_HIGH		RO	0x0
14	GPIO43_EDGE_LOW		RO	0x0
13	GPIO43_LEVEL_HIGH		RO	0x0
12	GPIO43_LEVEL_LOW		RO	0x0
11	GPIO42_EDGE_HIGH		RO	0x0
10	GPIO42_EDGE_LOW		RO	0x0
9	GPIO42_LEVEL_HIGH		RO	0x0
8	GPIO42_LEVEL_LOW		RO	0x0
7	GPIO41_EDGE_HIGH		RO	0x0
6	GPIO41_EDGE_LOW		RO	0x0
5	GPIO41_LEVEL_HIGH		RO	0x0
4	GPIO41_LEVEL_LOW		RO	0x0
3	GPIO40_EDGE_HIGH		RO	0x0
2	GPIO40_EDGE_LOW		RO	0x0
1	GPIO40_LEVEL_HIGH		RO	0x0
0	GPIO40_LEVEL_LOW		RO	0x0

IO_BANK0: DORMANT_WAKE_INTE0 Register

Offset: 0x2d8

Description

Interrupt Enable for dormant_wake

Table 800.
DORMANT_WAKE_INT
E0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RW	0x0
30	GPIO7_EDGE_LOW		RW	0x0
29	GPIO7_LEVEL_HIGH		RW	0x0
28	GPIO7_LEVEL_LOW		RW	0x0
27	GPIO6_EDGE_HIGH		RW	0x0
26	GPIO6_EDGE_LOW		RW	0x0
25	GPIO6_LEVEL_HIGH		RW	0x0
24	GPIO6_LEVEL_LOW		RW	0x0
23	GPIO5_EDGE_HIGH		RW	0x0
22	GPIO5_EDGE_LOW		RW	0x0
21	GPIO5_LEVEL_HIGH		RW	0x0
20	GPIO5_LEVEL_LOW		RW	0x0
19	GPIO4_EDGE_HIGH		RW	0x0
18	GPIO4_EDGE_LOW		RW	0x0
17	GPIO4_LEVEL_HIGH		RW	0x0
16	GPIO4_LEVEL_LOW		RW	0x0
15	GPIO3_EDGE_HIGH		RW	0x0
14	GPIO3_EDGE_LOW		RW	0x0
13	GPIO3_LEVEL_HIGH		RW	0x0
12	GPIO3_LEVEL_LOW		RW	0x0
11	GPIO2_EDGE_HIGH		RW	0x0
10	GPIO2_EDGE_LOW		RW	0x0
9	GPIO2_LEVEL_HIGH		RW	0x0
8	GPIO2_LEVEL_LOW		RW	0x0
7	GPIO1_EDGE_HIGH		RW	0x0
6	GPIO1_EDGE_LOW		RW	0x0
5	GPIO1_LEVEL_HIGH		RW	0x0
4	GPIO1_LEVEL_LOW		RW	0x0
3	GPIO0_EDGE_HIGH		RW	0x0
2	GPIO0_EDGE_LOW		RW	0x0
1	GPIO0_LEVEL_HIGH		RW	0x0
0	GPIO0_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTE1 Register

Offset: 0x2dc

Description

Interrupt Enable for dormant_wake

Table 801.
DORMANT_WAKE_INT
E1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RW	0x0
30	GPIO15_EDGE_LOW		RW	0x0
29	GPIO15_LEVEL_HIGH		RW	0x0
28	GPIO15_LEVEL_LOW		RW	0x0
27	GPIO14_EDGE_HIGH		RW	0x0
26	GPIO14_EDGE_LOW		RW	0x0
25	GPIO14_LEVEL_HIGH		RW	0x0
24	GPIO14_LEVEL_LOW		RW	0x0
23	GPIO13_EDGE_HIGH		RW	0x0
22	GPIO13_EDGE_LOW		RW	0x0
21	GPIO13_LEVEL_HIGH		RW	0x0
20	GPIO13_LEVEL_LOW		RW	0x0
19	GPIO12_EDGE_HIGH		RW	0x0
18	GPIO12_EDGE_LOW		RW	0x0
17	GPIO12_LEVEL_HIGH		RW	0x0
16	GPIO12_LEVEL_LOW		RW	0x0
15	GPIO11_EDGE_HIGH		RW	0x0
14	GPIO11_EDGE_LOW		RW	0x0
13	GPIO11_LEVEL_HIGH		RW	0x0
12	GPIO11_LEVEL_LOW		RW	0x0
11	GPIO10_EDGE_HIGH		RW	0x0
10	GPIO10_EDGE_LOW		RW	0x0
9	GPIO10_LEVEL_HIGH		RW	0x0
8	GPIO10_LEVEL_LOW		RW	0x0
7	GPIO9_EDGE_HIGH		RW	0x0
6	GPIO9_EDGE_LOW		RW	0x0
5	GPIO9_LEVEL_HIGH		RW	0x0
4	GPIO9_LEVEL_LOW		RW	0x0
3	GPIO8_EDGE_HIGH		RW	0x0
2	GPIO8_EDGE_LOW		RW	0x0
1	GPIO8_LEVEL_HIGH		RW	0x0
0	GPIO8_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTE2 Register

Offset: 0x2e0

Description

Interrupt Enable for dormant_wake

Table 802.
DORMANT_WAKE_INT
E2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RW	0x0
30	GPIO23_EDGE_LOW		RW	0x0
29	GPIO23_LEVEL_HIGH		RW	0x0
28	GPIO23_LEVEL_LOW		RW	0x0
27	GPIO22_EDGE_HIGH		RW	0x0
26	GPIO22_EDGE_LOW		RW	0x0
25	GPIO22_LEVEL_HIGH		RW	0x0
24	GPIO22_LEVEL_LOW		RW	0x0
23	GPIO21_EDGE_HIGH		RW	0x0
22	GPIO21_EDGE_LOW		RW	0x0
21	GPIO21_LEVEL_HIGH		RW	0x0
20	GPIO21_LEVEL_LOW		RW	0x0
19	GPIO20_EDGE_HIGH		RW	0x0
18	GPIO20_EDGE_LOW		RW	0x0
17	GPIO20_LEVEL_HIGH		RW	0x0
16	GPIO20_LEVEL_LOW		RW	0x0
15	GPIO19_EDGE_HIGH		RW	0x0
14	GPIO19_EDGE_LOW		RW	0x0
13	GPIO19_LEVEL_HIGH		RW	0x0
12	GPIO19_LEVEL_LOW		RW	0x0
11	GPIO18_EDGE_HIGH		RW	0x0
10	GPIO18_EDGE_LOW		RW	0x0
9	GPIO18_LEVEL_HIGH		RW	0x0
8	GPIO18_LEVEL_LOW		RW	0x0
7	GPIO17_EDGE_HIGH		RW	0x0
6	GPIO17_EDGE_LOW		RW	0x0
5	GPIO17_LEVEL_HIGH		RW	0x0
4	GPIO17_LEVEL_LOW		RW	0x0
3	GPIO16_EDGE_HIGH		RW	0x0
2	GPIO16_EDGE_LOW		RW	0x0
1	GPIO16_LEVEL_HIGH		RW	0x0
0	GPIO16_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTE3 Register

Offset: 0x2e4

Description

Interrupt Enable for dormant_wake

Table 803.
DORMANT_WAKE_INT
E3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RW	0x0
30	GPIO31_EDGE_LOW		RW	0x0
29	GPIO31_LEVEL_HIGH		RW	0x0
28	GPIO31_LEVEL_LOW		RW	0x0
27	GPIO30_EDGE_HIGH		RW	0x0
26	GPIO30_EDGE_LOW		RW	0x0
25	GPIO30_LEVEL_HIGH		RW	0x0
24	GPIO30_LEVEL_LOW		RW	0x0
23	GPIO29_EDGE_HIGH		RW	0x0
22	GPIO29_EDGE_LOW		RW	0x0
21	GPIO29_LEVEL_HIGH		RW	0x0
20	GPIO29_LEVEL_LOW		RW	0x0
19	GPIO28_EDGE_HIGH		RW	0x0
18	GPIO28_EDGE_LOW		RW	0x0
17	GPIO28_LEVEL_HIGH		RW	0x0
16	GPIO28_LEVEL_LOW		RW	0x0
15	GPIO27_EDGE_HIGH		RW	0x0
14	GPIO27_EDGE_LOW		RW	0x0
13	GPIO27_LEVEL_HIGH		RW	0x0
12	GPIO27_LEVEL_LOW		RW	0x0
11	GPIO26_EDGE_HIGH		RW	0x0
10	GPIO26_EDGE_LOW		RW	0x0
9	GPIO26_LEVEL_HIGH		RW	0x0
8	GPIO26_LEVEL_LOW		RW	0x0
7	GPIO25_EDGE_HIGH		RW	0x0
6	GPIO25_EDGE_LOW		RW	0x0
5	GPIO25_LEVEL_HIGH		RW	0x0
4	GPIO25_LEVEL_LOW		RW	0x0
3	GPIO24_EDGE_HIGH		RW	0x0
2	GPIO24_EDGE_LOW		RW	0x0
1	GPIO24_LEVEL_HIGH		RW	0x0
0	GPIO24_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTE4 Register

Offset: 0x2e8

Description

Interrupt Enable for dormant_wake

Table 804.
DORMANT_WAKE_INT
E4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RW	0x0
30	GPIO39_EDGE_LOW		RW	0x0
29	GPIO39_LEVEL_HIGH		RW	0x0
28	GPIO39_LEVEL_LOW		RW	0x0
27	GPIO38_EDGE_HIGH		RW	0x0
26	GPIO38_EDGE_LOW		RW	0x0
25	GPIO38_LEVEL_HIGH		RW	0x0
24	GPIO38_LEVEL_LOW		RW	0x0
23	GPIO37_EDGE_HIGH		RW	0x0
22	GPIO37_EDGE_LOW		RW	0x0
21	GPIO37_LEVEL_HIGH		RW	0x0
20	GPIO37_LEVEL_LOW		RW	0x0
19	GPIO36_EDGE_HIGH		RW	0x0
18	GPIO36_EDGE_LOW		RW	0x0
17	GPIO36_LEVEL_HIGH		RW	0x0
16	GPIO36_LEVEL_LOW		RW	0x0
15	GPIO35_EDGE_HIGH		RW	0x0
14	GPIO35_EDGE_LOW		RW	0x0
13	GPIO35_LEVEL_HIGH		RW	0x0
12	GPIO35_LEVEL_LOW		RW	0x0
11	GPIO34_EDGE_HIGH		RW	0x0
10	GPIO34_EDGE_LOW		RW	0x0
9	GPIO34_LEVEL_HIGH		RW	0x0
8	GPIO34_LEVEL_LOW		RW	0x0
7	GPIO33_EDGE_HIGH		RW	0x0
6	GPIO33_EDGE_LOW		RW	0x0
5	GPIO33_LEVEL_HIGH		RW	0x0
4	GPIO33_LEVEL_LOW		RW	0x0
3	GPIO32_EDGE_HIGH		RW	0x0
2	GPIO32_EDGE_LOW		RW	0x0
1	GPIO32_LEVEL_HIGH		RW	0x0
0	GPIO32_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTE5 Register

Offset: 0x2ec

Description

Interrupt Enable for dormant_wake

Table 805.
DORMANT_WAKE_INT
E5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RW	0x0
30	GPIO47_EDGE_LOW		RW	0x0
29	GPIO47_LEVEL_HIGH		RW	0x0
28	GPIO47_LEVEL_LOW		RW	0x0
27	GPIO46_EDGE_HIGH		RW	0x0
26	GPIO46_EDGE_LOW		RW	0x0
25	GPIO46_LEVEL_HIGH		RW	0x0
24	GPIO46_LEVEL_LOW		RW	0x0
23	GPIO45_EDGE_HIGH		RW	0x0
22	GPIO45_EDGE_LOW		RW	0x0
21	GPIO45_LEVEL_HIGH		RW	0x0
20	GPIO45_LEVEL_LOW		RW	0x0
19	GPIO44_EDGE_HIGH		RW	0x0
18	GPIO44_EDGE_LOW		RW	0x0
17	GPIO44_LEVEL_HIGH		RW	0x0
16	GPIO44_LEVEL_LOW		RW	0x0
15	GPIO43_EDGE_HIGH		RW	0x0
14	GPIO43_EDGE_LOW		RW	0x0
13	GPIO43_LEVEL_HIGH		RW	0x0
12	GPIO43_LEVEL_LOW		RW	0x0
11	GPIO42_EDGE_HIGH		RW	0x0
10	GPIO42_EDGE_LOW		RW	0x0
9	GPIO42_LEVEL_HIGH		RW	0x0
8	GPIO42_LEVEL_LOW		RW	0x0
7	GPIO41_EDGE_HIGH		RW	0x0
6	GPIO41_EDGE_LOW		RW	0x0
5	GPIO41_LEVEL_HIGH		RW	0x0
4	GPIO41_LEVEL_LOW		RW	0x0
3	GPIO40_EDGE_HIGH		RW	0x0
2	GPIO40_EDGE_LOW		RW	0x0
1	GPIO40_LEVEL_HIGH		RW	0x0
0	GPIO40_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTF0 Register

Offset: 0x2f0

Description

Interrupt Force for dormant_wake

Table 806.
DORMANT_WAKE_INT
F0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RW	0x0
30	GPIO7_EDGE_LOW		RW	0x0
29	GPIO7_LEVEL_HIGH		RW	0x0
28	GPIO7_LEVEL_LOW		RW	0x0
27	GPIO6_EDGE_HIGH		RW	0x0
26	GPIO6_EDGE_LOW		RW	0x0
25	GPIO6_LEVEL_HIGH		RW	0x0
24	GPIO6_LEVEL_LOW		RW	0x0
23	GPIO5_EDGE_HIGH		RW	0x0
22	GPIO5_EDGE_LOW		RW	0x0
21	GPIO5_LEVEL_HIGH		RW	0x0
20	GPIO5_LEVEL_LOW		RW	0x0
19	GPIO4_EDGE_HIGH		RW	0x0
18	GPIO4_EDGE_LOW		RW	0x0
17	GPIO4_LEVEL_HIGH		RW	0x0
16	GPIO4_LEVEL_LOW		RW	0x0
15	GPIO3_EDGE_HIGH		RW	0x0
14	GPIO3_EDGE_LOW		RW	0x0
13	GPIO3_LEVEL_HIGH		RW	0x0
12	GPIO3_LEVEL_LOW		RW	0x0
11	GPIO2_EDGE_HIGH		RW	0x0
10	GPIO2_EDGE_LOW		RW	0x0
9	GPIO2_LEVEL_HIGH		RW	0x0
8	GPIO2_LEVEL_LOW		RW	0x0
7	GPIO1_EDGE_HIGH		RW	0x0
6	GPIO1_EDGE_LOW		RW	0x0
5	GPIO1_LEVEL_HIGH		RW	0x0
4	GPIO1_LEVEL_LOW		RW	0x0
3	GPIO0_EDGE_HIGH		RW	0x0
2	GPIO0_EDGE_LOW		RW	0x0
1	GPIO0_LEVEL_HIGH		RW	0x0
0	GPIO0_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTF1 Register

Offset: 0x2f4

Description

Interrupt Force for dormant_wake

Table 807.
DORMANT_WAKE_INT
F1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RW	0x0
30	GPIO15_EDGE_LOW		RW	0x0
29	GPIO15_LEVEL_HIGH		RW	0x0
28	GPIO15_LEVEL_LOW		RW	0x0
27	GPIO14_EDGE_HIGH		RW	0x0
26	GPIO14_EDGE_LOW		RW	0x0
25	GPIO14_LEVEL_HIGH		RW	0x0
24	GPIO14_LEVEL_LOW		RW	0x0
23	GPIO13_EDGE_HIGH		RW	0x0
22	GPIO13_EDGE_LOW		RW	0x0
21	GPIO13_LEVEL_HIGH		RW	0x0
20	GPIO13_LEVEL_LOW		RW	0x0
19	GPIO12_EDGE_HIGH		RW	0x0
18	GPIO12_EDGE_LOW		RW	0x0
17	GPIO12_LEVEL_HIGH		RW	0x0
16	GPIO12_LEVEL_LOW		RW	0x0
15	GPIO11_EDGE_HIGH		RW	0x0
14	GPIO11_EDGE_LOW		RW	0x0
13	GPIO11_LEVEL_HIGH		RW	0x0
12	GPIO11_LEVEL_LOW		RW	0x0
11	GPIO10_EDGE_HIGH		RW	0x0
10	GPIO10_EDGE_LOW		RW	0x0
9	GPIO10_LEVEL_HIGH		RW	0x0
8	GPIO10_LEVEL_LOW		RW	0x0
7	GPIO9_EDGE_HIGH		RW	0x0
6	GPIO9_EDGE_LOW		RW	0x0
5	GPIO9_LEVEL_HIGH		RW	0x0
4	GPIO9_LEVEL_LOW		RW	0x0
3	GPIO8_EDGE_HIGH		RW	0x0
2	GPIO8_EDGE_LOW		RW	0x0
1	GPIO8_LEVEL_HIGH		RW	0x0
0	GPIO8_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTF2 Register

Offset: 0x2f8

Description

Interrupt Force for dormant_wake

Table 808.
DORMANT_WAKE_INT
F2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RW	0x0
30	GPIO23_EDGE_LOW		RW	0x0
29	GPIO23_LEVEL_HIGH		RW	0x0
28	GPIO23_LEVEL_LOW		RW	0x0
27	GPIO22_EDGE_HIGH		RW	0x0
26	GPIO22_EDGE_LOW		RW	0x0
25	GPIO22_LEVEL_HIGH		RW	0x0
24	GPIO22_LEVEL_LOW		RW	0x0
23	GPIO21_EDGE_HIGH		RW	0x0
22	GPIO21_EDGE_LOW		RW	0x0
21	GPIO21_LEVEL_HIGH		RW	0x0
20	GPIO21_LEVEL_LOW		RW	0x0
19	GPIO20_EDGE_HIGH		RW	0x0
18	GPIO20_EDGE_LOW		RW	0x0
17	GPIO20_LEVEL_HIGH		RW	0x0
16	GPIO20_LEVEL_LOW		RW	0x0
15	GPIO19_EDGE_HIGH		RW	0x0
14	GPIO19_EDGE_LOW		RW	0x0
13	GPIO19_LEVEL_HIGH		RW	0x0
12	GPIO19_LEVEL_LOW		RW	0x0
11	GPIO18_EDGE_HIGH		RW	0x0
10	GPIO18_EDGE_LOW		RW	0x0
9	GPIO18_LEVEL_HIGH		RW	0x0
8	GPIO18_LEVEL_LOW		RW	0x0
7	GPIO17_EDGE_HIGH		RW	0x0
6	GPIO17_EDGE_LOW		RW	0x0
5	GPIO17_LEVEL_HIGH		RW	0x0
4	GPIO17_LEVEL_LOW		RW	0x0
3	GPIO16_EDGE_HIGH		RW	0x0
2	GPIO16_EDGE_LOW		RW	0x0
1	GPIO16_LEVEL_HIGH		RW	0x0
0	GPIO16_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTF3 Register

Offset: 0x2fc

Description

Interrupt Force for dormant_wake

Table 809.
DORMANT_WAKE_INT
F3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RW	0x0
30	GPIO31_EDGE_LOW		RW	0x0
29	GPIO31_LEVEL_HIGH		RW	0x0
28	GPIO31_LEVEL_LOW		RW	0x0
27	GPIO30_EDGE_HIGH		RW	0x0
26	GPIO30_EDGE_LOW		RW	0x0
25	GPIO30_LEVEL_HIGH		RW	0x0
24	GPIO30_LEVEL_LOW		RW	0x0
23	GPIO29_EDGE_HIGH		RW	0x0
22	GPIO29_EDGE_LOW		RW	0x0
21	GPIO29_LEVEL_HIGH		RW	0x0
20	GPIO29_LEVEL_LOW		RW	0x0
19	GPIO28_EDGE_HIGH		RW	0x0
18	GPIO28_EDGE_LOW		RW	0x0
17	GPIO28_LEVEL_HIGH		RW	0x0
16	GPIO28_LEVEL_LOW		RW	0x0
15	GPIO27_EDGE_HIGH		RW	0x0
14	GPIO27_EDGE_LOW		RW	0x0
13	GPIO27_LEVEL_HIGH		RW	0x0
12	GPIO27_LEVEL_LOW		RW	0x0
11	GPIO26_EDGE_HIGH		RW	0x0
10	GPIO26_EDGE_LOW		RW	0x0
9	GPIO26_LEVEL_HIGH		RW	0x0
8	GPIO26_LEVEL_LOW		RW	0x0
7	GPIO25_EDGE_HIGH		RW	0x0
6	GPIO25_EDGE_LOW		RW	0x0
5	GPIO25_LEVEL_HIGH		RW	0x0
4	GPIO25_LEVEL_LOW		RW	0x0
3	GPIO24_EDGE_HIGH		RW	0x0
2	GPIO24_EDGE_LOW		RW	0x0
1	GPIO24_LEVEL_HIGH		RW	0x0
0	GPIO24_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTF4 Register

Offset: 0x300

Description

Interrupt Force for dormant_wake

Table 810.
DORMANT_WAKE_INT
F4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RW	0x0
30	GPIO39_EDGE_LOW		RW	0x0
29	GPIO39_LEVEL_HIGH		RW	0x0
28	GPIO39_LEVEL_LOW		RW	0x0
27	GPIO38_EDGE_HIGH		RW	0x0
26	GPIO38_EDGE_LOW		RW	0x0
25	GPIO38_LEVEL_HIGH		RW	0x0
24	GPIO38_LEVEL_LOW		RW	0x0
23	GPIO37_EDGE_HIGH		RW	0x0
22	GPIO37_EDGE_LOW		RW	0x0
21	GPIO37_LEVEL_HIGH		RW	0x0
20	GPIO37_LEVEL_LOW		RW	0x0
19	GPIO36_EDGE_HIGH		RW	0x0
18	GPIO36_EDGE_LOW		RW	0x0
17	GPIO36_LEVEL_HIGH		RW	0x0
16	GPIO36_LEVEL_LOW		RW	0x0
15	GPIO35_EDGE_HIGH		RW	0x0
14	GPIO35_EDGE_LOW		RW	0x0
13	GPIO35_LEVEL_HIGH		RW	0x0
12	GPIO35_LEVEL_LOW		RW	0x0
11	GPIO34_EDGE_HIGH		RW	0x0
10	GPIO34_EDGE_LOW		RW	0x0
9	GPIO34_LEVEL_HIGH		RW	0x0
8	GPIO34_LEVEL_LOW		RW	0x0
7	GPIO33_EDGE_HIGH		RW	0x0
6	GPIO33_EDGE_LOW		RW	0x0
5	GPIO33_LEVEL_HIGH		RW	0x0
4	GPIO33_LEVEL_LOW		RW	0x0
3	GPIO32_EDGE_HIGH		RW	0x0
2	GPIO32_EDGE_LOW		RW	0x0
1	GPIO32_LEVEL_HIGH		RW	0x0
0	GPIO32_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTF5 Register

Offset: 0x304

Description

Interrupt Force for dormant_wake

Table 811.
DORMANT_WAKE_INT
F5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RW	0x0
30	GPIO47_EDGE_LOW		RW	0x0
29	GPIO47_LEVEL_HIGH		RW	0x0
28	GPIO47_LEVEL_LOW		RW	0x0
27	GPIO46_EDGE_HIGH		RW	0x0
26	GPIO46_EDGE_LOW		RW	0x0
25	GPIO46_LEVEL_HIGH		RW	0x0
24	GPIO46_LEVEL_LOW		RW	0x0
23	GPIO45_EDGE_HIGH		RW	0x0
22	GPIO45_EDGE_LOW		RW	0x0
21	GPIO45_LEVEL_HIGH		RW	0x0
20	GPIO45_LEVEL_LOW		RW	0x0
19	GPIO44_EDGE_HIGH		RW	0x0
18	GPIO44_EDGE_LOW		RW	0x0
17	GPIO44_LEVEL_HIGH		RW	0x0
16	GPIO44_LEVEL_LOW		RW	0x0
15	GPIO43_EDGE_HIGH		RW	0x0
14	GPIO43_EDGE_LOW		RW	0x0
13	GPIO43_LEVEL_HIGH		RW	0x0
12	GPIO43_LEVEL_LOW		RW	0x0
11	GPIO42_EDGE_HIGH		RW	0x0
10	GPIO42_EDGE_LOW		RW	0x0
9	GPIO42_LEVEL_HIGH		RW	0x0
8	GPIO42_LEVEL_LOW		RW	0x0
7	GPIO41_EDGE_HIGH		RW	0x0
6	GPIO41_EDGE_LOW		RW	0x0
5	GPIO41_LEVEL_HIGH		RW	0x0
4	GPIO41_LEVEL_LOW		RW	0x0
3	GPIO40_EDGE_HIGH		RW	0x0
2	GPIO40_EDGE_LOW		RW	0x0
1	GPIO40_LEVEL_HIGH		RW	0x0
0	GPIO40_LEVEL_LOW		RW	0x0

IO_BANK0: DORMANT_WAKE_INTS0 Register

Offset: 0x308

Description

Interrupt status after masking & forcing for dormant_wake

Table 812.
DORMANT_WAKE_INT
S0 Register

Bits	Name	Description	Type	Reset
31	GPIO7_EDGE_HIGH		RO	0x0
30	GPIO7_EDGE_LOW		RO	0x0
29	GPIO7_LEVEL_HIGH		RO	0x0
28	GPIO7_LEVEL_LOW		RO	0x0
27	GPIO6_EDGE_HIGH		RO	0x0
26	GPIO6_EDGE_LOW		RO	0x0
25	GPIO6_LEVEL_HIGH		RO	0x0
24	GPIO6_LEVEL_LOW		RO	0x0
23	GPIO5_EDGE_HIGH		RO	0x0
22	GPIO5_EDGE_LOW		RO	0x0
21	GPIO5_LEVEL_HIGH		RO	0x0
20	GPIO5_LEVEL_LOW		RO	0x0
19	GPIO4_EDGE_HIGH		RO	0x0
18	GPIO4_EDGE_LOW		RO	0x0
17	GPIO4_LEVEL_HIGH		RO	0x0
16	GPIO4_LEVEL_LOW		RO	0x0
15	GPIO3_EDGE_HIGH		RO	0x0
14	GPIO3_EDGE_LOW		RO	0x0
13	GPIO3_LEVEL_HIGH		RO	0x0
12	GPIO3_LEVEL_LOW		RO	0x0
11	GPIO2_EDGE_HIGH		RO	0x0
10	GPIO2_EDGE_LOW		RO	0x0
9	GPIO2_LEVEL_HIGH		RO	0x0
8	GPIO2_LEVEL_LOW		RO	0x0
7	GPIO1_EDGE_HIGH		RO	0x0
6	GPIO1_EDGE_LOW		RO	0x0
5	GPIO1_LEVEL_HIGH		RO	0x0
4	GPIO1_LEVEL_LOW		RO	0x0
3	GPIO0_EDGE_HIGH		RO	0x0
2	GPIO0_EDGE_LOW		RO	0x0
1	GPIO0_LEVEL_HIGH		RO	0x0
0	GPIO0_LEVEL_LOW		RO	0x0

IO_BANK0: DORMANT_WAKE_INTS1 Register

Offset: 0x30c

Description

Interrupt status after masking & forcing for dormant_wake

Table 813.
DORMANT_WAKE_INT
S1 Register

Bits	Name	Description	Type	Reset
31	GPIO15_EDGE_HIGH		RO	0x0
30	GPIO15_EDGE_LOW		RO	0x0
29	GPIO15_LEVEL_HIGH		RO	0x0
28	GPIO15_LEVEL_LOW		RO	0x0
27	GPIO14_EDGE_HIGH		RO	0x0
26	GPIO14_EDGE_LOW		RO	0x0
25	GPIO14_LEVEL_HIGH		RO	0x0
24	GPIO14_LEVEL_LOW		RO	0x0
23	GPIO13_EDGE_HIGH		RO	0x0
22	GPIO13_EDGE_LOW		RO	0x0
21	GPIO13_LEVEL_HIGH		RO	0x0
20	GPIO13_LEVEL_LOW		RO	0x0
19	GPIO12_EDGE_HIGH		RO	0x0
18	GPIO12_EDGE_LOW		RO	0x0
17	GPIO12_LEVEL_HIGH		RO	0x0
16	GPIO12_LEVEL_LOW		RO	0x0
15	GPIO11_EDGE_HIGH		RO	0x0
14	GPIO11_EDGE_LOW		RO	0x0
13	GPIO11_LEVEL_HIGH		RO	0x0
12	GPIO11_LEVEL_LOW		RO	0x0
11	GPIO10_EDGE_HIGH		RO	0x0
10	GPIO10_EDGE_LOW		RO	0x0
9	GPIO10_LEVEL_HIGH		RO	0x0
8	GPIO10_LEVEL_LOW		RO	0x0
7	GPIO9_EDGE_HIGH		RO	0x0
6	GPIO9_EDGE_LOW		RO	0x0
5	GPIO9_LEVEL_HIGH		RO	0x0
4	GPIO9_LEVEL_LOW		RO	0x0
3	GPIO8_EDGE_HIGH		RO	0x0
2	GPIO8_EDGE_LOW		RO	0x0
1	GPIO8_LEVEL_HIGH		RO	0x0
0	GPIO8_LEVEL_LOW		RO	0x0

IO_BANK0: DORMANT_WAKE_INTS2 Register

Offset: 0x310

Description

Interrupt status after masking & forcing for dormant_wake

Table 814.
DORMANT_WAKE_INT
S2 Register

Bits	Name	Description	Type	Reset
31	GPIO23_EDGE_HIGH		RO	0x0
30	GPIO23_EDGE_LOW		RO	0x0
29	GPIO23_LEVEL_HIGH		RO	0x0
28	GPIO23_LEVEL_LOW		RO	0x0
27	GPIO22_EDGE_HIGH		RO	0x0
26	GPIO22_EDGE_LOW		RO	0x0
25	GPIO22_LEVEL_HIGH		RO	0x0
24	GPIO22_LEVEL_LOW		RO	0x0
23	GPIO21_EDGE_HIGH		RO	0x0
22	GPIO21_EDGE_LOW		RO	0x0
21	GPIO21_LEVEL_HIGH		RO	0x0
20	GPIO21_LEVEL_LOW		RO	0x0
19	GPIO20_EDGE_HIGH		RO	0x0
18	GPIO20_EDGE_LOW		RO	0x0
17	GPIO20_LEVEL_HIGH		RO	0x0
16	GPIO20_LEVEL_LOW		RO	0x0
15	GPIO19_EDGE_HIGH		RO	0x0
14	GPIO19_EDGE_LOW		RO	0x0
13	GPIO19_LEVEL_HIGH		RO	0x0
12	GPIO19_LEVEL_LOW		RO	0x0
11	GPIO18_EDGE_HIGH		RO	0x0
10	GPIO18_EDGE_LOW		RO	0x0
9	GPIO18_LEVEL_HIGH		RO	0x0
8	GPIO18_LEVEL_LOW		RO	0x0
7	GPIO17_EDGE_HIGH		RO	0x0
6	GPIO17_EDGE_LOW		RO	0x0
5	GPIO17_LEVEL_HIGH		RO	0x0
4	GPIO17_LEVEL_LOW		RO	0x0
3	GPIO16_EDGE_HIGH		RO	0x0
2	GPIO16_EDGE_LOW		RO	0x0
1	GPIO16_LEVEL_HIGH		RO	0x0
0	GPIO16_LEVEL_LOW		RO	0x0

IO_BANK0: DORMANT_WAKE_INTS3 Register

Offset: 0x314

Description

Interrupt status after masking & forcing for dormant_wake

Table 815.
DORMANT_WAKE_INT
S3 Register

Bits	Name	Description	Type	Reset
31	GPIO31_EDGE_HIGH		RO	0x0
30	GPIO31_EDGE_LOW		RO	0x0
29	GPIO31_LEVEL_HIGH		RO	0x0
28	GPIO31_LEVEL_LOW		RO	0x0
27	GPIO30_EDGE_HIGH		RO	0x0
26	GPIO30_EDGE_LOW		RO	0x0
25	GPIO30_LEVEL_HIGH		RO	0x0
24	GPIO30_LEVEL_LOW		RO	0x0
23	GPIO29_EDGE_HIGH		RO	0x0
22	GPIO29_EDGE_LOW		RO	0x0
21	GPIO29_LEVEL_HIGH		RO	0x0
20	GPIO29_LEVEL_LOW		RO	0x0
19	GPIO28_EDGE_HIGH		RO	0x0
18	GPIO28_EDGE_LOW		RO	0x0
17	GPIO28_LEVEL_HIGH		RO	0x0
16	GPIO28_LEVEL_LOW		RO	0x0
15	GPIO27_EDGE_HIGH		RO	0x0
14	GPIO27_EDGE_LOW		RO	0x0
13	GPIO27_LEVEL_HIGH		RO	0x0
12	GPIO27_LEVEL_LOW		RO	0x0
11	GPIO26_EDGE_HIGH		RO	0x0
10	GPIO26_EDGE_LOW		RO	0x0
9	GPIO26_LEVEL_HIGH		RO	0x0
8	GPIO26_LEVEL_LOW		RO	0x0
7	GPIO25_EDGE_HIGH		RO	0x0
6	GPIO25_EDGE_LOW		RO	0x0
5	GPIO25_LEVEL_HIGH		RO	0x0
4	GPIO25_LEVEL_LOW		RO	0x0
3	GPIO24_EDGE_HIGH		RO	0x0
2	GPIO24_EDGE_LOW		RO	0x0
1	GPIO24_LEVEL_HIGH		RO	0x0
0	GPIO24_LEVEL_LOW		RO	0x0

IO_BANK0: DORMANT_WAKE_INTS4 Register

Offset: 0x318

Description

Interrupt status after masking & forcing for dormant_wake

Table 816.
DORMANT_WAKE_INT
S4 Register

Bits	Name	Description	Type	Reset
31	GPIO39_EDGE_HIGH		RO	0x0
30	GPIO39_EDGE_LOW		RO	0x0
29	GPIO39_LEVEL_HIGH		RO	0x0
28	GPIO39_LEVEL_LOW		RO	0x0
27	GPIO38_EDGE_HIGH		RO	0x0
26	GPIO38_EDGE_LOW		RO	0x0
25	GPIO38_LEVEL_HIGH		RO	0x0
24	GPIO38_LEVEL_LOW		RO	0x0
23	GPIO37_EDGE_HIGH		RO	0x0
22	GPIO37_EDGE_LOW		RO	0x0
21	GPIO37_LEVEL_HIGH		RO	0x0
20	GPIO37_LEVEL_LOW		RO	0x0
19	GPIO36_EDGE_HIGH		RO	0x0
18	GPIO36_EDGE_LOW		RO	0x0
17	GPIO36_LEVEL_HIGH		RO	0x0
16	GPIO36_LEVEL_LOW		RO	0x0
15	GPIO35_EDGE_HIGH		RO	0x0
14	GPIO35_EDGE_LOW		RO	0x0
13	GPIO35_LEVEL_HIGH		RO	0x0
12	GPIO35_LEVEL_LOW		RO	0x0
11	GPIO34_EDGE_HIGH		RO	0x0
10	GPIO34_EDGE_LOW		RO	0x0
9	GPIO34_LEVEL_HIGH		RO	0x0
8	GPIO34_LEVEL_LOW		RO	0x0
7	GPIO33_EDGE_HIGH		RO	0x0
6	GPIO33_EDGE_LOW		RO	0x0
5	GPIO33_LEVEL_HIGH		RO	0x0
4	GPIO33_LEVEL_LOW		RO	0x0
3	GPIO32_EDGE_HIGH		RO	0x0
2	GPIO32_EDGE_LOW		RO	0x0
1	GPIO32_LEVEL_HIGH		RO	0x0
0	GPIO32_LEVEL_LOW		RO	0x0

IO_BANK0: DORMANT_WAKE_INTS5 Register

Offset: 0x31c

Description

Interrupt status after masking & forcing for dormant_wake

Table 817.
DORMANT_WAKE_INT
S5 Register

Bits	Name	Description	Type	Reset
31	GPIO47_EDGE_HIGH		RO	0x0
30	GPIO47_EDGE_LOW		RO	0x0
29	GPIO47_LEVEL_HIGH		RO	0x0
28	GPIO47_LEVEL_LOW		RO	0x0
27	GPIO46_EDGE_HIGH		RO	0x0
26	GPIO46_EDGE_LOW		RO	0x0
25	GPIO46_LEVEL_HIGH		RO	0x0
24	GPIO46_LEVEL_LOW		RO	0x0
23	GPIO45_EDGE_HIGH		RO	0x0
22	GPIO45_EDGE_LOW		RO	0x0
21	GPIO45_LEVEL_HIGH		RO	0x0
20	GPIO45_LEVEL_LOW		RO	0x0
19	GPIO44_EDGE_HIGH		RO	0x0
18	GPIO44_EDGE_LOW		RO	0x0
17	GPIO44_LEVEL_HIGH		RO	0x0
16	GPIO44_LEVEL_LOW		RO	0x0
15	GPIO43_EDGE_HIGH		RO	0x0
14	GPIO43_EDGE_LOW		RO	0x0
13	GPIO43_LEVEL_HIGH		RO	0x0
12	GPIO43_LEVEL_LOW		RO	0x0
11	GPIO42_EDGE_HIGH		RO	0x0
10	GPIO42_EDGE_LOW		RO	0x0
9	GPIO42_LEVEL_HIGH		RO	0x0
8	GPIO42_LEVEL_LOW		RO	0x0
7	GPIO41_EDGE_HIGH		RO	0x0
6	GPIO41_EDGE_LOW		RO	0x0
5	GPIO41_LEVEL_HIGH		RO	0x0
4	GPIO41_LEVEL_LOW		RO	0x0
3	GPIO40_EDGE_HIGH		RO	0x0
2	GPIO40_EDGE_LOW		RO	0x0
1	GPIO40_LEVEL_HIGH		RO	0x0
0	GPIO40_LEVEL_LOW		RO	0x0

9.11.2. IO - QSPI Bank

The QSPI Bank IO registers start at a base address of `0x40030000` (defined as `IO_QSPI_BASE` in SDK).

Table 818. List of IO_QSPI registers

Offset	Name	Info
0x000	USBPHY_DP_STATUS	
0x004	USBPHY_DP_CTRL	
0x008	USBPHY_DM_STATUS	
0x00c	USBPHY_DM_CTRL	
0x010	GPIO_QSPI_SCLK_STATUS	
0x014	GPIO_QSPI_SCLK_CTRL	
0x018	GPIO_QSPI_SS_STATUS	
0x01c	GPIO_QSPI_SS_CTRL	
0x020	GPIO_QSPI_SD0_STATUS	
0x024	GPIO_QSPI_SD0_CTRL	
0x028	GPIO_QSPI_SD1_STATUS	
0x02c	GPIO_QSPI_SD1_CTRL	
0x030	GPIO_QSPI_SD2_STATUS	
0x034	GPIO_QSPI_SD2_CTRL	
0x038	GPIO_QSPI_SD3_STATUS	
0x03c	GPIO_QSPI_SD3_CTRL	
0x200	IRQSUMMARY_PROC0_SECURE	
0x204	IRQSUMMARY_PROC0_NONSECURE	
0x208	IRQSUMMARY_PROC1_SECURE	
0x20c	IRQSUMMARY_PROC1_NONSECURE	
0x210	IRQSUMMARY_COMA_WAKE_SECURE	
0x214	IRQSUMMARY_COMA_WAKE_NONSECURE	
0x218	INTR	Raw Interrupts
0x21c	PROC0_INTE	Interrupt Enable for proc0
0x220	PROC0_INTF	Interrupt Force for proc0
0x224	PROC0_INTS	Interrupt status after masking & forcing for proc0
0x228	PROC1_INTE	Interrupt Enable for proc1
0x22c	PROC1_INTF	Interrupt Force for proc1
0x230	PROC1_INTS	Interrupt status after masking & forcing for proc1
0x234	DORMANT_WAKE_INTE	Interrupt Enable for dormant_wake
0x238	DORMANT_WAKE_INTF	Interrupt Force for dormant_wake
0x23c	DORMANT_WAKE_INTS	Interrupt status after masking & forcing for dormant_wake

IO_QSPI: USBPHY_DP_STATUS Register

Offset: 0x000

Table 819.
USBPHY_DP_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: USBPHY_DP_CTRL Register

Offset: 0x004

Table 820.
USBPHY_DP_CTRL
Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x02 → uart1_tx 0x03 → i2c0_sda 0x05 → sio_56 0x1f → null	RW	0x1f

IO_QSPI: USBPHY_DM_STATUS Register

Offset: 0x008

Table 821.
USBPHY_DM_STATUS
Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: USBPHY_DM_CTRL Register

Offset: 0x00c

Table 822.
USBPHY_DM_CTRL
Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0

Bits	Name	Description	Type	Reset
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x02 → uart1_rx 0x03 → i2c0_scl 0x05 → sio_57 0x1f → null	RW	0x1f

IO_QSPI: GPIO_QSPI_SCLK_STATUS Register

Offset: 0x010

Table 823.
GPIO_QSPI_SCLK_STA
TUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: GPIO_QSPI_SCLK_CTRL Register

Offset: 0x014

Table 824.
GPIO_QSPI_SCLK_CTR
L Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0

Bits	Name	Description	Type	Reset
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → xip_sclk 0x02 → uart1_cts 0x03 → i2c1_sda 0x05 → sio_58 0x0b → uart1_tx 0x1f → null	RW	0x1f

IO_QSPI: GPIO_QSPI_SS_STATUS Register

Offset: 0x018

Table 825.
GPIO_QSPI_SS_STATUS Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: GPIO_QSPI_SS_CTRL Register

Offset: 0x01c

Table 826.
GPIO_QSPI_SS_CTRL Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → xip_ss_n_0 0x02 → uart1_rts 0x03 → i2c1_scl 0x05 → sio_59 0x0b → uart1_rx 0x1f → null	RW	0x1f

IO_QSPI: GPIO_QSPI_SD0_STATUS Register

Offset: 0x020

Table 827.
GPIO_QSPI_SD0_STAT
US Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register override is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: GPIO_QSPI_SD0_CTRL Register

Offset: 0x024

Table 828.
GPIO_QSPI_SD0_CTRL
Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → xip_sd0 0x02 → uart0_tx 0x03 → i2c0_sda 0x05 → sio_60 0x1f → null	RW	0x1f

IO_QSPI: GPIO_QSPI_SD1_STATUS Register

Offset: 0x028

Table 829.
GPIO_QSPI_SD1_STAT
US Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
9	OUTTOPAD	output signal to pad after register override is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: GPIO_QSPI_SD1_CTRL Register

Offset: 0x02c

Table 830.
GPIO_QSPI_SD1_CTRL
Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → xip_sd1 0x02 → uart0_rx 0x03 → i2c0_scl 0x05 → sio_61 0x1f → null	RW	0x1f

IO_QSPI: GPIO_QSPI_SD2_STATUS Register

Offset: 0x030

Table 831.
GPIO_QSPI_SD2_STAT
US Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: GPIO_QSPI_SD2_CTRL Register

Offset: 0x034

Table 832.
GPIO_QSPI_SD2_CTRL
Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → xip_sd2 0x02 → uart0_cts 0x03 → i2c1_sda 0x05 → sio_62 0x0b → uart0_tx 0x1f → null	RW	0x1f

IO_QSPI: GPIO_QSPI_SD3_STATUS Register

Offset: 0x038

Table 833.
GPIO_QSPI_SD3_STAT
US Register

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	IRQTOPROC	interrupt to processors, after override is applied	RO	0x0
25:18	Reserved.	-	-	-
17	INFROMPAD	input signal from pad, before filtering and override are applied	RO	0x0
16:14	Reserved.	-	-	-
13	OETOPAD	output enable to pad after register overide is applied	RO	0x0
12:10	Reserved.	-	-	-
9	OUTTOPAD	output signal to pad after register overide is applied	RO	0x0
8:0	Reserved.	-	-	-

IO_QSPI: GPIO_QSPI_SD3_CTRL Register

Offset: 0x03c

Table 834.
GPIO_QSPI_SD3_CTRL
Register

Bits	Name	Description	Type	Reset
31:30	Reserved.	-	-	-
29:28	IRQOVER	0x0 → don't invert the interrupt 0x1 → invert the interrupt 0x2 → drive interrupt low 0x3 → drive interrupt high	RW	0x0
27:18	Reserved.	-	-	-
17:16	INOVER	0x0 → don't invert the peri input 0x1 → invert the peri input 0x2 → drive peri input low 0x3 → drive peri input high	RW	0x0
15:14	OEOVER	0x0 → drive output enable from peripheral signal selected by funcsel 0x1 → drive output enable from inverse of peripheral signal selected by funcsel 0x2 → disable output 0x3 → enable output	RW	0x0
13:12	OUTOVER	0x0 → drive output from peripheral signal selected by funcsel 0x1 → drive output from inverse of peripheral signal selected by funcsel 0x2 → drive output low 0x3 → drive output high	RW	0x0
11:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	FUNCSEL	0-31 → selects pin function according to the gpio table 31 == NULL 0x00 → xip_sd3 0x02 → uart0_rts 0x03 → i2c1_scl 0x05 → sio_63 0x0b → uart0_rx 0x1f → null	RW	0x1f

IO_QSPI: IRQSUMMARY_PROC0_SECURE Register

Offset: 0x200

Table 835.
IRQSUMMARY_PROC0_SECURE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	GPIO_QSPI_SD3		RO	0x0
6	GPIO_QSPI_SD2		RO	0x0
5	GPIO_QSPI_SD1		RO	0x0
4	GPIO_QSPI_SD0		RO	0x0
3	GPIO_QSPI_SS		RO	0x0
2	GPIO_QSPI_SCLK		RO	0x0
1	USBPHY_DM		RO	0x0
0	USBPHY_DP		RO	0x0

IO_QSPI: IRQSUMMARY_PROC0_NONSECURE Register

Offset: 0x204

Table 836.
IRQSUMMARY_PROC0_NONSECURE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	GPIO_QSPI_SD3		RO	0x0
6	GPIO_QSPI_SD2		RO	0x0
5	GPIO_QSPI_SD1		RO	0x0
4	GPIO_QSPI_SD0		RO	0x0
3	GPIO_QSPI_SS		RO	0x0
2	GPIO_QSPI_SCLK		RO	0x0
1	USBPHY_DM		RO	0x0
0	USBPHY_DP		RO	0x0

IO_QSPI: IRQSUMMARY_PROC1_SECURE Register

Offset: 0x208

Table 837.
IRQSUMMARY_PROC1_SECURE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7	GPIO_QSPI_SD3		RO	0x0
6	GPIO_QSPI_SD2		RO	0x0
5	GPIO_QSPI_SD1		RO	0x0
4	GPIO_QSPI_SD0		RO	0x0
3	GPIO_QSPI_SS		RO	0x0
2	GPIO_QSPI_SCLK		RO	0x0
1	USBPHY_DM		RO	0x0
0	USBPHY_DP		RO	0x0

IO_QSPI: IRQSUMMARY_PROC1_NONSECURE Register

Offset: 0x20c

Table 838.
IRQSUMMARY_PROC1_NONSECURE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	GPIO_QSPI_SD3		RO	0x0
6	GPIO_QSPI_SD2		RO	0x0
5	GPIO_QSPI_SD1		RO	0x0
4	GPIO_QSPI_SD0		RO	0x0
3	GPIO_QSPI_SS		RO	0x0
2	GPIO_QSPI_SCLK		RO	0x0
1	USBPHY_DM		RO	0x0
0	USBPHY_DP		RO	0x0

IO_QSPI: IRQSUMMARY_COMA_WAKE_SECURE Register

Offset: 0x210

Table 839.
IRQSUMMARY_COMA_WAKE_SECURE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	GPIO_QSPI_SD3		RO	0x0
6	GPIO_QSPI_SD2		RO	0x0
5	GPIO_QSPI_SD1		RO	0x0
4	GPIO_QSPI_SD0		RO	0x0
3	GPIO_QSPI_SS		RO	0x0
2	GPIO_QSPI_SCLK		RO	0x0
1	USBPHY_DM		RO	0x0
0	USBPHY_DP		RO	0x0

IO_QSPI: IRQSUMMARY_COMA_WAKE_NONSECURE Register

Offset: 0x214

Table 840.
IRQSUMMARY_COMA_WAKE_NONSECURE Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	GPIO_QSPI_SD3		RO	0x0
6	GPIO_QSPI_SD2		RO	0x0
5	GPIO_QSPI_SD1		RO	0x0
4	GPIO_QSPI_SD0		RO	0x0
3	GPIO_QSPI_SS		RO	0x0
2	GPIO_QSPI_SCLK		RO	0x0
1	USBPHY_DM		RO	0x0
0	USBPHY_DP		RO	0x0

IO_QSPI: INTR Register

Offset: 0x218

Description

Raw Interrupts

Table 841. INTR Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		WC	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		WC	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RO	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RO	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		WC	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		WC	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RO	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RO	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		WC	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		WC	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RO	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RO	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		WC	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		WC	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RO	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RO	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		WC	0x0
14	GPIO_QSPI_SS_EDGE_LOW		WC	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RO	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RO	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		WC	0x0

Bits	Name	Description	Type	Reset
10	GPIO_QSPI_SCLK_EDGE_LOW		WC	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RO	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RO	0x0
7	USBPHY_DM_EDGE_HIGH		WC	0x0
6	USBPHY_DM_EDGE_LOW		WC	0x0
5	USBPHY_DM_LEVEL_HIGH		RO	0x0
4	USBPHY_DM_LEVEL_LOW		RO	0x0
3	USBPHY_DP_EDGE_HIGH		WC	0x0
2	USBPHY_DP_EDGE_LOW		WC	0x0
1	USBPHY_DP_LEVEL_HIGH		RO	0x0
0	USBPHY_DP_LEVEL_LOW		RO	0x0

IO_QSPI: PROC0_INTE Register

Offset: 0x21c

Description

Interrupt Enable for proc0

Table 842.
PROC0_INTE Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RW	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RW	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RW	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RW	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RW	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RW	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RW	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RW	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RW	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RW	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RW	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RW	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RW	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RW	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RW	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RW	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RW	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RW	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RW	0x0

Bits	Name	Description	Type	Reset
12	GPIO_QSPI_SS_LEVEL_LOW		RW	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RW	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RW	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RW	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RW	0x0
7	USBPHY_DM_EDGE_HIGH		RW	0x0
6	USBPHY_DM_EDGE_LOW		RW	0x0
5	USBPHY_DM_LEVEL_HIGH		RW	0x0
4	USBPHY_DM_LEVEL_LOW		RW	0x0
3	USBPHY_DP_EDGE_HIGH		RW	0x0
2	USBPHY_DP_EDGE_LOW		RW	0x0
1	USBPHY_DP_LEVEL_HIGH		RW	0x0
0	USBPHY_DP_LEVEL_LOW		RW	0x0

IO_QSPI: PROC0_INTF Register

Offset: 0x220

Description

Interrupt Force for proc0

Table 843.
PROC0_INTF Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RW	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RW	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RW	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RW	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RW	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RW	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RW	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RW	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RW	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RW	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RW	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RW	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RW	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RW	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RW	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RW	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RW	0x0

Bits	Name	Description	Type	Reset
14	GPIO_QSPI_SS_EDGE_LOW		RW	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RW	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RW	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RW	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RW	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RW	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RW	0x0
7	USBPHY_DM_EDGE_HIGH		RW	0x0
6	USBPHY_DM_EDGE_LOW		RW	0x0
5	USBPHY_DM_LEVEL_HIGH		RW	0x0
4	USBPHY_DM_LEVEL_LOW		RW	0x0
3	USBPHY_DP_EDGE_HIGH		RW	0x0
2	USBPHY_DP_EDGE_LOW		RW	0x0
1	USBPHY_DP_LEVEL_HIGH		RW	0x0
0	USBPHY_DP_LEVEL_LOW		RW	0x0

IO_QSPI: PROC0_INTS Register

Offset: 0x224

Description

Interrupt status after masking & forcing for proc0

Table 844.
PROC0_INTS Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RO	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RO	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RO	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RO	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RO	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RO	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RO	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RO	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RO	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RO	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RO	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RO	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RO	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RO	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RO	0x0

Bits	Name	Description	Type	Reset
16	GPIO_QSPI_SD0_LEVEL_LOW		RO	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RO	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RO	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RO	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RO	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RO	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RO	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RO	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RO	0x0
7	USBPHY_DM_EDGE_HIGH		RO	0x0
6	USBPHY_DM_EDGE_LOW		RO	0x0
5	USBPHY_DM_LEVEL_HIGH		RO	0x0
4	USBPHY_DM_LEVEL_LOW		RO	0x0
3	USBPHY_DP_EDGE_HIGH		RO	0x0
2	USBPHY_DP_EDGE_LOW		RO	0x0
1	USBPHY_DP_LEVEL_HIGH		RO	0x0
0	USBPHY_DP_LEVEL_LOW		RO	0x0

IO_QSPI: PROC1_INTE Register

Offset: 0x228

Description

Interrupt Enable for proc1

Table 845.
PROC1_INTE Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RW	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RW	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RW	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RW	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RW	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RW	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RW	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RW	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RW	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RW	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RW	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RW	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RW	0x0

Bits	Name	Description	Type	Reset
18	GPIO_QSPI_SD0_EDGE_LOW		RW	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RW	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RW	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RW	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RW	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RW	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RW	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RW	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RW	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RW	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RW	0x0
7	USBPHY_DM_EDGE_HIGH		RW	0x0
6	USBPHY_DM_EDGE_LOW		RW	0x0
5	USBPHY_DM_LEVEL_HIGH		RW	0x0
4	USBPHY_DM_LEVEL_LOW		RW	0x0
3	USBPHY_DP_EDGE_HIGH		RW	0x0
2	USBPHY_DP_EDGE_LOW		RW	0x0
1	USBPHY_DP_LEVEL_HIGH		RW	0x0
0	USBPHY_DP_LEVEL_LOW		RW	0x0

IO_QSPI: PROC1_INTF Register

Offset: 0x22c

Description

Interrupt Force for proc1

Table 846.
PROC1_INTF Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RW	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RW	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RW	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RW	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RW	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RW	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RW	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RW	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RW	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RW	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RW	0x0

Bits	Name	Description	Type	Reset
20	GPIO_QSPI_SD1_LEVEL_LOW		RW	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RW	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RW	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RW	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RW	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RW	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RW	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RW	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RW	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RW	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RW	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RW	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RW	0x0
7	USBPHY_DM_EDGE_HIGH		RW	0x0
6	USBPHY_DM_EDGE_LOW		RW	0x0
5	USBPHY_DM_LEVEL_HIGH		RW	0x0
4	USBPHY_DM_LEVEL_LOW		RW	0x0
3	USBPHY_DP_EDGE_HIGH		RW	0x0
2	USBPHY_DP_EDGE_LOW		RW	0x0
1	USBPHY_DP_LEVEL_HIGH		RW	0x0
0	USBPHY_DP_LEVEL_LOW		RW	0x0

IO_QSPI: PROC1_INTS Register

Offset: 0x230

Description

Interrupt status after masking & forcing for proc1

Table 847.
PROC1_INTS Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RO	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RO	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RO	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RO	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RO	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RO	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RO	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RO	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RO	0x0

Bits	Name	Description	Type	Reset
22	GPIO_QSPI_SD1_EDGE_LOW		RO	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RO	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RO	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RO	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RO	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RO	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RO	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RO	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RO	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RO	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RO	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RO	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RO	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RO	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RO	0x0
7	USBPHY_DM_EDGE_HIGH		RO	0x0
6	USBPHY_DM_EDGE_LOW		RO	0x0
5	USBPHY_DM_LEVEL_HIGH		RO	0x0
4	USBPHY_DM_LEVEL_LOW		RO	0x0
3	USBPHY_DP_EDGE_HIGH		RO	0x0
2	USBPHY_DP_EDGE_LOW		RO	0x0
1	USBPHY_DP_LEVEL_HIGH		RO	0x0
0	USBPHY_DP_LEVEL_LOW		RO	0x0

IO_QSPI: DORMANT_WAKE_INTE Register

Offset: 0x234

Description

Interrupt Enable for dormant_wake

Table 848.
DORMANT_WAKE_INTE Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RW	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RW	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RW	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RW	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RW	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RW	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RW	0x0

Bits	Name	Description	Type	Reset
24	GPIO_QSPI_SD2_LEVEL_LOW		RW	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RW	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RW	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RW	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RW	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RW	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RW	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RW	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RW	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RW	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RW	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RW	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RW	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RW	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RW	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RW	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RW	0x0
7	USBPHY_DM_EDGE_HIGH		RW	0x0
6	USBPHY_DM_EDGE_LOW		RW	0x0
5	USBPHY_DM_LEVEL_HIGH		RW	0x0
4	USBPHY_DM_LEVEL_LOW		RW	0x0
3	USBPHY_DP_EDGE_HIGH		RW	0x0
2	USBPHY_DP_EDGE_LOW		RW	0x0
1	USBPHY_DP_LEVEL_HIGH		RW	0x0
0	USBPHY_DP_LEVEL_LOW		RW	0x0

IO_QSPI: DORMANT_WAKE_INTF Register

Offset: 0x238

Description

Interrupt Force for dormant_wake

Table 849.
DORMANT_WAKE_INTF Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RW	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RW	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RW	0x0
28	GPIO_QSPI_SD3_LEVEL_LOW		RW	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RW	0x0

Bits	Name	Description	Type	Reset
26	GPIO_QSPI_SD2_EDGE_LOW		RW	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RW	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RW	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RW	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RW	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RW	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RW	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RW	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RW	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RW	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RW	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RW	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RW	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RW	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RW	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RW	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RW	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RW	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RW	0x0
7	USBPHY_DM_EDGE_HIGH		RW	0x0
6	USBPHY_DM_EDGE_LOW		RW	0x0
5	USBPHY_DM_LEVEL_HIGH		RW	0x0
4	USBPHY_DM_LEVEL_LOW		RW	0x0
3	USBPHY_DP_EDGE_HIGH		RW	0x0
2	USBPHY_DP_EDGE_LOW		RW	0x0
1	USBPHY_DP_LEVEL_HIGH		RW	0x0
0	USBPHY_DP_LEVEL_LOW		RW	0x0

IO_QSPI: DORMANT_WAKE_INTS Register

Offset: 0x23c

Description

Interrupt status after masking & forcing for dormant_wake

Table 850.
DORMANT_WAKE_INTS Register

Bits	Name	Description	Type	Reset
31	GPIO_QSPI_SD3_EDGE_HIGH		RO	0x0
30	GPIO_QSPI_SD3_EDGE_LOW		RO	0x0
29	GPIO_QSPI_SD3_LEVEL_HIGH		RO	0x0

Bits	Name	Description	Type	Reset
28	GPIO_QSPI_SD3_LEVEL_LOW		RO	0x0
27	GPIO_QSPI_SD2_EDGE_HIGH		RO	0x0
26	GPIO_QSPI_SD2_EDGE_LOW		RO	0x0
25	GPIO_QSPI_SD2_LEVEL_HIGH		RO	0x0
24	GPIO_QSPI_SD2_LEVEL_LOW		RO	0x0
23	GPIO_QSPI_SD1_EDGE_HIGH		RO	0x0
22	GPIO_QSPI_SD1_EDGE_LOW		RO	0x0
21	GPIO_QSPI_SD1_LEVEL_HIGH		RO	0x0
20	GPIO_QSPI_SD1_LEVEL_LOW		RO	0x0
19	GPIO_QSPI_SD0_EDGE_HIGH		RO	0x0
18	GPIO_QSPI_SD0_EDGE_LOW		RO	0x0
17	GPIO_QSPI_SD0_LEVEL_HIGH		RO	0x0
16	GPIO_QSPI_SD0_LEVEL_LOW		RO	0x0
15	GPIO_QSPI_SS_EDGE_HIGH		RO	0x0
14	GPIO_QSPI_SS_EDGE_LOW		RO	0x0
13	GPIO_QSPI_SS_LEVEL_HIGH		RO	0x0
12	GPIO_QSPI_SS_LEVEL_LOW		RO	0x0
11	GPIO_QSPI_SCLK_EDGE_HIGH		RO	0x0
10	GPIO_QSPI_SCLK_EDGE_LOW		RO	0x0
9	GPIO_QSPI_SCLK_LEVEL_HIGH		RO	0x0
8	GPIO_QSPI_SCLK_LEVEL_LOW		RO	0x0
7	USBPHY_DM_EDGE_HIGH		RO	0x0
6	USBPHY_DM_EDGE_LOW		RO	0x0
5	USBPHY_DM_LEVEL_HIGH		RO	0x0
4	USBPHY_DM_LEVEL_LOW		RO	0x0
3	USBPHY_DP_EDGE_HIGH		RO	0x0
2	USBPHY_DP_EDGE_LOW		RO	0x0
1	USBPHY_DP_LEVEL_HIGH		RO	0x0
0	USBPHY_DP_LEVEL_LOW		RO	0x0

9.11.3. Pad Control - User Bank

The User Bank Pad Control registers start at a base address of `0x40038000` (defined as `PADS_BANK0_BASE` in SDK).

Table 851. List of
PADS_BANK0
registers

Offset	Name	Info
0x00	VOLTAGE_SELECT	Voltage select. Per bank control
0x04	GPIO0	

Offset	Name	Info
0x08	GPIO1	
0x0c	GPIO2	
0x10	GPIO3	
0x14	GPIO4	
0x18	GPIO5	
0x1c	GPIO6	
0x20	GPIO7	
0x24	GPIO8	
0x28	GPIO9	
0x2c	GPIO10	
0x30	GPIO11	
0x34	GPIO12	
0x38	GPIO13	
0x3c	GPIO14	
0x40	GPIO15	
0x44	GPIO16	
0x48	GPIO17	
0x4c	GPIO18	
0x50	GPIO19	
0x54	GPIO20	
0x58	GPIO21	
0x5c	GPIO22	
0x60	GPIO23	
0x64	GPIO24	
0x68	GPIO25	
0x6c	GPIO26	
0x70	GPIO27	
0x74	GPIO28	
0x78	GPIO29	
0x7c	GPIO30	
0x80	GPIO31	
0x84	GPIO32	
0x88	GPIO33	
0x8c	GPIO34	
0x90	GPIO35	
0x94	GPIO36	

Offset	Name	Info
0x98	GPIO37	
0x9c	GPIO38	
0xa0	GPIO39	
0xa4	GPIO40	
0xa8	GPIO41	
0xac	GPIO42	
0xb0	GPIO43	
0xb4	GPIO44	
0xb8	GPIO45	
0xbc	GPIO46	
0xc0	GPIO47	
0xc4	SWCLK	
0xc8	SWD	

PADS_BANK0: VOLTAGE_SELECT Register

Offset: 0x00

Table 852.
VOLTAGE_SELECT
Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Voltage select. Per bank control 0x0 → Set voltage to 3.3V (DVDD >= 2V5) 0x1 → Set voltage to 1.8V (DVDD <= 1V8)	RW	0x0

PADS_BANK0: GPIO0 Register

Offset: 0x04

Table 853. GPIO0
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1

Bits	Name	Description	Type	Reset
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO1 Register

Offset: 0x08

Table 854. GPIO1 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO2 Register

Offset: 0x0c

Table 855. GPIO2 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO3 Register

Offset: 0x10

Table 856. GPIO3 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO4 Register

Offset: 0x14

Table 857. GPIO4 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO5 Register

Offset: 0x18

Table 858. GPIO5 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO6 Register

Offset: 0x1c

Table 859. GPIO6 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO7 Register

Offset: 0x20

Table 860. GPIO7 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1

Bits	Name	Description	Type	Reset
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO8 Register

Offset: 0x24

Table 861. GPIO8 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO9 Register

Offset: 0x28

Table 862. GPIO9 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO10 Register

Offset: 0x2c

Table 863. GPIO10 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO11 Register

Offset: 0x30

Table 864. GPIO11 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO12 Register

Offset: 0x34

Table 865. GPIO12 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO13 Register

Offset: 0x38

Table 866. GPIO13 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO14 Register

Offset: 0x3c

Table 867. GPIO14 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO15 Register

Offset: 0x40

Table 868. GPIO15 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO16 Register

Offset: 0x44

Table 869. GPIO16 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO17 Register

Offset: 0x48

Table 870. GPIO17 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO18 Register

Offset: 0x4c

Table 871. GPIO18 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO19 Register

Offset: 0x50

Table 872. GPIO19 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO20 Register

Offset: 0x54

Table 873. GPIO20 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO21 Register

Offset: 0x58

Table 874. GPIO21 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO22 Register

Offset: 0x5c

Table 875. GPIO22 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO23 Register

Offset: 0x60

Table 876. GPIO23 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO24 Register

Offset: 0x64

Table 877. GPIO24 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO25 Register

Offset: 0x68

Table 878. GPIO25 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO26 Register

Offset: 0x6c

Table 879. GPIO26 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO27 Register

Offset: 0x70

Table 880. GPIO27 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO28 Register

Offset: 0x74

Table 881. GPIO28 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO29 Register

Offset: 0x78

Table 882. GPIO29 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO30 Register

Offset: 0x7c

Table 883. GPIO30 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO31 Register

Offset: 0x80

Table 884. GPIO31 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO32 Register

Offset: 0x84

Table 885. GPIO32 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO33 Register

Offset: 0x88

Table 886. GPIO33 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO34 Register

Offset: 0x8c

Table 887. GPIO34 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO35 Register

Offset: 0x90

Table 888. GPIO35 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO36 Register

Offset: 0x94

Table 889. GPIO36 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO37 Register

Offset: 0x98

Table 890. GPIO37 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO38 Register

Offset: 0x9c

Table 891. GPIO38 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO39 Register

Offset: 0xa0

Table 892. GPIO39 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO40 Register

Offset: 0xa4

Table 893. GPIO40 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO41 Register

Offset: 0xa8

Table 894. GPIO41 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO42 Register

Offset: 0xac

Table 895. GPIO42 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO43 Register

Offset: 0xb0

Table 896. GPIO43 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO44 Register

Offset: 0xb4

Table 897. GPIO44 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO45 Register

Offset: 0xb8

Table 898. GPIO45 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO46 Register

Offset: 0xbc

Table 899. GPIO46 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: GPIO47 Register

Offset: 0xc0

Table 900. GPIO47 Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x0

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: SWCLK Register

Offset: 0xc4

Table 901. SWCLK Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x0
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x1
2	PDE	Pull down enable	RW	0x0
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_BANK0: SWD Register

Offset: 0xc8

Table 902. SWD Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x0
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x1
2	PDE	Pull down enable	RW	0x0
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

9.11.4. Pad Control - QSPI Bank

The QSPI Bank Pad Control registers start at a base address of [0x40040000](#) (defined as [PADS_QSPI_BASE](#) in SDK).

Table 903. List of [PADS_QSPI](#) registers

Offset	Name	Info
0x00	VOLTAGE_SELECT	Voltage select. Per bank control
0x04	GPIO_QSPI_SCLK	
0x08	GPIO_QSPI_SD0	
0x0c	GPIO_QSPI_SD1	
0x10	GPIO_QSPI_SD2	
0x14	GPIO_QSPI_SD3	
0x18	GPIO_QSPI_SS	

[PADS_QSPI](#): [VOLTAGE_SELECT](#) Register

Offset: 0x00

Table 904.
[VOLTAGE_SELECT](#)
Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Voltage select. Per bank control 0x0 → Set voltage to 3.3V (DVDD >= 2V5) 0x1 → Set voltage to 1.8V (DVDD <= 1V8)	RW	0x0

[PADS_QSPI](#): [GPIO_QSPI_SCLK](#) Register

Offset: 0x04

Table 905.
[GPIO_QSPI_SCLK](#)
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_QSPI: GPIO_QSPI_SD0 Register

Offset: 0x08

Table 906.
GPIO_QSPI_SD0
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_QSPI: GPIO_QSPI_SD1 Register

Offset: 0x0c

Table 907.
GPIO_QSPI_SD1
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_QSPI: GPIO_QSPI_SD2 Register

Offset: 0x10

Table 908.
GPIO_QSPI_SD2
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x1
2	PDE	Pull down enable	RW	0x0
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_QSPI: GPIO_QSPI_SD3 Register

Offset: 0x14

Table 909.
GPIO_QSPI_SD3
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1

Bits	Name	Description	Type	Reset
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x1
2	PDE	Pull down enable	RW	0x0
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

PADS_QSPI: GPIO_QSPI_SS Register

Offset: 0x18

Table 910.
GPIO_QSPI_SS
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ISO	Pad isolation control. Remove this once the pad is configured by software.	RW	0x1
7	OD	Output disable. Has priority over output enable from peripherals	RW	0x0
6	IE	Input enable	RW	0x1
5:4	DRIVE	Drive strength. 0x0 → 2mA 0x1 → 4mA 0x2 → 8mA 0x3 → 12mA	RW	0x1
3	PUE	Pull up enable	RW	0x1
2	PDE	Pull down enable	RW	0x0
1	SCHMITT	Enable schmitt trigger	RW	0x1
0	SLEWFAST	Slew rate control. 1 = Fast, 0 = Slow	RW	0x0

Chapter 10. Security

This chapter describes the RP2350 security model and the hardware that implements it. Separate overviews are given for Arm and RISC-V architecture modes, because the processor security features and level of bootrom support differ.

10.1. Overview (Arm)

RP2350 provides hardware and bootrom security features for three purposes:

1. Prevent unauthorised code from running on the device
2. Prevent unauthorised reading of user code and data
3. Isolate trusted and untrusted software, running concurrently on the device, from one another

Point one is referred to in this datasheet as *secure boot*, and it is a prerequisite to points two and three, since being able to run unauthorised code on the device would mean its internals could easily be accessed. The bootrom secure boot implementation and related hardware security features implement the root of trust for secure RP2350 applications; bootrom contents is fixed at design time and is completely immutable.

Point two is referred to as *encrypted boot*, and is an additional layer of protection which makes it more difficult to clone devices, or to dump and reverse-engineer their firmware. Encrypted boot is implemented using a signed decryption stage prepended to a binary as a post-build step. Decryption keys are stored in on-device OTP memory, which can be locked down after use.

Point three allows applications to enforce internal security boundaries, such that one part of an application being compromised still does not allow access to critical hardware such as the voltage regulator or protected OTP storage used for cryptographic keys.

Hardware features such as the glitch detector and redundancy coprocessor mitigate common classes of fault injection attacks and help maintain boot integrity even with physical access.

10.1.1. Secure Boot

As the recipient of a blank RP2350 device you can permanently alter it, such that it will only run your code. You can further alter the device to revoke the ability to run older software versions.

The RP2350 bootrom uses a cryptographic signature, based on the SHA-256 hash algorithm and secp256k1 ECDSA elliptic curve cipher, to distinguish authentic from inauthentic binaries:

- Code and data are digested with a SHA-256 algorithm as they are loaded
- The image's *signature* – a hash encrypted with the user's private key, and stored in the image – is decrypted using the user's public key, which is also stored in the image
- The decrypted signature is checked against the actual SHA-256 hash value measured when loading the binary
- The image's public key is checked against a SHA-256 key fingerprint, stored in OTP

If both checks succeed, the bootrom assumes someone in possession of the private key registered by an OTP public key fingerprint calculated the same SHA-256 hash. Based on the properties of hash functions, it further assumes that the binary contents has not been tampered with since the signature was generated. Hence this is an authentic binary blessed by the owner of the private key, and the bootrom will entertain the idea of running it.

The image may also have an anti-rollback version number, which the bootrom checks against a counter stored in OTP. The bootrom refuses to boot images with rollback versions lower than the OTP counter number, and automatically increments the OTP counter upon booting a higher version. This is useful if older binaries have known vulnerabilities: installing a newer version can automatically revoke the ability to downgrade to the older versions. See [Section 5.1.6](#) for

more discussion of bootrom anti-rollback support.

RP2350 can boot from any of the following sources:

- Entering external flash directly via execute-in-place (XIP)
- Loading into SRAM from external flash
- Loading into SRAM from user-specified OTP contents
- Loading into SRAM via USB or other serial bootloader
- Loading into SRAM via debugger

RP2350 enforces signatures on all of these boot media, with the exception of the debugger. An external host has control of RP2350's processors and can completely skip execution of the bootrom. Disabling debug is part of the secure boot enable procedure outlined in [Section 10.4.1](#).

Although signatures *can* be enforced on a flash execute-in-place binary, this is not recommended as the flash contents can change between being checked and being executed, for example if an attacker emulates a QSPI device using an FPGA or another microcontroller. RP2350 has sufficient SRAM capacity to load complete applications into SRAM and verify in-place before execution, which avoids this issue.

Pure-software secure boot implementations are susceptible to fault injection attacks when an attacker has physical access to the device, which is often the case for embedded hardware. Our very own Pico is a popular tool for voltage fault injection. The RP2350 glitch detectors ([Section 10.7](#)) and redundancy coprocessor ([Section 3.6.3](#)) mitigate fault injection attacks by detecting out-of-envelope operation and bringing the system to a safe halt, rather than potentially being manipulated into booting an unauthorised binary. The glitch detectors must be enabled by setting the `CRIT1.GLITCH_DETECTOR_ENABLE` OTP flag, whereas the redundancy coprocessor is used unconditionally by the bootrom.

[Section 10.4.1](#) describes the procedure for enabling secure boot on a blank RP2350 device.

10.1.2. Encrypted Boot

RP2350 contains 8 kB of OTP, which can be protected at 128-byte page granularity. Protection is in the form of hard locks, which permanently revoke read or write access by Secure or Non-secure code, or of soft locks, which revoke permissions only until the next reset of the OTP block. This is the intended mechanism for storing decryption keys used for encrypted boot.

RP2350 supports loading of encrypted binaries from external flash into SRAM, which can then decrypt their own contents in-place. Many implementations are possible, but for a concrete example, this section describes the flash-resident binary encryption support provided by the SDK and [picotool](#). First a plain SRAM binary is processed into an encrypted binary, as a post-build step:

1. The payload binary is signed using the boot private key, if not already signed.
2. The payload binary is encrypted using the encryption key, which is different from the private key.
3. A small *decryption stage* is prepended to the binary, with a modified copy of the payload's `IMAGE_DEF` (the original is unreadable as it is encrypted).
4. The decryption stage is signed together with the encrypted contents, using the boot private key.

An encrypted binary boots as a packaged RAM binary ([Section 5.1.7](#)), and decrypts itself in place:

1. The entire encrypted binary is loaded into SRAM.
2. The bootrom verifies the signature of the decryption stage, then jumps in.
3. The decryption stage reads the decryption key stored in OTP, and may then soft-lock that OTP page until next boot.
4. Using this key, the decryption stage decrypts the encrypted payload.
5. The decryption stage calls the `chain_image()` bootrom API ([Section 5.4.12.2](#)) on the decrypted region of SRAM.

6. The bootrom verifies the decrypted payload in the same manner as it verified the decryption stage, then jumps in.

The decryption stage is not itself encrypted, but it is signed. Storing the decryption stage in the clear does not present additional risk because the source code for the decryption stage is already open source, and highly scrutinised. Without the decryption key, the encrypted payload can not be read: this key only exists on-device, so can not be recovered from static analysis of the encrypted binary.

Resetting the OTP to reopen soft locks also resets the processors. Upon reset the processors will re-run the decryption stage and re-lock the page with the decryption key. The [BOOTDIS](#) register allows the bootrom to detect OTP resets and disable the watchdog and POWMAN boot vectors, ensuring the decryption stage is not skipped, and the key remains protected.

NOTE

The decryption stage is deliberately not included in the bootrom, so that it can be updated. The bootrom handles only public key cryptography, so there is no concern of power analysis attacks, but this reasoning does not apply to the decryption stage. Power analysis mitigations often require multiple iterations as analysis techniques improve.

This scheme supports designs where the decryption key is accessible *only* to the decryption stage. For cases where the decryption key is also used at runtime to read additional encrypted flash contents on-demand, the processor security features and OTP page locks can be used to restrict key access to a small subset of trusted code, such as a TF-M Secure Storage service.

In addition to software mitigations provided by the decryption stage, RP2350 supports randomising the frequency controls of its internal ring oscillator ([Section 8.3](#)) to make it more difficult to recover the system clock from power traces.

Encrypted execute-in-place is not supported in hardware, but the spare 32 MiB cached XIP window ([Section 4.4.1](#)) can be used to provide software-defined execute-in-place by trapping cache misses and then pinning at the miss address. This may be used to decrypt cache lines on-demand from external flash, and have the decrypted content cached transparently by the XIP cache.

10.1.3. Isolating Trusted and Untrusted Software

In security- or safety-critical applications, access must be limited to those who need it. For example, a JPEG decode library should not be able to access the core voltage regulator and increase DVDD to 3.3V, unless perhaps you are decompressing a very large JPEG. The Cortex-M33 processors contain hardware that separates two execution contexts, known as Secure and Non-secure, and enforces a number of invariants between them, such as:

- Non-secure code can not access Secure memory
- The Secure context can not execute Non-secure memory
- Non-secure code can not directly access peripherals managed by Secure code
- Non-secure code can not prevent Secure interrupts from being serviced

By making less of your code able to access your most critical hardware and data, you reduce the chance of accidentally exposing this critical hardware and data to the outside world. How exactly the Cortex-M33 implements this separation is discussed further in [Section 10.2](#), and in full detail in the [Armv8-M Architecture Reference Manual](#).

To make the programming model of Secure and Non-secure software consistent, and to avoid overhead in Non-secure code, RP2350 extends the Secure/Non-secure separation throughout the system. For example, by allowing DMA channels to be assigned for Secure/Non-secure use, Non-secure code can use DMA transfers to accelerate its peripheral accesses, without the possibility of the DMA becoming a vector for breaking one of the security model invariants, such as Non-secure code using the DMA to read Secure memory.

The key hardware features here are:

- The Cortex-M33's implementation of the Secure and Non-secure states ([Section 10.2](#))

- The DMA's implementation of matching security states, per-channel ([Section 10.6](#))
- The system-level bus access filtering implemented by ACCESSCTRL ([Section 10.5](#))
- Peripheral-level filtering, such as the per-GPIO access filtering of the SIO GPIO registers ([Section 3.1.1](#))

10.2. Processor Security Features (Arm)

The Cortex-M33 processors on RP2350 are configured with the following standard Arm security features:

- Support for the Armv8-M Security extension
- Eight security attribution unit (SAU) regions
- Eight Secure and eight Non-secure memory protection unit (MPU) regions

These features are covered exhaustively in the [Armv8-M Architecture Reference Manual](#), the Cortex-M33 Technical Reference Manual, and the Cortex-M33 section of this datasheet ([Section 3.7](#)). This section gives a high-level summary, as well as documenting the implementation-defined attribution unit included in RP2350.

10.2.1. Background

The Cortex-M33 processors on RP2350 are configured to support the Armv8-M Security Extension. Hardware in the processor maintains two separate execution contexts, called the Secure and Non-secure domain. Access to important data, such as cryptographic keys, or hardware, such as the system voltage regulator, can be limited to the Secure domain, and the Non-secure domain is prevented from interfering with Secure execution. When this datasheet writes out Secure and Non-secure with this capitalisation, it is referring to these two Arm security domains, or to the associated bus attributes.

It's not necessarily the case that the author of code running in the Non-secure domain is expected to be malicious – it could be that Non-secure code is running a complex protocol stack like USB, whose implementation is expected to be easily exploited, or just crash fatally, and we would like to isolate some critical software from these effects. The RP2350 bootrom, for example, runs all of its USB code in the Non-secure domain so that this code does not have to be considered in the design of more critical parts of the bootrom such as boot signature enforcement.

An Armv8-M processor implementing the Security Extension remembers at all times whether it is in the Secure or Non-secure execution state. Based on this state, it limits the memory regions it can access via load/store instructions, and the memory regions it can execute. All of the processor's AHB accesses are tagged according to whether they originated from the Secure or the Non-secure context, so that peripherals and the system bus fabric itself can filter transfers based on security domain, for example, using the access control lists described in [Section 10.5](#).

An internal processor peripheral called the Security Attribution Unit (SAU) defines, from the processor's point of view, which address ranges are accessible to the Secure and Non-secure domains. The number of distinct address ranges which can be decoded by the SAU is limited, which is why system-level bus filters are provided for assigning peripherals to security domains.

The processor changes security state synchronously using special function calls from Secure → Non-secure code or vice versa. It can also change security state asynchronously when an interrupt routed to the Secure domain occurs whilst the processor is in the Non-secure state, or (if permitted) vice versa.

Both Cortex-M33 processors on RP2350 implement the security extension, so each processor maintains its own Secure and Non-secure context. The Secure and Non-secure context on each core can communicate with one another, for example using shared memory or the Secure/Non-secure SIO mailbox FIFOs. If the cores are used symmetrically – a shared dual-core Secure context, and a shared dual-core Non-secure context – then the processor SAUs must be kept synchronised, so that memory writable from a Non-secure context on one core is not executable in a Secure context on the other core. The DMA MPU, which supports the same region shape and count as the SAU, must also be kept synchronised with the processor SAUs.

It may be simpler to use the cores asymmetrically, with all Secure services implemented on one core only. The

`FORCE_CORE_NS` register can make all core 1 accesses appear as Non-secure on the system bus, for the purpose of security filtering implemented in the fabric and peripherals, as well as for SIO registers banked over Secure/Non-secure. This does not affect PPB accesses, as the sole exception. Core 1 is not aware this is in effect (other than by explicitly checking the register), and can still maintain its own Secure/Non-secure context internally, but it is considered "the nonsecure core" for the purpose of access to system hardware.

10.2.2. IDAU Address Map

The Cortex-M33 provides an implementation-defined attribution unit (IDAU) interface, which allows system implementers such as Raspberry Pi Ltd to augment the security attribution map defined by the SAU. The RP2350 IDAU is a hardwired address decode network, with no user configuration. Its address map is as follows:

Start (hex)	End (hex)	Contents	IDAU Attribute
00000000	000042ff	Arm boot	Exempt
00004300	00007dff	USB/RISC-V boot	Non-secure (instruction fetch), Exempt (load/store)
00007e00	00007fff	BootROM SGs	Secure and Non-secure-Callable
10000000	1fffffff	XIP	Non-secure
20000000	20081fff	SRAM	Non-secure
40000000	4fffffff	APB	Exempt
50000000	5fffffff	AHB	Exempt
d0000000	dfffffff	SIO	Exempt

Exempt regions are not checked by the processor against its current security state. Effectively, the processor considers them Secure when the processor is in the Secure state, and Non-secure when the processor is in the Non-secure state.

Peripherals are Exempt because you are expected to assign them to security domains using the controls in ACCESSCTRL (Section 10.5), since there are not enough SAU regions to perform meaningful peripheral assignment, and since having separate Secure and Non-secure mirrors of the peripherals is an unnecessary source of programming errors. The SIO is Exempt because it is internally banked over Secure and Non-secure, based on the bus access's security attribute, which generally matches the processor's current security state.

The first part of the bootrom is Exempt, because it contains routines expected to be called by both Secure and Non-secure software in cases where it may not be desirable for Non-secure code to elevate through a Secure Gateway. An example of this is the bootrom `memcpy()` implementation. Code in the Exempt ROM region is hardened against return-oriented programming (ROP) attacks using the redundancy coprocessor's stack canary instructions.

After a certain watermark, which may vary depending on ROM revision, the ROM becomes IDAU-Non-secure for the purpose of instruction fetch. If an Non-secure SAU region is placed over the bootrom (which is expected to be the case in general, to get the correct NSC attribute on the Secure Gateway region), then this part of the ROM will become non-executable to Secure code, and consequently this part of the bootrom is not ROP-hardened. This part of the ROM contains the NSBOOT (including USB boot) implementation, as well as a RISC-V Armv6-M emulator which can be used to emulate most of the bootrom on RISC-V processors. This region is only implemented on the instruction-side IDAU query: this is an implementation detail which improves timing on the load/store IDAU query, and does not have security implications (given the mask ROM is inherently unwriteable) other than that the `tt` instruction will not be aware of this region.

The final 512 bytes of the bootrom has the Secure, Non-secure-Callable (NSC) attribute. This means it contains entry points for Non-secure calls into Secure code. Note that for this IDAU-defined attribute to take effect, the SAU-defined attribute for this range must also be NSC or lower. The recommended configuration is a single Non-secure SAU region covering the entirety of the bootrom. The bootrom exits into user code with the SAU enabled, and SAU region 7 active and covering the entirety of the bootrom.

XIP and SRAM are Non-secure in the IDAU, as they are expected to be divided using the SAU. Note that where the SAU

and IDAU differ, if the IDAU attribute is not Exempt, then the processor takes whichever is greater out of the SAU and IDAU attribute, in the order Secure > Non-secure-Callable > Non-secure.

Addresses not listed in this table are not decoded by the system AHB crossbar, and will return bus faults if accessed. In these ranges, the ROM's IDAU map is mirrored every 32 kB up to `0x0fffffff`, and the remaining addresses are Non-secure in the IDAU.

10.3. Overview (RISC-V)

The RP2350 bootrom does not implement secure boot for RISC-V processors. Secure flash boot can still be implemented on RISC-V, by storing secure boot code in OTP and disabling other boot media via the `BOOT_FLAGS0` row in OTP, but this is not supported natively by the RP2350 bootrom.

The RISC-V processors on RP2350 implement Machine and User execution modes, and the standard Physical Memory Protection unit (PMP), which can be used to enforce internal security or safety boundaries. See [Section 10.4](#).

Non-processor-specific hardware security features, such as debug disable OTP flags and the glitch detectors, functionally identically between Arm and RISC-V. Note however that the redundancy coprocessor (RCP) is not accessible from the RISC-V processors, as it uses a Cortex-M33-specific coprocessor interface.

10.4. Processor Security Features (RISC-V)

The Hazard3 processors on RP2350 implement the following standard RISC-V security features:

- The Machine and User execution modes (M-mode and U-mode)
- The Physical Memory Protection unit (PMP)

M-mode has full access to the processor's internal status registers, whereas U-mode does not. The processor's bus accesses are tagged with its current execution mode, and these are filtered by ACCESSCTRL bus filters, as described in [Section 10.5.2](#).

The processor starts in M-mode, and also enters M-mode upon taking any trap (exception or interrupt). It enters U-mode only by executing a return-from-M-mode instruction, `mret`, with previous privilege set to U-mode. This means all interrupts initially target M-mode, but can be deprivileged to U-mode via software routing. Note that stacks are software-managed on RISC-V, so some software cooperation is required to fully separate the two execution contexts, though there are enough hardware hooks to make this possible. See [Section 3.8.2](#) for more details of interrupts and exceptions on RISC-V, and how they relate to the core's privilege levels.

The PMP is a memory protection unit built into each RISC-V processor, which filters every instruction execution address and every load/store address against a list of permission regions. The Hazard3 instances on RP2350 are configured with 8 PMP regions each, with a 32-byte granule, and naturally aligned power-of-2 region support only.

Additionally, there are 3 PMP hardwired regions, which set a default User-mode RW permission on peripherals and a User-mode RWX permission on the ROM. These are assigned region numbers 8 through 10, which means they can be overridden by any dynamically configured regions, as lower-numbered regions always take precedence. These regions are included because the peripherals are expected to be assigned using ACCESSCTRL, rather than PMP, as there are many more peripherals than PMP regions – since this would mean setting a blanket U-mode RW permission on all peripherals in the common case, it made sense to implement this permission with hardwired regions, which are much cheaper than dynamically configured regions. The hardwired regions play a similar role to the Exempt regions in the RP2350 Cortex-M IDAU.

Together with the ACCESSCTRL filters, these PMP regions are an effective mechanism for partitioning addresses which are accessible to U-mode from those that are not. Hazard3 also includes one custom PMP feature, the `PMPCFGMO` register, which allows the PMP to set *M-mode* permissions as well as U-mode, without locking. This is useful for preventing accidental, rather than deliberate, access to a memory region.

10.4.1. Secure Boot Enable Procedure

To enable secure boot:

1. Program at least one public key fingerprint into OTP, starting at `BOOTKEY0_0`
2. Mark programmed keys as valid, by programming `BOOT_FLAGS1.KEY_VALID`
3. Mark unused keys as invalid, by programming `BOOT_FLAGS1.KEY_INVALID`
 - o `KEY_INVALID` takes precedence over `KEY_VALID`, so this prevents more keys being added later
 - o Program `KEY_INVALID` with additional bits to revoke keys at a later time
4. Disable debug, by programming `CRIT1.DEBUG_DISABLE` or `CRIT1.SECURE_DEBUG_DISABLE`, and/or installing a debug key ([Section 3.5.9.2](#))
5. Optionally enable the glitch detector ([Section 10.7](#)), by programming `CRIT1.GLITCH_DETECTOR_ENABLE` and setting the desired sensitivity in `CRIT1.GLITCH_DETECTOR_SENS`
6. Disable unused boot options such as USB and UART boot in `BOOT_FLAGS0`
7. Enable secure boot, by programming `CRIT1.SECURE_BOOT_ENABLE`

This procedure is irreversible. Make sure before programming that you are using the correct public key, correctly hashed. `picotool` supports programming keys into OTP from standard PEM files and performs the fingerprint hashing automatically. Programming the wrong key will make it impossible, or at least highly inconvenient, to run code on your device.

10.5. Access Control

The access control registers (ACCESSCTRL) define permissions required to access GPIOs, and bus endpoints such as peripherals and memory devices.

For each bus endpoint – for example, PIO0 – a bus access control register such as `PIO0` controls which AHB5 managers can access it, and at which bus security levels. This register has further implications, such as access to the `RESETS` controls for that block. [Section 10.5.2](#) goes into full detail on the bus access control registers.

For each GPIO, including the QSPI and USB DP/DM pins, a bit in the `GPIO_NSMASK0` and `GPIO_NSMASK1` register can be set to make that GPIO accessible to both the Secure and Non-secure domains, or clear to make it Secure-only. This has system-wide implications, such as whether a GPIO is visible to the Non-secure SIO, whether Non-secure code can access that GPIO's IO muxing and pad control registers, and whether the GPIO can select peripherals which are accessible only to Secure bus access. [Section 10.5.1](#) lists the effects of these registers exhaustively.

ACCESSCTRL registers are always fully readable by the processors in any security or privilege state, so that Non-secure software can enumerate the hardware it is allowed to access. However, writes to ACCESSCTRL are strictly controlled. Unprivileged writes, and writes from the DMA, return a bus fault. Writes from a Non-secure, Privileged (NSP) context are generally ignored, with the sole exception of the Non-secure, Unprivileged (NSU) bits in bus access control registers. These bits are Non-secure-writable if and only if the NSP bit is set. Writes can be further locked down using the `LOCK` register, which will cause writes from specific masters to be ignored.

To reduce the risk of accidental writes, all ACCESSCTRL registers with the exception of `GPIO_NSMASK0` and `GPIO_NSMASK1` require the 16-bit value `0xacce` to be present in the most-significant 16 bits of the write data. You can achieve this by ORing the value `0xacce0000` with the write data. This includes atomic SET/CLR/XOR alias writes. DMA writes are also forbidden, to avoid accidentally wiping permissions with a misconfigured DMA channel.

⚠️ IMPORTANT

Writes with the upper 16 bits not equal to `0xacce` will return a bus fault, in preference to silently leaving the permissions unchanged.

Finally, the `FORCE_CORE_NS` register makes core 1's bus accesses appear to be Non-secure at system level. This supports schemes where all Secure services are run on core 0, and therefore core 1 should not be able to access Secure hardware.

10.5.1. GPIO Access Control

The GPIO Non-secure access mask registers, `GPIO_NSmask0` and `GPIO_NSmask1`, contain one bit per GPIO. The layout of these two registers matches the layout of the SIO GPIO registers (Section 3.1.3), including the positions of the QSPI and USB DM/DP bits. Each GPIO is accessible to Non-secure software if and only if the relevant `GPIO_NSmask` bit is set. This prevents Non-secure software from interfering with or observing GPIOs used by Secure software.

All system-level GPIO controls, such as the IO and pad control registers, are shared by Secure and Non-secure code. However, access to these registers is filtered on a GPIO-by-GPIO basis according to the `GPIO_NSmask` register. This means that the same code can run unmodified in either a Secure or Non-secure context, and Secure software does not have to implement any interfaces for Non-secure GPIO access, provided that the appropriate GPIO security mask has been configured.

Setting a `GPIO_NSmask` bit has the following effects for the corresponding GPIO:

- The relevant SIO GPIO register bit (Section 3.1.3) becomes accessible through bus access to the Non-secure SIO. (Otherwise they are read-only zero.)
- The relevant SIO GPIO register bit becomes accessible to Non-secure code using GPIO coprocessor instructions (Section 3.6.1), assuming the GPIO coprocessor has been granted to Non-secure code in the Cortex-M33 NSACR register, and enabled in CPACR_NS. (Otherwise they are read-only zero.)
- The relevant IO control register (Section 9.11.1 or Section 9.11.2) becomes accessible to Non-secure code. (Otherwise it is read-only zero.)
- GPIO functions for Secure-only peripherals can not be selected on this GPIO
 - Attempting to select such a peripheral will select the null function (`0x1f`) instead
 - If a Secure-only peripheral is selected at the time that this GPIO is made Non-secure-accessible, then the selection will be changed to the null function.
- The relevant pad control register (Section 9.11.3 or Section 9.11.4) becomes accessible to Non-secure code. (Otherwise it is read-only zero.)
- Interrupts for this GPIO are routed to the Non-secure GPIO interrupts, `IO_IRQ_BANK0_NS` and `IO_IRQ_QSPI_NS`, rather than the default Secure interrupts, `IO_IRQ_BANK0` and `IO_IRQ_QSPI`. (See Section 3.2 for the system IRQ listing.)
- The relevant interrupt control and status bits, e.g. in `PROC0_INTS0`, become accessible to Non-secure code. (Otherwise they are read-only zero.)
- The GPIO can be read by PIO instances which are Non-secure-accessible. (Otherwise it will read as zero.)
 - Like the SIO, PIO can observe GPIOs even when they are not function-selected, so additional logic is added to mask Secure-only GPIOs from PIO instances which are accessible to Non-secure code

NOTE

Due to [RP2350-E3](#), on RP2350A (QFN-60), access to the `PADS_BANK0` registers is controlled by the wrong bits of `GPIO_NSmask`. On QFN-60 you must disable Non-secure access to the pads registers, and implement a software interface for Non-secure code to manipulate its assigned PADS registers.

10.5.2. Bus Access Control

The bus access control registers define which combinations of Secure/Non-secure and Privileged/Unprivileged are permitted to access each downstream bus port, as well as which upstream sources (processor 0/1, DMA or debugger) are permitted. This is the mechanism used to assign peripherals to security domains.

All accesses on the system bus ([Section 2.1](#)) are checked against the permission list for their destination. Accesses which do not fit the criteria specified in the relevant ACCESSCTRL register are filtered: the access does not reach its destination, and instead a bus error is returned directly from the bus fabric. There is no effect on the destination register, and no data is returned. Bus errors will generally result in an exception on the offending processor, or an error flag raised on the offending DMA channel.

There are 8 bits in each register (for example the `ADC` register). The `SP`, `SU`, `NSP` and `NSU` bits correspond to the processor security state from which a bus transfer originated, or the security level of the originating DMA channel:

- The `SP` bit must be set to allow access from:
 - Privileged software running in the Secure domain on an Arm processor
 - Machine-mode software on a RISC-V processor
 - A DMA channel with a security level of `SP` (3)
- The `SU` bit must be set, in addition to the `SP` bit, to allow access from:
 - User (unprivileged) software running in the Secure domain on an Arm processor
 - A DMA channel with a security level of `SU` (2)
- The `NSP` bit must be set to allow access from:
 - Privileged software running in the Non-secure domain on an Arm processor
 - Privileged Arm software running in the Secure domain on core 1, when `FORCE_CORE_NS.CORE1` is set
 - Machine-mode RISC-V software running on core 1, when `FORCE_CORE_NS.CORE1` is set
 - A DMA channel with a security level of `NSP` (1)
- The `NSU` bit must be set, in addition to the `NSP` bit, to allow access from:
 - User (unprivileged) software running in the Non-secure domain on an Arm processor
 - User (unprivileged) Arm software running on core 1 when `FORCE_CORE_NS.CORE1` is set
 - User-mode software on a RISC-V processor
 - A DMA channel with a security level of `NSU` (0)

NOTE

The security/privilege of AHB Mem-AP accesses are configurable, and have the same bus security/privilege level as load/stores from the corresponding security/privilege context on that processor. There is one AHB Mem-AP for each Arm processor.

NOTE

RISC-V Debug-mode memory accesses have the same bus security/privilege level as Machine-mode software running on that processor, and RISC-V System Bus Access through the Debug Module has the same bus security/privilege level as Machine-mode software running on core 1.

The **DBG**, **DMA**, **CORE1** and **CORE0** bits must be set in addition to the relevant security/privilege bits, in order to allow access from a particular source of bus accesses. The **DBG** bit corresponds to:

- Accesses from either Arm processor's AHB Mem-AP
- Accesses from either RISC-V core in Debug mode
- Accesses from RISC-V System Bus Access

Having debug access controlled separately from software-driven processor access means that, even with software locked out of a register block, the developer may still be able to access that block from the debugger.

Most bus access permission bits are Secure, Privileged-writable only. The sole exception is the NSU bit, which is also writable from a Non-secure, Privileged context if and only if the NSP bit in the same register is set. The intention is that once the Secure domain has granted Non-secure access, it is then up to Non-secure software to decide whether to grant Unprivileged access within the Non-secure domain.

10.5.2.1. Default Permissions

Most bus endpoints default to Secure access only, from any master, but there are exceptions. The following default to fully open access, i.e. any combination of security/privilege, from any master, for example because they are expected to be divided up by the processors' internal memory protection hardware:

- Boot ROM ([Section 4.1](#))
- XIP ([Section 4.4](#))
- SRAM ([Section 4.2](#))
- SYSINFO ([Section 12.15.1](#))

The following default to Secure, Privileged access (SP) only, from any master:

- XIP_AUX (DMA FIFOs) ([Section 4.4.3](#))
- SHA-256 ([Section 12.13](#))

The following default to Secure, Privileged access (SP) only, with DMA access forbidden by default:

- POWMAN ([Section 6.5](#)), which includes power management and voltage regulator control registers
- True random number generator ([Section 12.12](#))
- Clock control registers ([Section 8.1](#))
- XOSC ([Section 8.2](#))
- ROSC ([Section 8.3](#))
- SYSCFG ([Section 12.15.2](#))
- PLLs ([Section 8.6](#))

- Tick generators ([Section 8.5](#))
- Watchdog ([Section 12.9](#))
- PSM ([Section 7.4](#))
- XIP control registers ([Section 4.4.5](#))
- QMI ([Section 12.14](#))
- CoreSight trace DMA FIFO
- CoreSight self-hosted debug window

10.5.2.2. Other Effects of Bus Permissions

To avoid contradictory configurations such as a Secure-only peripheral being selected on a Non-secure-accessible GPIO, and to improve portability between Secure and Non-secure software, the bus access permission lists are also propagated to certain other system-level hardware:

- The reset controls for a given peripheral in the RESETS block ([Section 7.5](#)) are Non-secure-accessible if and only if the peripheral itself is Non-secure-accessible (note that Non-secure access to the RESETS block itself must also be granted)
- Non-secure-inaccessible peripherals can not be function-selected on Non-secure-accessible GPIOs, and attempting to do so will select the null GPIO function (`0x1f`)
- PIO blocks which are accessible to Non-secure, and those which are not, can not perform cross-PIO operations such as observing each other's interrupt flags
- PIO blocks which are accessible to Non-secure can not read Secure-only GPIOs
- DMA channels below the least-set effective permission bit (ignoring `SU` when `SP` is clear, and ignoring `NSU` when `NSP` is clear) are disconnected from that peripheral's DREQ signals

10.5.2.3. Blocks Without Bus Access Control

There are four memory-mapped blocks which do not have bus access control registers in ACCESSCTRL:

- The Cortex-M PPB, which is internal to the processors, and is banked internally over Secure/Non-secure
- The SIO, which is also banked over Secure/Non-secure access
- ACCESSCTRL itself, which is always world-readable and has its own internal filtering for writes
- Boot RAM, which is hardwired for Secure access only

10.5.3. List of Registers

The ACCESSCTRL registers start at a base address of `0x40060000` (defined as `ACCESSCTRL_BASE` in the SDK).

Table 911. List of ACCESSCTRL registers

Offset	Name	Info
0x00	LOCK	<p>Once a LOCK bit is written to 1, ACCESSCTRL silently ignores writes from that master. LOCK is writable only by a Secure, Privileged processor or debugger.</p> <p>LOCK bits are only writable when their value is zero. Once set, they can never be cleared, except by a full reset of ACCESSCTRL.</p> <p>Setting the LOCK bit does not affect whether an access raises a bus error. Unprivileged writes, or writes from the DMA, will continue to raise bus errors. All other accesses will continue not to.</p>
0x04	FORCE_CORE_NS	<p>Force core 1's bus accesses to always be Non-secure, no matter the core's internal state.</p> <p>Useful for schemes where one core is designated as the Non-secure core, since some peripherals may filter individual registers internally based on security state but not on master ID.</p>
0x08	CFGRESET	<p>Write 1 to reset all ACCESSCTRL configuration, except for the LOCK and FORCE_CORE_NS registers.</p> <p>This bit is used in the RP2350 bootrom to quickly restore ACCESSCTRL to a known state during the boot path.</p> <p>Note that, like all registers in ACCESSCTRL, this register is not writable when the writer's corresponding LOCK bit is set, therefore a master which has been locked out of ACCESSCTRL can not use the CFGRESET register to disturb its contents.</p>
0x0c	GPIO_NSMASK0	<p>Control whether GPIO0...31 are accessible to Non-secure code. Writable only by a Secure, Privileged processor or debugger.</p> <p>0 → Secure access only</p> <p>1 → Secure + Non-secure access</p>
0x10	GPIO_NSMASK1	Control whether GPIO32..47 are accessible to Non-secure code, and whether QSPI and USB bitbang are accessible through the Non-secure SIO. Writable only by a Secure, Privileged processor or debugger.
0x14	ROM	Control access to ROM. Defaults to fully open access.
0x18	XIP_MAIN	Control access to XIP_MAIN. Defaults to fully open access.
0x1c	SRAM0	Control access to SRAM0. Defaults to fully open access.
0x20	SRAM1	Control access to SRAM1. Defaults to fully open access.
0x24	SRAM2	Control access to SRAM2. Defaults to fully open access.
0x28	SRAM3	Control access to SRAM3. Defaults to fully open access.
0x2c	SRAM4	Control access to SRAM4. Defaults to fully open access.
0x30	SRAM5	Control access to SRAM5. Defaults to fully open access.
0x34	SRAM6	Control access to SRAM6. Defaults to fully open access.
0x38	SRAM7	Control access to SRAM7. Defaults to fully open access.

Offset	Name	Info
0x3c	SRAM8	Control access to SRAM8. Defaults to fully open access.
0x40	SRAM9	Control access to SRAM9. Defaults to fully open access.
0x44	DMA	Control access to DMA. Defaults to Secure access from any master.
0x48	USBCTRL	Control access to USBCTRL. Defaults to Secure access from any master.
0x4c	PIO0	Control access to PIO0. Defaults to Secure access from any master.
0x50	PIO1	Control access to PIO1. Defaults to Secure access from any master.
0x54	PIO2	Control access to PIO2. Defaults to Secure access from any master.
0x58	CORESIGHT_TRACE	Control access to CORESIGHT_TRACE. Defaults to Secure, Privileged processor or debug access only.
0x5c	CORESIGHT_PERIPH	Control access to CORESIGHT_PERIPH. Defaults to Secure, Privileged processor or debug access only.
0x60	SYSINFO	Control access to SYSINFO. Defaults to fully open access.
0x64	RESETS	Control access to RESETS. Defaults to Secure access from any master.
0x68	IO_BANK0	Control access to IO_BANK0. Defaults to Secure access from any master.
0x6c	IO_BANK1	Control access to IO_BANK1. Defaults to Secure access from any master.
0x70	PADS_BANK0	Control access to PADS_BANK0. Defaults to Secure access from any master.
0x74	PADS_QSPI	Control access to PADS_QSPI. Defaults to Secure access from any master.
0x78	BUSCTRL	Control access to BUSCTRL. Defaults to Secure access from any master.
0x7c	ADC	Control access to ADC. Defaults to Secure access from any master.
0x80	HSTX	Control access to HSTX. Defaults to Secure access from any master.
0x84	I2C0	Control access to I2C0. Defaults to Secure access from any master.
0x88	I2C1	Control access to I2C1. Defaults to Secure access from any master.
0x8c	PWM	Control access to PWM. Defaults to Secure access from any master.
0x90	SPI0	Control access to SPI0. Defaults to Secure access from any master.
0x94	SPI1	Control access to SPI1. Defaults to Secure access from any master.

Offset	Name	Info
0x98	TIMER0	Control access to TIMER0. Defaults to Secure access from any master.
0x9c	TIMER1	Control access to TIMER1. Defaults to Secure access from any master.
0xa0	UART0	Control access to UART0. Defaults to Secure access from any master.
0xa4	UART1	Control access to UART1. Defaults to Secure access from any master.
0xa8	OTP	Control access to OTP. Defaults to Secure access from any master.
0xac	TBMAN	Control access to TBMAN. Defaults to Secure access from any master.
0xb0	POWMAN	Control access to POWMAN. Defaults to Secure, Privileged processor or debug access only.
0xb4	TRNG	Control access to TRNG. Defaults to Secure, Privileged processor or debug access only.
0xb8	SHA256	Control access to SHA256. Defaults to Secure, Privileged access only.
0xbc	SYSCFG	Control access to SYSCFG. Defaults to Secure, Privileged processor or debug access only.
0xc0	CLOCKS	Control access to CLOCKS. Defaults to Secure, Privileged processor or debug access only.
0xc4	XOSC	Control access to XOSC. Defaults to Secure, Privileged processor or debug access only.
0xc8	ROSC	Control access to ROSC. Defaults to Secure, Privileged processor or debug access only.
0xcc	PLL_SYS	Control access to PLL_SYS. Defaults to Secure, Privileged processor or debug access only.
0xd0	PLL_USB	Control access to PLL_USB. Defaults to Secure, Privileged processor or debug access only.
0xd4	TICKS	Control access to TICKS. Defaults to Secure, Privileged processor or debug access only.
0xd8	WATCHDOG	Control access to WATCHDOG. Defaults to Secure, Privileged processor or debug access only.
0xdc	PSM	Control access to PSM. Defaults to Secure, Privileged processor or debug access only.
0xe0	XIP_CTRL	Control access to XIP_CTRL. Defaults to Secure, Privileged processor or debug access only.
0xe4	XIP_QMI	Control access to XIP_QMI. Defaults to Secure, Privileged processor or debug access only.
0xe8	XIP_AUX	Control access to XIP_AUX. Defaults to Secure, Privileged access only.

ACCESSCTRL: LOCK Register

Offset: 0x00

Description

Once a LOCK bit is written to 1, ACCESSCTRL silently ignores writes from that master. LOCK is writable only by a Secure, Privileged processor or debugger.

LOCK bits are only writable when their value is zero. Once set, they can never be cleared, except by a full reset of ACCESSCTRL.

Setting the LOCK bit does not affect whether an access raises a bus error. Unprivileged writes, or writes from the DMA, will continue to raise bus errors. All other accesses will continue not to.

Table 912. LOCK Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	DEBUG		RW	0x0
2	DMA		RO	0x1
1	CORE1		RW	0x0
0	CORE0		RW	0x0

ACCESSCTRL: FORCE_CORE_NS Register

Offset: 0x04

Description

Force core 1's bus accesses to always be Non-secure, no matter the core's internal state.

Useful for schemes where one core is designated as the Non-secure core, since some peripherals may filter individual registers internally based on security state but not on master ID.

Table 913. FORCE_CORE_NS Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	CORE1		RW	0x0
0	Reserved.	-	-	-

ACCESSCTRL: CFGRESET Register

Offset: 0x08

Table 914. CFGRESET Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	<p>Write 1 to reset all ACCESSCTRL configuration, except for the LOCK and FORCE_CORE_NS registers.</p> <p>This bit is used in the RP2350 bootrom to quickly restore ACCESSCTRL to a known state during the boot path.</p> <p>Note that, like all registers in ACCESSCTRL, this register is not writable when the writer's corresponding LOCK bit is set, therefore a master which has been locked out of ACCESSCTRL can not use the CFGRESET register to disturb its contents.</p>	SC	0x0

ACCESSCTRL: GPIO_NSMASK0 Register

Offset: 0x0c

Table 915. GPIO_NSMASK0 Register

Bits	Description	Type	Reset
31:0	<p>Control whether GPIO0...31 are accessible to Non-secure code. Writable only by a Secure, Privileged processor or debugger.</p> <p>0 → Secure access only</p> <p>1 → Secure + Non-secure access</p>	RW	0x00000000

ACCESSCTRL: GPIO_NSMASK1 Register

Offset: 0x10

Description

Control whether GPIO32..47 are accessible to Non-secure code, and whether QSPI and USB bitbang are accessible through the Non-secure SIO. Writable only by a Secure, Privileged processor or debugger.

Table 916. GPIO_NSMASK1 Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		RW	0x0
27	QSPI_CSN		RW	0x0
26	QSPI_SCK		RW	0x0
25	USB_DM		RW	0x0
24	USB_DP		RW	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		RW	0x0000

ACCESSCTRL: ROM Register

Offset: 0x14

Description

Control whether debugger, DMA, core 0 and core 1 can access ROM, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which

becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 917. ROM Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, ROM can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, ROM can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, ROM can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, ROM can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, ROM can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, ROM can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, ROM can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, ROM can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: XIP_MAIN Register

Offset: 0x18

Description

Control whether debugger, DMA, core 0 and core 1 can access XIP_MAIN, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 918. XIP_MAIN Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, XIP_MAIN can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, XIP_MAIN can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, XIP_MAIN can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
4	CORE0	If 1, XIP_MAIN can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, XIP_MAIN can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, XIP_MAIN can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, XIP_MAIN can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, XIP_MAIN can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM0 Register

Offset: 0x1c

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM0, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 919. SRAM0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM0 can be accessed from a Non-secure, Privileged context.	RW	0x1

Bits	Name	Description	Type	Reset
0	NSU	If 1, and NSP is also set, SRAM0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM1 Register

Offset: 0x20

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM1, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 920. SRAM1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM1 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM1 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM1 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM1 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM1 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM1 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM1 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM1 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM2 Register

Offset: 0x24

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM2, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 921. SRAM2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM2 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM2 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM2 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM2 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM2 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM2 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM2 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM2 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM3 Register

Offset: 0x28

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM3, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 922. SRAM3 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM3 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM3 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
5	CORE1	If 1, SRAM3 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM3 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM3 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM3 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM3 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM3 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM4 Register

Offset: 0x2c

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM4, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 923. SRAM4 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM4 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM4 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM4 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM4 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM4 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM4 can be accessed from a Secure, Unprivileged context.	RW	0x1

Bits	Name	Description	Type	Reset
1	NSP	If 1, SRAM4 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM4 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM5 Register

Offset: 0x30

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM5, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 924. SRAM5 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM5 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM5 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM5 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM5 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM5 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM5 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM5 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM5 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM6 Register

Offset: 0x34

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM6, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 925. SRAM6 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM6 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM6 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM6 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM6 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM6 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM6 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM6 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM6 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM7 Register

Offset: 0x38

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM7, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 926. SRAM7 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM7 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
6	DMA	If 1, SRAM7 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM7 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM7 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM7 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM7 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM7 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM7 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM8 Register

Offset: 0x3c

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM8, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 927. SRAM8 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM8 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM8 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM8 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM8 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
3	SP	If 1, SRAM8 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM8 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM8 can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SRAM8 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: SRAM9 Register

Offset: 0x40

Description

Control whether debugger, DMA, core 0 and core 1 can access SRAM9, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 928. SRAM9 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SRAM9 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SRAM9 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SRAM9 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SRAM9 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SRAM9 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SRAM9 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SRAM9 can be accessed from a Non-secure, Privileged context.	RW	0x1

Bits	Name	Description	Type	Reset
0	NSU	If 1, and NSP is also set, SRAM9 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: DMA Register

Offset: 0x44

Description

Control whether debugger, DMA, core 0 and core 1 can access DMA, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 929. DMA Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, DMA can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, DMA can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, DMA can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, DMA can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, DMA can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, DMA can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, DMA can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, DMA can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: USBCTRL Register

Offset: 0x48

Description

Control whether debugger, DMA, core 0 and core 1 can access USBCTRL, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 930. USBCTRL Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, USBCTRL can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, USBCTRL can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, USBCTRL can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, USBCTRL can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, USBCTRL can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, USBCTRL can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, USBCTRL can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, USBCTRL can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PIO0 Register

Offset: 0x4c

Description

Control whether debugger, DMA, core 0 and core 1 can access PIO0, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 931. PIO0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PIO0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PIO0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
5	CORE1	If 1, PIO0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PIO0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PIO0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PIO0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, PIO0 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, PIO0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PIO1 Register

Offset: 0x50

Description

Control whether debugger, DMA, core 0 and core 1 can access PIO1, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 932. PIO1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PIO1 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PIO1 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, PIO1 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PIO1 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PIO1 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PIO1 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, PIO1 can be accessed from a Non-secure, Privileged context.	RW	0x0

Bits	Name	Description	Type	Reset
0	NSU	If 1, and NSP is also set, PIO1 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PIO2 Register

Offset: 0x54

Description

Control whether debugger, DMA, core 0 and core 1 can access PIO2, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 933. PIO2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PIO2 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PIO2 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, PIO2 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PIO2 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PIO2 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PIO2 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, PIO2 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, PIO2 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: CORESIGHT_TRACE Register

Offset: 0x58

Description

Control whether debugger, DMA, core 0 and core 1 can access CORESIGHT_TRACE, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 934.
CORESIGHT_TRACE
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, CORESIGHT_TRACE can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, CORESIGHT_TRACE can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, CORESIGHT_TRACE can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, CORESIGHT_TRACE can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, CORESIGHT_TRACE can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, CORESIGHT_TRACE can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, CORESIGHT_TRACE can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, CORESIGHT_TRACE can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: CORESIGHT_PERIPH Register

Offset: 0x5c

Description

Control whether debugger, DMA, core 0 and core 1 can access CORESIGHT_PERIPH, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 935.
CORESIGHT_PERIPH
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, CORESIGHT_PERIPH can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, CORESIGHT_PERIPH can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0

Bits	Name	Description	Type	Reset
5	CORE1	If 1, CORESIGHT_PERIPH can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, CORESIGHT_PERIPH can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, CORESIGHT_PERIPH can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, CORESIGHT_PERIPH can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, CORESIGHT_PERIPH can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, CORESIGHT_PERIPH can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: SYSINFO Register

Offset: 0x60

Description

Control whether debugger, DMA, core 0 and core 1 can access SYSINFO, and at what security/privilege levels they can do so.

Defaults to fully open access.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 936. SYSINFO Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SYSINFO can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SYSINFO can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SYSINFO can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SYSINFO can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SYSINFO can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SYSINFO can be accessed from a Secure, Unprivileged context.	RW	0x1

Bits	Name	Description	Type	Reset
1	NSP	If 1, SYSINFO can be accessed from a Non-secure, Privileged context.	RW	0x1
0	NSU	If 1, and NSP is also set, SYSINFO can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x1

ACCESSCTRL: RESETS Register

Offset: 0x64

Description

Control whether debugger, DMA, core 0 and core 1 can access RESETS, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 937. RESETS Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, RESETS can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, RESETS can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, RESETS can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, RESETS can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, RESETS can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, RESETS can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, RESETS can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, RESETS can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: IO_BANK0 Register

Offset: 0x68

Description

Control whether debugger, DMA, core 0 and core 1 can access IO_BANK0, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 938. IO_BANK0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, IO_BANK0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, IO_BANK0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, IO_BANK0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, IO_BANK0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, IO_BANK0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, IO_BANK0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, IO_BANK0 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, IO_BANK0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: IO_BANK1 Register

Offset: 0x6c

Description

Control whether debugger, DMA, core 0 and core 1 can access IO_BANK1, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 939. IO_BANK1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, IO_BANK1 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
6	DMA	If 1, IO_BANK1 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, IO_BANK1 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, IO_BANK1 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, IO_BANK1 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, IO_BANK1 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, IO_BANK1 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, IO_BANK1 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PADS_BANK0 Register

Offset: 0x70

Description

Control whether debugger, DMA, core 0 and core 1 can access PADS_BANK0, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 940.
PADS_BANK0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PADS_BANK0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PADS_BANK0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, PADS_BANK0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PADS_BANK0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
3	SP	If 1, PADS_BANK0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PADS_BANK0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, PADS_BANK0 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, PADS_BANK0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PADS_QSPI Register

Offset: 0x74

Description

Control whether debugger, DMA, core 0 and core 1 can access PADS_QSPI, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 941. PADS_QSPI Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PADS_QSPI can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PADS_QSPI can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, PADS_QSPI can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PADS_QSPI can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PADS_QSPI can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PADS_QSPI can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, PADS_QSPI can be accessed from a Non-secure, Privileged context.	RW	0x0

Bits	Name	Description	Type	Reset
0	NSU	If 1, and NSP is also set, PADS_QSPI can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: BUSCTRL Register

Offset: 0x78

Description

Control whether debugger, DMA, core 0 and core 1 can access BUSCTRL, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 942. BUSCTRL Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, BUSCTRL can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, BUSCTRL can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, BUSCTRL can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, BUSCTRL can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, BUSCTRL can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, BUSCTRL can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, BUSCTRL can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, BUSCTRL can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: ADC Register

Offset: 0x7c

Description

Control whether debugger, DMA, core 0 and core 1 can access ADC, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 943. ADC Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, ADC can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, ADC can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, ADC can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, ADC can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, ADC can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, ADC can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, ADC can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, ADC can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: HSTX Register

Offset: 0x80

Description

Control whether debugger, DMA, core 0 and core 1 can access HSTX, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 944. HSTX Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, HSTX can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, HSTX can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
5	CORE1	If 1, HSTX can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, HSTX can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, HSTX can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, HSTX can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, HSTX can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, HSTX can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: I2C0 Register

Offset: 0x84

Description

Control whether debugger, DMA, core 0 and core 1 can access I2C0, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 945. I2C0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, I2C0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, I2C0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, I2C0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, I2C0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, I2C0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, I2C0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, I2C0 can be accessed from a Non-secure, Privileged context.	RW	0x0

Bits	Name	Description	Type	Reset
0	NSU	If 1, and NSP is also set, I2C0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: I2C1 Register

Offset: 0x88

Description

Control whether debugger, DMA, core 0 and core 1 can access I2C1, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 946. I2C1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, I2C1 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, I2C1 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, I2C1 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, I2C1 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, I2C1 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, I2C1 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, I2C1 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, I2C1 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PWM Register

Offset: 0x8c

Description

Control whether debugger, DMA, core 0 and core 1 can access PWM, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 947. PWM Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PWM can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PWM can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, PWM can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PWM can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PWM can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PWM can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, PWM can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, PWM can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: SPI0 Register

Offset: 0x90

Description

Control whether debugger, DMA, core 0 and core 1 can access SPI0, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 948. SPI0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SPI0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SPI0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SPI0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
4	CORE0	If 1, SPI0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SPI0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SPI0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SPI0 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, SPI0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: SPI1 Register

Offset: 0x94

Description

Control whether debugger, DMA, core 0 and core 1 can access SPI1, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 949. SPI1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SPI1 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SPI1 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SPI1 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SPI1 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SPI1 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SPI1 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, SPI1 can be accessed from a Non-secure, Privileged context.	RW	0x0

Bits	Name	Description	Type	Reset
0	NSU	If 1, and NSP is also set, SPI1 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: TIMER0 Register

Offset: 0x98

Description

Control whether debugger, DMA, core 0 and core 1 can access TIMER0, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 950. TIMER0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, TIMER0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, TIMER0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, TIMER0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, TIMER0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, TIMER0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, TIMER0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, TIMER0 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, TIMER0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: TIMER1 Register

Offset: 0x9c

Description

Control whether debugger, DMA, core 0 and core 1 can access TIMER1, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 951. TIMER1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, TIMER1 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, TIMER1 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, TIMER1 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, TIMER1 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, TIMER1 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, TIMER1 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, TIMER1 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, TIMER1 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: UART0 Register

Offset: 0xa0

Description

Control whether debugger, DMA, core 0 and core 1 can access UART0, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 952. UART0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, UART0 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, UART0 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
5	CORE1	If 1, UART0 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, UART0 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, UART0 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, UART0 can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, UART0 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, UART0 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: UART1 Register

Offset: 0xa4

Description

Control whether debugger, DMA, core 0 and core 1 can access UART1, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 953. UART1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, UART1 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, UART1 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, UART1 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, UART1 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, UART1 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, UART1 can be accessed from a Secure, Unprivileged context.	RW	0x1

Bits	Name	Description	Type	Reset
1	NSP	If 1, UART1 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, UART1 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: OTP Register

Offset: 0xa8

Description

Control whether debugger, DMA, core 0 and core 1 can access OTP, and at what security/privilege levels they can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 954. OTP Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, OTP can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, OTP can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, OTP can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, OTP can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, OTP can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, OTP can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, OTP can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, OTP can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: TBMAN Register

Offset: 0xac

Description

Control whether debugger, DMA, core 0 and core 1 can access TBMAN, and at what security/privilege levels they

can do so.

Defaults to Secure access from any master.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 955. TBMAN Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, TBMAN can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, TBMAN can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, TBMAN can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, TBMAN can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, TBMAN can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, TBMAN can be accessed from a Secure, Unprivileged context.	RW	0x1
1	NSP	If 1, TBMAN can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, TBMAN can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: POWMAN Register

Offset: 0xb0

Description

Control whether debugger, DMA, core 0 and core 1 can access POWMAN, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 956. POWMAN Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, POWMAN can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
6	DMA	If 1, POWMAN can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, POWMAN can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, POWMAN can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, POWMAN can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, POWMAN can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, POWMAN can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, POWMAN can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: TRNG Register

Offset: 0xb4

Description

Control whether debugger, DMA, core 0 and core 1 can access TRNG, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 957. TRNG Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, TRNG can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, TRNG can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, TRNG can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, TRNG can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, TRNG can be accessed from a Secure, Privileged context.	RW	0x1

Bits	Name	Description	Type	Reset
2	SU	If 1, and SP is also set, TRNG can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, TRNG can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, TRNG can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: SHA256 Register

Offset: 0xb8

Description

Control whether debugger, DMA, core 0 and core 1 can access SHA256, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 958. SHA256 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SHA256 can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SHA256 can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, SHA256 can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SHA256 can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SHA256 can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SHA256 can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, SHA256 can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, SHA256 can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: SYSCFG Register

Offset: 0xbc**Description**

Control whether debugger, DMA, core 0 and core 1 can access SYSCFG, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 959. SYSCFG Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, SYSCFG can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, SYSCFG can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, SYSCFG can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, SYSCFG can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, SYSCFG can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, SYSCFG can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, SYSCFG can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, SYSCFG can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: CLOCKS Register**Offset:** 0xc0**Description**

Control whether debugger, DMA, core 0 and core 1 can access CLOCKS, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 960. CLOCKS Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7	DBG	If 1, CLOCKS can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, CLOCKS can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, CLOCKS can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, CLOCKS can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, CLOCKS can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, CLOCKS can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, CLOCKS can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, CLOCKS can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: XOSC Register

Offset: 0xc4

Description

Control whether debugger, DMA, core 0 and core 1 can access XOSC, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 961. XOSC Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, XOSC can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, XOSC can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, XOSC can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, XOSC can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
3	SP	If 1, XOSC can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, XOSC can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, XOSC can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, XOSC can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: ROSC Register

Offset: 0xc8

Description

Control whether debugger, DMA, core 0 and core 1 can access ROSC, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 962. ROSC Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, ROSC can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, ROSC can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, ROSC can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, ROSC can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, ROSC can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, ROSC can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, ROSC can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, ROSC can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PLL_SYS Register

Offset: 0xcc

Description

Control whether debugger, DMA, core 0 and core 1 can access PLL_SYS, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 963. PLL_SYS Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PLL_SYS can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PLL_SYS can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, PLL_SYS can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PLL_SYS can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PLL_SYS can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PLL_SYS can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, PLL_SYS can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, PLL_SYS can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PLL_USB Register

Offset: 0xd0

Description

Control whether debugger, DMA, core 0 and core 1 can access PLL_USB, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 964. PLL_USB Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7	DBG	If 1, PLL_USB can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PLL_USB can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, PLL_USB can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PLL_USB can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PLL_USB can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PLL_USB can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, PLL_USB can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, PLL_USB can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: TICKS Register

Offset: 0xd4

Description

Control whether debugger, DMA, core 0 and core 1 can access TICKS, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 965. TICKS Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, TICKS can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, TICKS can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, TICKS can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, TICKS can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1

Bits	Name	Description	Type	Reset
3	SP	If 1, TICKS can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, TICKS can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, TICKS can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, TICKS can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: WATCHDOG Register

Offset: 0xd8

Description

Control whether debugger, DMA, core 0 and core 1 can access WATCHDOG, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 966.
WATCHDOG Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, WATCHDOG can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, WATCHDOG can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, WATCHDOG can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, WATCHDOG can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, WATCHDOG can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, WATCHDOG can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, WATCHDOG can be accessed from a Non-secure, Privileged context.	RW	0x0

Bits	Name	Description	Type	Reset
0	NSU	If 1, and NSP is also set, WATCHDOG can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: PSM Register

Offset: 0xdc

Description

Control whether debugger, DMA, core 0 and core 1 can access PSM, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 967. PSM Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, PSM can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, PSM can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, PSM can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, PSM can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, PSM can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, PSM can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, PSM can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, PSM can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: XIP_CTRL Register

Offset: 0xe0

Description

Control whether debugger, DMA, core 0 and core 1 can access XIP_CTRL, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 968. XIP_CTRL Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, XIP_CTRL can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, XIP_CTRL can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0
5	CORE1	If 1, XIP_CTRL can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, XIP_CTRL can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, XIP_CTRL can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, XIP_CTRL can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, XIP_CTRL can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, XIP_CTRL can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: XIP_QMI Register

Offset: 0xe4

Description

Control whether debugger, DMA, core 0 and core 1 can access XIP_QMI, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged processor or debug access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 969. XIP_QMI Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, XIP_QMI can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, XIP_QMI can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x0

Bits	Name	Description	Type	Reset
5	CORE1	If 1, XIP_QMI can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, XIP_QMI can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, XIP_QMI can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, XIP_QMI can be accessed from a Secure, Unprivileged context.	RW	0x0
1	NSP	If 1, XIP_QMI can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, XIP_QMI can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

ACCESSCTRL: XIP_AUX Register

Offset: 0xe8

Description

Control whether debugger, DMA, core 0 and core 1 can access XIP_AUX, and at what security/privilege levels they can do so.

Defaults to Secure, Privileged access only.

This register is writable only from a Secure, Privileged processor or debugger, with the exception of the NSU bit, which becomes Non-secure-Privileged-writable when the NSP bit is set.

Table 970. XIP_AUX Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	DBG	If 1, XIP_AUX can be accessed by the debugger, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
6	DMA	If 1, XIP_AUX can be accessed by the DMA, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
5	CORE1	If 1, XIP_AUX can be accessed by core 1, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
4	CORE0	If 1, XIP_AUX can be accessed by core 0, at security/privilege levels permitted by SP/NSP/SU/NSU in this register.	RW	0x1
3	SP	If 1, XIP_AUX can be accessed from a Secure, Privileged context.	RW	0x1
2	SU	If 1, and SP is also set, XIP_AUX can be accessed from a Secure, Unprivileged context.	RW	0x0

Bits	Name	Description	Type	Reset
1	NSP	If 1, XIP_AUX can be accessed from a Non-secure, Privileged context.	RW	0x0
0	NSU	If 1, and NSP is also set, XIP_AUX can be accessed from a Non-secure, Unprivileged context. This bit is writable from a Non-secure, Privileged context, if and only if the NSP bit is set.	RW	0x0

10.6. DMA

The RP2350 system DMA is a peripheral which performs arbitrary reads and writes on memory. This means that, as with a processor, care is necessary to maintain isolation between memory or peripherals owned by different security domains. As a rule, any given processor context must not access memory or peripherals belonging to a more secure context, and the DMA helps maintain this invariant by ensuring software can not use the DMA to access a more secure context on its behalf, including such cases as a processor programming the DMA to program the DMA.

RP2350 extends the processor security/privilege states to individual DMA channels, and the DMA filters its own memory accesses with a built-in memory protection unit (MPU) similarly capable to the Armv8-M SAU or the RISC-V PMP. When correctly configured, this allows multiple security domains to transparently and safely share DMA resources. It is also possible to assign the entire DMA block wholesale to a single security domain using the ACCESSCTRL registers ([Section 10.5](#)) if this fine-grained configuration is not desired.

This section gives an overview of the DMA's security features. The specific hardware details are documented in [Section 12.6.6](#).

10.6.1. Channel Security Attributes

Each channel is assigned a security level using the per-channel registers starting at [SECCFG_CH0](#). This assignment defines:

- The minimum privilege required to configure and control the channel, or observe its status
- The bus privilege at which the channel performs its memory accesses

For the sake of comparing security levels, the DMA assigns the following total order to AHB5 security/privilege attributes: Secure + Privileged > Secure + Unprivileged > Non-secure + Privileged > Non-secure + Unprivileged.

A channel's security level can be changed freely up until any of the channel's control registers is written. After this point, its security level is locked, and can't be changed until the DMA block is reset. At reset, all channels are Secure + Privileged (security level = 3, the maximum).

The effects of the channel SECCFG registers are listed exhaustively in the relevant DMA documentation, [Section 12.6.6.1](#).

10.6.2. Memory Protection Unit

The RP2350 DMA features a memory protection unit which is configured with the security/privilege level required to access up to 8 different address ranges, plus a default level for addresses not matched by any of those 8 ranges. The addresses of all DMA reads/writes are checked against the MPU address map, and if the originating channel's security level is lower than that defined in the address map, the access is filtered. A filtered access has no effect on the downstream bus, and returns a bus error to the offending channel.

The DMA memory protection unit is configured by DMA control registers starting from [MPU_CTRL](#). See [Section 12.6.6.3](#)

for more details.

10.6.3. DREQ Attributes

Channels are not permitted to interface with the DREQs of peripherals which are above their security level, as determined by the peripheral access controls in ACCESSCTRL. This is done to avoid any information being inferred from the timing of secure peripheral transfers, and because the clear handshake on the RP2040 DREQ can be used maliciously to cause a Secure DMA channel to overflow its destination FIFO and corrupt/lose data. (See [Section 12.6.4.2](#) for details of the DREQ handshake.)

The DREQ security levels are driven by the ACCESSCTRL block access registers. Specifically, ACCESSCTRL takes the index of the least-significant set bit in the 4-bit permission mask (having first ANDed the SP into SU, and NSP into SU) to create a 2-bit integer which is compared with the DMA channel's security level to determine whether it can interface with this DREQ.

10.6.4. IRQ Attributes

Each of the four shared DMA interrupt lines (IRQs) has a configurable security level. The IRQ's security level is compared with channel security levels, and with the bus privilege of accesses to the DMA's interrupt control registers, to determine:

- Whether a bus access is permitted to read/write the INTE/INTF/INTS registers for this IRQ
- Whether a given channel will be visible in this IRQ's INTS register (and therefore whether that channel will cause assertion of this interrupt)
- Whether a given channel can have its interrupt pending flag set/cleared via this IRQ's INTF/INTS registers

For a bus access to view/configure an interrupt, it must have a security level greater than or equal to the interrupt's security level. For an IRQ to observe a channel's interrupt pending flag, it must have a security level greater than or equal to the channel's security level.

There is only a single INTR register. Which channels' interrupts can be observed and cleared through INTR is determined by comparing channel security levels to the security level of the INTR bus access.

10.7. Glitch Detector

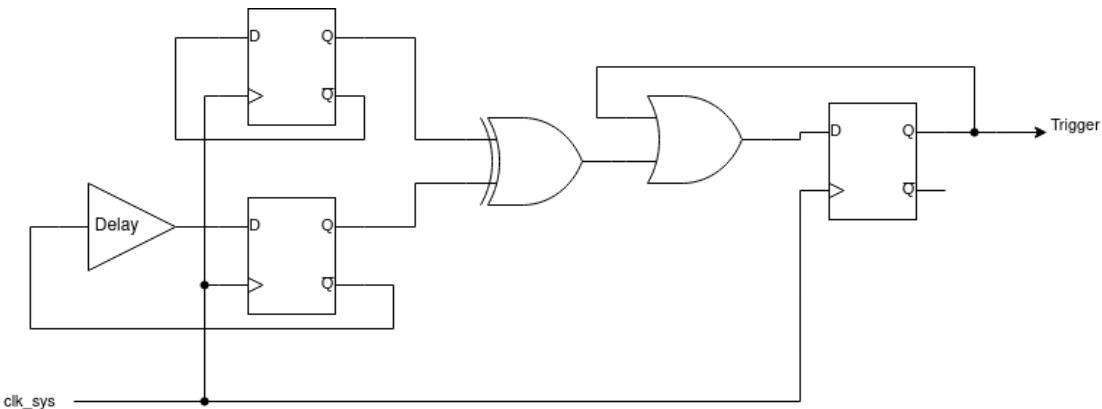
The glitch detector detects loss of setup and hold margin in the system clock domain, which may be caused by deliberate external manipulation of the system clock or core supply voltage. When it detects loss, the glitch detector triggers a system reset rather than allowing software to continue to execute in a possibly undefined state. It responds within one system clock cycle, unlike the brownout detector, which has much more limited analog bandwidth.

The glitch detector is disabled by default, and can be armed by setting the `GLITCH_DETECTOR_ENABLE` flag in OTP. For debugging purposes, you can also enable the glitch detector via the `ARM` register. This is not recommended in security-sensitive applications, as the system is vulnerable until the point that software can enable the detectors.

10.7.1. Theory of Operation

The glitch detector comprises of four identical detector circuits, based on a pair of D flip-flops. These detector circuits are each placed in different, physically distant locations within the core voltage domain.

Figure 40. Glitch detector trigger circuit. Two flops each toggle on every system clock cycle. One has a programmable delay line in its feedback path, the other does not. Loss of setup or hold margin causes one of the flops to fail to toggle, so the flops values differ, setting the trigger output.



The detector triggers when the two D-flops take on different values, which is impossible under normal circumstances. The delay line is programmable from 75% to 120% of the minimum system clock period in increments of 15%. Higher delays make the circuit more sensitive to loss of setup margin. To configure initial sensitivity, use the [GLITCH_DETECTOR_SENS](#) OTP flags. You can fine-tune sensitivity for each detector using the [SENSITIVITY](#) register.

Because the circuit is constructed from digital standard cells, it closely tracks the changes in propagation delay to nearby cells caused by voltage and temperature fluctuations. Therefore the delay line's propagation delay is specified as a fraction of the maximum system clock data path delay, rather than a fixed time in nanoseconds.

10.7.2. Trigger Response

When any of the detectors fires, the corresponding bit in the [TRIG_STATUS](#) is set. If the glitch detector block is armed, this detector event also resets almost all logic in the switched core domain. The glitch detector is armed if:

- The [DISARM](#) register is not set to the disarming bit pattern, and at least one of the following is true:
 - The [GLITCH_DETECTOR_EN](#) OTP flag was programmed some time before the most recent reset of the OTP block
 - The [ARM](#) register is set to an arming bit pattern

This holds the majority of the switched core domain in reset for approximately 120 microseconds before releasing the reset. Specifically, this resets the PSM (Section 7.3), which resets all PSM-controlled resets starting with the processor cold reset domain, in addition to all blocks reset by the [RESETS](#) block, which is itself reset by the PSM. The detector circuits are also reset, as is the system watchdog including the watchdog scratch registers.

After a glitch detector-initiated reset, the [CHIP_RESET.HAD_GLITCH_DETECT](#) flag is set so that software can diagnose that the last reset was caused by a glitch detector trigger. Check the [TRIG_STATUS](#) register to see which detector fired. This can be useful for tuning the thresholds of individual detectors.

The only way to clear the detector circuits is to reset them, either via a full switched core domain reset (such as the [RUN](#) pin, the SW-DP reset request, a PoR/BoR reset, or a reset of the switched core domain configured by POWMAN controls), or by arming the glitch detector block so that the detectors reset along with the PSM.

Recovering from the glitch detector firing requires the low-power oscillator to be running (Section 8.4). Allowing the glitch detectors to fire when the LPOSC is disabled results in the chip holding itself in reset indefinitely until an external reset such as the [RUN](#) pin resets the detectors.

10.7.3. List of Registers

The glitch detector control registers start at an address of [0x40158000](#).

Table 971. List of [GLITCH_DETECTOR](#) registers

Offset	Name	Info
0x00	ARM	<p>Forcibly arm the glitch detectors, if they are not already armed by OTP. When armed, any individual detector trigger will cause a restart of the switched core power domain's power-on reset state machine.</p> <p>Glitch detector triggers are recorded accumulatively in TRIG_STATUS. If the system is reset by a glitch detector trigger, this is recorded in POWMAN_CHIP_RESET.</p> <p>This register is Secure read/write only.</p>
0x04	DISARM	
0x08	SENSITIVITY	<p>Adjust the sensitivity of glitch detectors to values other than their OTP-provided defaults.</p> <p>This register is Secure read/write only.</p>
0x0c	LOCK	
0x10	TRIG_STATUS	<p>Set when a detector output triggers. Write-1-clear.</p> <p>(May immediately return high if the detector remains in a failed state. Detectors can only be cleared by a full reset of the switched core power domain.)</p> <p>This register is Secure read/write only.</p>
0x14	TRIG_FORCE	<p>Simulate the firing of one or more detectors. Writing ones to this register will set the matching bits in STATUS_TRIG.</p> <p>If the glitch detectors are currently armed, writing ones will also immediately reset the switched core power domain, and set the reset reason latches in POWMAN_CHIP_RESET to indicate a glitch detector resets.</p> <p>This register is Secure read/write only.</p>

GLITCH_DETECTOR: ARM Register

Offset: 0x00

Table 972. ARM Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Forcibly arm the glitch detectors, if they are not already armed by OTP. When armed, any individual detector trigger will cause a restart of the switched core power domain's power-on reset state machine.</p> <p>Glitch detector triggers are recorded accumulatively in TRIG_STATUS. If the system is reset by a glitch detector trigger, this is recorded in POWMAN_CHIP_RESET.</p> <p>This register is Secure read/write only.</p> <p>0x5bad → Do not force the glitch detectors to be armed</p> <p>0x0000 → Force the glitch detectors to be armed. (Any value other than ARM_NO counts as YES)</p>	RW	0x5bad

GLITCH_DETECTOR: DISARM Register

Offset: 0x04

Table 973. DISARM Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Forcibly disarm the glitch detectors, if they are armed by OTP. Ignored if ARM is YES.</p> <p>This register is Secure read/write only.</p> <p>0x0000 → Do not disarm the glitch detectors. (Any value other than DISARM_YES counts as NO)</p> <p>0xdcaf → Disarm the glitch detectors</p>	RW	0x0000

GLITCH_DETECTOR: SENSITIVITY Register

Offset: 0x08

Description

Adjust the sensitivity of glitch detectors to values other than their OTP-provided defaults.

This register is Secure read/write only.

Table 974. SENSITIVITY Register

Bits	Name	Description	Type	Reset
31:24	DEFAULT	<p>0x00 → Use the default sensitivity configured in OTP for all detectors. (Any value other than DEFAULT_NO counts as YES)</p> <p>0xde → Do not use the default sensitivity configured in OTP. Instead use the value from this register.</p>	RW	0x00
23:16	Reserved.	-	-	-
15:14	DET3_INV	Must be the inverse of DET3, else the default value is used.	RW	0x0
13:12	DET2_INV	Must be the inverse of DET2, else the default value is used.	RW	0x0
11:10	DET1_INV	Must be the inverse of DET1, else the default value is used.	RW	0x0

Bits	Name	Description	Type	Reset
9:8	DET0_INV	Must be the inverse of DET0, else the default value is used.	RW	0x0
7:6	DET3	Set sensitivity for detector 3. Higher values are more sensitive.	RW	0x0
5:4	DET2	Set sensitivity for detector 2. Higher values are more sensitive.	RW	0x0
3:2	DET1	Set sensitivity for detector 1. Higher values are more sensitive.	RW	0x0
1:0	DET0	Set sensitivity for detector 0. Higher values are more sensitive.	RW	0x0

GLITCH_DETECTOR: LOCK Register

Offset: 0x0c

Table 975. LOCK Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	Write any nonzero value to disable writes to ARM, DISARM, SENSITIVITY and LOCK. This register is Secure read/write only.	RW	0x00

GLITCH_DETECTOR: TRIG_STATUS Register

Offset: 0x10

Description

Set when a detector output triggers. Write-1-clear.

(May immediately return high if the detector remains in a failed state. Detectors can only be cleared by a full reset of the switched core power domain.)

This register is Secure read/write only.

Table 976. TRIG_STATUS Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	DET3		WC	0x0
2	DET2		WC	0x0
1	DET1		WC	0x0
0	DET0		WC	0x0

GLITCH_DETECTOR: TRIG_FORCE Register

Offset: 0x14

Table 977. TRIG_FORCE Register

Bits	Description	Type	Reset
31:4	Reserved.	-	-

Bits	Description	Type	Reset
3:0	<p>Simulate the firing of one or more detectors. Writing ones to this register will set the matching bits in STATUS_TRIG.</p> <p>If the glitch detectors are currently armed, writing ones will also immediately reset the switched core power domain, and set the reset reason latches in POWMAN_CHIP_RESET to indicate a glitch detector resets.</p> <p>This register is Secure read/write only.</p>	SC	0x0

10.8. Factory Test JTAG

RP2350 contains JTAG hardware that is used to test devices after manufacturing. It is not a public interface, but its capabilities are documented here for user risk assessment.

Much like the user-facing SWD debug, the JTAG interface is disabled at power-on, and enabled only once the OTP power-on state machine has completed. If the [CRIT1.SECURE_BOOT_ENABLE](#), [CRIT1.SECURE_DEBUG_DISABLE](#) or [CRIT1.DEBUG_DISABLE](#) flag is set, then the JTAG interface remains held in reset indefinitely, so it cannot be communicated with and cannot control internal hardware. The only way to re-enable the JTAG interface after setting one of these critical flags is to set the RMA OTP flag ([Section 10.9](#)), which also permanently disables read and write access to user OTP pages. The RMA flag itself is write-protected using the page 63 protection flags, so you can prevent untrusted software from programming the RMA flag.

To take the JTAG interface out of reset, write to bit 0 of the RP-AP control register, accessed via SWD. To connect the JTAG interface to GPIOs ([TCK](#), [TMS](#), [TDI](#), [TDO](#) on $\text{GPIO0} \rightarrow 3$), set bit 1 of the RP-AP control register. The RP-AP is always accessible, even when external debug is disabled, because it is also used to enter the debug keys ([Section 3.5.9.2](#)). However, attempts to remove the JTAG reset are ignored when any of the aforementioned critical OTP flags are set.

The JTAG interface provides:

- Standard test capabilities such as IDCODE and EXTEST (boundary scan); these are *not* guaranteed to be IEEE-compliant, as this is an internal factory test interface, not a user-facing debug port
- Full AHB bus access, with Secure and Privileged attributes and an HMASTER of 3 (debugger)
- Asynchronous access to a small subset of register controls, generally limited to clocks, oscillators and reset controls

The JTAG interface's AHB bus access is muxed in place of the DMA write port, when the JTAG interface is enabled.

Any and all details of the factory test JTAG interface, with the exception of which OTP flags disable and re-enable it, are subject to change with revisions of the RP2350 silicon.

10.9. Decommissioning

Devices returned to Raspberry Pi Ltd for fault analysis must be **decommissioned** before return, to restore factory test functionality. A device is decommissioned by programming the OTP [PAGE63_LOCK0.RMA](#) flag to 1. Return may be requested by Raspberry Pi Ltd when diagnosing systematic issues across a population of devices.

Setting the RMA flag has two effects:

- The factory test JTAG interface is re-enabled, irrespective of the values of any [CRIT1](#) flags.
- Pages 3 through 61 become permanently inaccessible: this is all pages which do not have predefined contents listed in [Section 13.9](#).

The effect on OTP contents is as though all had been promoted to the inaccessible lock level:

- write attempts will fail
- read attempts will return all-ones, when read via an unguarded read alias, or bus faults, when read via a guarded read alias

The logic which disables OTP access and the logic which re-enables the test interface are driven from the same signal internally, so this bit does not provide external access to user OTP contents, provided no sensitive material is stored in pages 0, 1, 2, 62 or 63. Setting the RMA flag is irreversible, and may render the device permanently unusable, if it is configured to boot from OTP contents stored in pages 3 through 61.

After setting the RMA flag, test the OTP access (e.g. via the SWD interface) and verify for yourself that any sensitive data stored in OTP has been made inaccessible.

The page 63 lock word has no other function besides RMA, since pages 62/63 contain the lock words themselves, each of which is protected by its own permissions. This means the RMA flag can be write-protected by setting either a hard or soft lock on page 63.

Chapter 11. PIO

Chapter 12. Peripherals

12.1. UART

Arm Documentation

Excerpted from the [PrimeCell UART \(PL011\) Technical Reference Manual](#). Used with permission.

RP2350 has 2 identical instances of a UART peripheral, based on the Arm Primecell UART (PL011) (Revision r1p5).

Each instance supports the following features:

- Separate 32×8 Tx and 32×12 Rx FIFOs
- Programmable baud rate generator, clocked by `clk_peri` (see [Figure 30](#))
- Standard asynchronous communication bits (start, stop, parity) added on transmit and removed on receive
- Line break detection
- Programmable serial interface (5, 6, 7, or 8 bits)
- 1 or 2 stop bits
- Programmable hardware flow control

Each UART can be connected to a number of GPIO pins as defined in the GPIO muxing [table](#) in [Section 9.4](#). Connections to the GPIO muxing use a prefix including the UART instance name `uart0_` or `uart1_`, and include the following:

- Transmit data `tx` (referred to as `UARTTXD` in the following sections)
- Received data `rx` (referred to as `UARTRXD` in the following sections)
- Output flow control `rts` (referred to as `nUARTRTS` in the following sections)
- Input flow control `cts` (referred to as `nUARTCTS` in the following sections)

The modem mode and IrDA mode of the PL011 are not supported.

The `UARTCLK` is driven from `clk_peri`, and `PCLK` is driven from the system clock `clk_sys` (see [Figure 30](#)).

12.1.1. Overview

The UART performs:

- Serial-to-parallel conversion on data received from a peripheral device
- Parallel-to-serial conversion on data transmitted to the peripheral device

The CPU reads and writes data and control/status information through the AMBA APB interface. The transmit and receive paths are buffered with internal FIFO memories that store up to 32 bytes independently in both transmit and receive modes.

The UART:

- Includes a programmable baud rate generator that generates a common transmit and receive internal clock from the UART internal reference clock input, `UARTCLK`
- Offers similar functionality to the industry-standard 16C650 UART device
- Supports a maximum baud rate of `UARTCLK` / 16 in UART mode (7.8 Mbaud at 125MHz)

The UART operation and baud rate values are controlled by the Line Control Register ([UARTLCR_H](#)) and the baud rate divisor registers: Integer Baud Rate Register ([UARTIBRD](#)), and Fractional Baud Rate Register ([UARTFBRD](#)).

The UART can generate:

- Individually maskable interrupts from the receive (including timeout), transmit, modem status and error conditions
- A single combined interrupt so that the output is asserted if any of the individual interrupts are asserted and unmasked
- DMA request signals for interfacing with a Direct Memory Access (DMA) controller

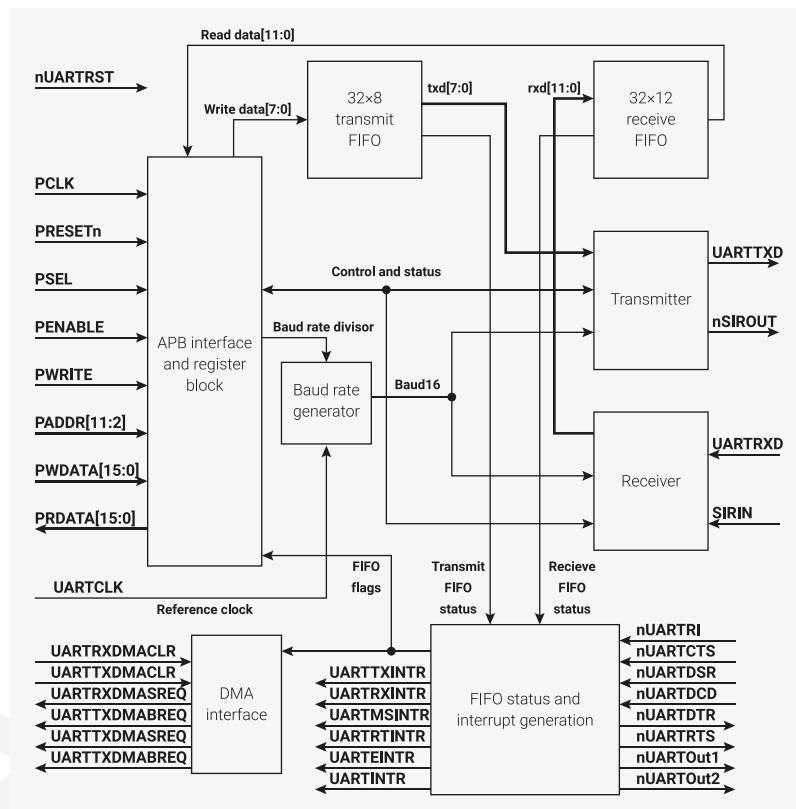
If a framing, parity, or break error occurs during reception, the appropriate error bit is set and stored in the FIFO. If an overrun condition occurs, the overrun register bit is set immediately and FIFO data is prevented from being overwritten.

You can program the FIFOs to be 1-byte deep providing a conventional double-buffered UART interface.

There is a programmable hardware flow control feature that uses the [nUARTCTS](#) input and the [nUARTRTS](#) output to automatically control the serial data flow.

12.1.2. Functional description

Figure 41. UART block diagram. Test logic is not shown for clarity.



12.1.2.1. AMBA APB interface

The AMBA APB interface generates read and write decodes for accesses to status/control registers, and the transmit and receive FIFOs.

12.1.2.2. Register block

The register block stores data written, or to be read across the AMBA APB interface.

12.1.2.3. Baud rate generator

The baud rate generator contains free-running counters that generate the internal clocks: Baud16 and IrLPBaud16 signals. Baud16 provides timing information for UART transmit and receive control. Baud16 is a stream of pulses with a width of one [UARTCLK](#) clock period and a frequency of 16 times the baud rate.

12.1.2.4. Transmit FIFO

The transmit FIFO is an 8-bit wide, 32 location deep, FIFO memory buffer. CPU data written across the APB interface is stored in the FIFO until read out by the transmit logic. When disabled, the transmit FIFO acts like a one byte holding register.

12.1.2.5. Receive FIFO

The receive FIFO is a 12-bit wide, 32 location deep, FIFO memory buffer. Received data and corresponding error bits are stored in the receive FIFO by the receive logic until read out by the CPU across the APB interface. When disabled, the receive FIFO acts like a one byte holding register.

12.1.2.6. Transmit logic

The transmit logic performs parallel-to-serial conversion on the data read from the transmit FIFO. Control logic outputs the serial bit stream in the following order:

1. Start bit
2. Data bits (Least Significant Bit (LSB) first)
3. Parity bit
4. Stop bits according to the programmed configuration in control registers

12.1.2.7. Receive logic

The receive logic performs serial-to-parallel conversion on the received bit stream after a valid start pulse has been detected. Receive logic includes overrun, parity, frame error checking, and line break detection; you can find the output of these checks in the status that accompanies the data written to the receive FIFO.

12.1.2.8. Interrupt generation logic

The UART generates individual maskable active HIGH interrupts to the processor interrupt controllers. To generate combined interrupts, the UART outputs an OR function of the individual interrupt requests.

For more information, see [Section 12.1.6](#).

12.1.2.9. DMA interface

The UART provides an interface to connect to the DMA controller as a UART DMA; for more information, see [Section 12.1.5](#).

12.1.2.10. Synchronizing registers and logic

The UART supports both asynchronous and synchronous operation of the clocks, [PCLK](#) and [UARTCLK](#). The UART implements always-on synchronisation registers and handshaking logic. This has a minimal impact on performance and area. The UART performs control signal synchronisation on both directions of data flow (from the [PCLK](#) to the [UARTCLK](#) domain, and from the [UARTCLK](#) to the [PCLK](#) domain).

12.1.3. Operation

12.1.3.1. Clock signals

The frequency selected for [UARTCLK](#) must accommodate the required range of baud rates:

- [UARTCLK](#) (min) $\geq 16 \times$ baud_rate (max)
- [UARTCLK](#) (max) $\leq 16 \times 65535 \times$ baud_rate (min)

For example, for a range of baud rates from 110 baud to 460800 baud the [UARTCLK](#) frequency must be between 7.3728MHz to 115.34MHz.

To use all baud rates, the [UARTCLK](#) frequency must fall within the required error limits.

There is also a constraint on the ratio of clock frequencies for [PCLK](#) to [UARTCLK](#). The frequency of [UARTCLK](#) must be no more than 5/3 times faster than the frequency of [PCLK](#):

- $\text{UARTCLK} \leq 5/3 \times \text{PCLK}$

For example, in UART mode, to generate 921600 baud when [UARTCLK](#) is 14.7456MHz, [PCLK](#) must be greater than or equal to 8.85276MHz. This ensures that the UART has sufficient time to write the received data to the receive FIFO.

12.1.3.2. UART operation

Control data is written to the UART Line Control Register, [UARTLCR](#). This register is 30 bits wide internally, but provides external access through the APB interface by writes to the following registers:

- [UARTLCR_H](#), which defines the following:
 - transmission parameters
 - word length
 - buffer mode
 - number of transmitted stop bits
 - parity mode
 - break generation
- [UARTIBRD](#), which defines the integer baud rate divider
- [UARTFBRD](#), which defines the fractional baud rate divider

12.1.3.2.1. Fractional baud rate divider

The baud rate divisor is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. The baud rate generator uses the baud rate divisor to determine the bit period. The fractional baud rate divider enables the use of any clock with a frequency greater than 3.6864MHz to act as [UARTCLK](#), while it is still possible to generate all the standard baud rates.

The 16-bit integer is written to the Integer Baud Rate Register, [UARTIBRD](#). The 6-bit fractional part is written to the Fractional Baud Rate Register, [UARTFBRD](#). The Baud Rate Divisor has the following relationship to [UARTCLK](#):

Baud Rate Divisor = $\text{UARTCLK}/(16 \times \text{Baud Rate}) = BRD_I + BRD_F$ where BRD_I is the integer part and BRD_F is the fractional part separated by a decimal point as shown in Figure 42.

Figure 42. Baud rate divisor.



To calculate the 6-bit number (m), multiply the fractional part of the required baud rate divisor by 64 (2^n , where n is the width of the **UARTFBRD** register) and add 0.5 to account for rounding errors:

$$m = \text{integer}(BRD_F \times 2^n + 0.5)$$

The UART generates an internal clock enable signal, Baud16. This is a stream of **UARTCLK**-wide pulses with an average frequency of 16 times the required baud rate. Divide this signal by 16 to give the transmit clock. A low number in the baud rate divisor produces a short bit period, and a high number in the baud rate divisor produces a long bit period.

12.1.3.2.2. Data transmission or reception

The UART uses two 32-byte FIFOs to store data received and transmitted. The receive FIFO has an extra four bits per character for status information. For transmission, data is written into the transmit FIFO. If the UART is enabled, it causes a data frame to start transmitting with the parameters indicated in the Line Control Register, **UARTLCR_H**. Data continues to be transmitted until there is no data left in the transmit FIFO. The **BUSY** signal goes HIGH immediately after data writes to the transmit FIFO (that is, the FIFO is non-empty) and remains asserted HIGH while data transmits. **BUSY** is negated only when the transmit FIFO is empty, and the last character has been transmitted from the shift register, including the stop bits. **BUSY** can be asserted HIGH even though the UART might no longer be enabled.

For each sample of data, three readings are taken and the majority value is kept. In the following paragraphs, the middle sampling point is defined, and one sample is taken either side of it.

When the receiver is idle (**UARTRXD** continuously 1, in the marking state) and a LOW is detected on the data input (a start bit has been received), the receive counter, with the clock enabled by Baud16, begins running and data is sampled on the eighth cycle of that counter in UART mode, or the fourth cycle of the counter in SIR mode to allow for the shorter logic 0 pulses (half way through a bit period).

The start bit is valid if **UARTRXD** is still LOW on the eighth cycle of Baud16, otherwise a false start bit is detected and it is ignored.

If the start bit was valid, successive data bits are sampled on every 16th cycle of Baud16 (that is, one bit period later) according to the programmed length of the data characters. The parity bit is then checked if parity mode was enabled.

Lastly, a valid stop bit is confirmed if **UARTRXD** is HIGH, otherwise a framing error has occurred. When a full word is received, the data is stored in the receive FIFO, with any error bits associated with that word

12.1.3.2.3. Error bits

The receive FIFO stores three error bits in bits 8 (framing), 9 (parity), and 10 (break), each associated with a particular character. An additional error bit, stored in bit 11 of the receive FIFO, indicates an overrun error.

12.1.3.2.4. Overrun bit

The overrun bit is not associated with the character in the receive FIFO. The overrun error is set when the FIFO is full and the next character is completely received in the shift register. The data in the shift register is overwritten, but it is not written into the FIFO. When an empty location becomes available in the FIFO, another character is received and the state of the overrun bit is copied into the receive FIFO along with the received character. The overrun state is then cleared. Table 978 lists the bit functions of the receive FIFO.

Table 978. Receive FIFO bit functions

FIFO bit	Function
11	Overrun indicator
10	Break error
9	Parity error
8	Framing error
7:0	Received data

12.1.3.2.5. Disabling the FIFOs

The bottom entry of the transmit and receive sides of the UART both have the equivalent of a 1-byte holding register. You can manipulate flags to disable the FIFOs, allowing you to use the bottom entry of the FIFOs as a 1-byte register. However, this doesn't physically disable the FIFOs. When using the FIFOs as a 1-byte register, a write to the data register bypasses the holding register unless the transmit shift register is already in use.

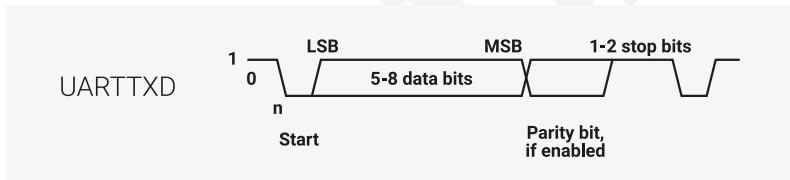
12.1.3.2.6. System and diagnostic loopback testing

To perform loopback testing for UART data, set the Loop Back Enable (LBE) bit to 1 in the Control Register, [UARTCR](#).

Data transmitted on [UARTTXD](#) is received on the [UARTRXD](#) input.

12.1.3.3. UART character frame

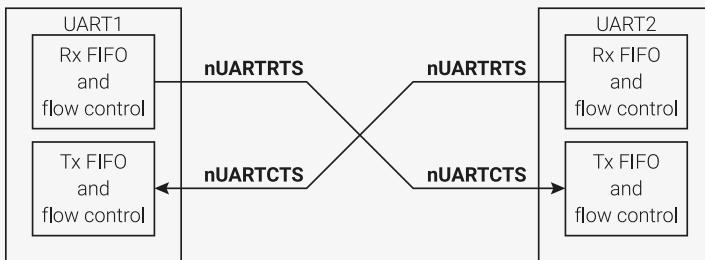
Figure 43. UART character frame.



12.1.4. UART hardware flow control

The fully-selectable hardware flow control feature enables you to control the serial data flow with the [nUARTRTS](#) output and [nUARTCTS](#) input signals. [Figure 44](#) shows how to communicate between two devices using hardware flow control:

Figure 44. Hardware flow control between two similar devices.



When the RTS flow control is enabled, [nUARTRTS](#) is asserted until the receive FIFO is filled up to the programmed watermark level. When the CTS flow control is enabled, the transmitter can only transmit data when [nUARTCTS](#) is asserted.

The hardware flow control is selectable using the [RTSEn](#) and [CTSEn](#) bits in the Control Register, [UARTCR](#). [Table 979](#) shows how to configure [UARTCR](#) register bits to enable RTS and/or CTS.

Table 979. Control bits to enable and disable hardware flow control.

UARTCR register bits		
CTSEn	RTSEn	Description
1	1	Both RTS and CTS flow control enabled
1	0	Only CTS flow control enabled
0	1	Only RTS flow control enabled
0	0	Both RTS and CTS flow control disabled

NOTE

When RTS flow control is enabled, the software cannot use the RTSEn bit in the Control Register (UARTCR) to control the status of [nUARTRTS](#).

12.1.4.1. RTS flow control

The RTS flow control logic is linked to the programmable receive FIFO watermark levels.

When RTS flow control is disabled, the receive FIFO receives data until full, or no more data is transmitted to it.

When RTS flow control is enabled, the [nUARTRTS](#) is asserted until the receive FIFO fills up to the watermark level. When the receive FIFO reaches the watermark level, the [nUARTRTS](#) signal is deasserted. This indicates that the FIFO has no more room to receive data. The transmission of data is expected to cease after the current character has been transmitted. When the receive FIFO drains below the watermark level, the [nUARTRTS](#) signal is reasserted.

12.1.4.2. CTS flow control

The CTS flow control logic is linked to the [nUARTCTS](#) signal.

When CTS flow control is disabled, the transmitter transmits data until the transmit FIFO is empty.

When CTS flow control is enabled, the transmitter checks the [nUARTCTS](#) signal before transmitting each byte. It only transmits the byte if the [nUARTCTS](#) signal is asserted. As long as the transmit FIFO is not empty and [nUARTCTS](#) is asserted, data continues to transmit. If the transmit FIFO is empty and the [nUARTCTS](#) signal is asserted, no data is transmitted. If the [nUARTCTS](#) signal is deasserted during transmission, the transmitter finishes transmitting the current character before stopping.

12.1.5. UART DMA Interface

The UART provides an interface to connect to a DMA controller. The DMA operation of the UART is controlled using the DMA Control Register, [UARTDMACR](#). The DMA interface includes the following signals:

For receive:

[UARTRXDMASREQ](#)

Single character DMA transfer request, asserted by the UART. For receive, one character consists of up to 12 bits. This signal is asserted when the receive FIFO contains at least one character.

[UARTRXDMABREQ](#)

Burst DMA transfer request, asserted by the UART. This signal is asserted when the receive FIFO contains more characters than the programmed watermark level. You can program the watermark level for each FIFO using the Interrupt FIFO Level Select Register ([UARTIFLS](#)).

UARTRXDMACLR

DMA request clear, asserted by a DMA controller to clear the receive request signals. If DMA burst transfer is requested, the clear signal is asserted during the transfer of the last data in the burst.

For transmit:

UARTRXDMASREQ

Single character DMA transfer request, asserted by the UART. For transmit, one character consists of up to eight bits. This signal is asserted when there is at least one empty location in the transmit FIFO.

UARTRXDMABREQ

Burst DMA transfer request, asserted by the UART. This signal is asserted when the transmit FIFO contains less characters than the watermark level. You can program the watermark level for each FIFO using the Interrupt FIFO Level Select Register ([UARTIFLS](#)).

UARTRXDMACLR

DMA request clear, asserted by a DMA controller to clear the transmit request signals. If DMA burst transfer is requested, the clear signal is asserted during the transfer of the last data in the burst.

The burst transfer and single transfer request signals are not mutually exclusive: they can both be asserted at the same time. When the receive FIFO exceeds the watermark level, the burst transfer request and the single transfer request signals are both asserted. When the receive FIFO is below than the watermark level, only the single transfer request signal is asserted. This is useful in situations where the number of characters left to be received in the stream is less than a burst.

Consider a scenario where the watermark level is set to four, but 19 characters are left to be received. The DMA controller then transfers four bursts of four characters and three single transfers to complete the stream.

NOTE

For the remaining three characters, the UART cannot assert the burst request.

Each request signal remains asserted until the relevant DMACLR signal is asserted. After the request clear signal is deasserted, a request signal can become active again, depending on the conditions described previously. All request signals are deasserted if the UART is disabled or the relevant DMA enable bit, **TXDMAE** or **RXDMAE**, in the DMA Control Register, [UARTDMACR](#), is cleared.

If you disable the FIFOs in the UART, it operates in character mode. Character mode limits FIFO transfers to a single character at a time, so only the DMA single transfer mode can operate. In character mode, only the [UARTRXDMASREQ](#) and [UARTRXDMABREQ](#) request signals can be asserted. For information about disabling the FIFOs, see the Line Control Register, [UARTLCR_H](#).

When the UART is in the FIFO enabled mode, data transfers can use either single or burst transfers depending on the programmed watermark level and the amount of data in the FIFO. [Table 980](#) lists the trigger points for [UARTRXDMABREQ](#) and [UARTRXDMABREQ](#), depending on the watermark level, for the transmit and receive FIFOs.

Table 980. DMA trigger points for the transmit and receive FIFOs.

Watermark level	Burst length	
	Transmit (number of empty locations)	Receive (number of filled locations)
1/8	28	4
1/4	24	8
1/2	16	16
3/4	8	24
7/8	4	28

In addition, the **DMAONERR** bit in the DMA Control Register, [UARTDMACR](#), supports the use of the receive error interrupt,

UARTEINTR. It enables the DMA receive request outputs, **UARTRXDMASREQ** or **UARTRXDMABREQ**, to be masked out when the UART error interrupt, **UARTEINTR**, is asserted. The DMA receive request outputs remain inactive until the **UARTEINTR** is cleared. The DMA transmit request outputs are unaffected.

Figure 45. DMA transfer waveforms.

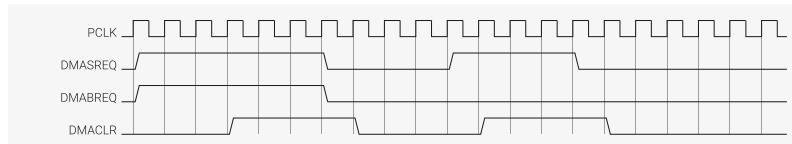


Figure 45 shows the timing diagram for both a single transfer request and a burst transfer request with the appropriate **DMACLR** signal. The signals are all synchronous to PCLK. For the sake of clarity it is assumed that there is no synchronization of the request signals in the DMA controller.

12.1.6. Interrupts

There are eleven maskable interrupts generated in the UART. On RP2350, only the combined interrupt output, **UARTINTR**, is connected.

To enable or disable individual interrupts, change the mask bits in the Interrupt Mask Set/Clear Register, **UARTIMSC**. Set the appropriate mask bit HIGH to enable the interrupt.

The transmit and receive dataflow interrupts **UARTRXINTR** and **UARTTXINTR** have been separated from the status interrupts. This enables you to use **UARTRXINTR** and **UARTTXINTR** to read or write data in response to FIFO trigger levels.

The error interrupt, **UARTEINTR**, can be triggered when there is an error in the reception of data. A number of error conditions are possible.

The modem status interrupt, **UARTMSINTR**, is a combined interrupt of all the individual modem status signals.

The status of the individual interrupt sources can be read either from the Raw Interrupt Status Register, **UARTRIS**, or from the Masked Interrupt Status Register, **UARTMIS**.

12.1.6.1. **UARTMSINTR**

The modem status interrupt is asserted if any of the modem status signals (**nUARTCTS**, **nUARTOCD**, **nUARTDSR**, and **nUARTRI**) change. To clear the modem status interrupt, write a 1 to the bits corresponding to the modem status signals that generated the interrupt in the Interrupt Clear Register (**UARTICR**).

12.1.6.2. **UARTRXINTR**

The receive interrupt changes state when one of the following events occurs:

- The FIFOs are enabled and the receive FIFO reaches the programmed trigger level. This asserts the receive interrupt HIGH. To clear the receive interrupt, read data from the receive FIFO until it drops below the trigger level.
- The FIFOs are disabled (have a depth of one location) and data is received, thereby filling the receive FIFO. This asserts the receive interrupt HIGH. To clear the receive interrupt, perform a single read from the receive FIFO.

In both cases, you can also clear the interrupt manually.

12.1.6.3. **UARTTXINTR**

The transmit interrupt changes state when one of the following events occurs:

- The FIFOs are enabled and the transmit FIFO is equal to or lower than the programmed trigger level. This asserts the transmit interrupt HIGH. To clear the transmit interrupt, write data to the transmit FIFO until it exceeds the

trigger level.

- The FIFOs are disabled (have a depth of one location) and there is no data present in the transmit FIFO. This asserts the transmit interrupt HIGH. To clear the transmit interrupt, perform a single write to the transmit FIFO.

In both cases, you can also clear the interrupt manually.

To update the transmit FIFO, write data to the transmit FIFO before or after enabling the UART and the interrupts.

NOTE

The transmit interrupt is based on a transition through a level, rather than on the level itself. When the interrupt and the UART is enabled before any data is written to the transmit FIFO, the interrupt is not set. The interrupt is only set after written data leaves the single location of the transmit FIFO and it becomes empty.

12.1.6.4. [UARTRTINTR](#)

The receive timeout interrupt is asserted when the receive FIFO is not empty and no more data is received during a 32-bit period.

The receive timeout interrupt is cleared in the following scenarios:

- the FIFO becomes empty through reading all the data or by reading the holding register
- a 1 is written to the corresponding bit of the Interrupt Clear Register, [UARTICR](#)

12.1.6.5. [UARTEINTR](#)

The error interrupt is asserted when an error occurs in the reception of data by the UART. The interrupt can be caused by a number of different error conditions:

- framing
- parity
- break
- overrun

To determine the cause of the interrupt, read the Raw Interrupt Status Register ([UARTRIS](#)) or the Masked Interrupt Status Register ([UARTMIS](#)). To clear the interrupt, write to the relevant bits of the Interrupt Clear Register, [UARTICR](#) (bits 7 to 10 are the error clear bits).

12.1.6.6. [UARTINTR](#)

The interrupts are also combined into a single output, that is an OR function of the individual masked sources. You can connect this output to a system interrupt controller to provide another level of masking on a individual peripheral basis.

The combined UART interrupt is asserted if any of the individual interrupts are asserted and enabled.

12.1.7. Programmer's Model

The SDK provides a [uart_init](#) function to configure the UART with a particular baud rate. Once the UART is initialised, the user must configure a GPIO pin as [UART_TX](#) and [UART_RX](#). See [Section 9.10.1](#) for more information on selecting a GPIO function.

To initialise the UART, the [uart_init](#) function takes the following steps:

1. De-asserts the reset

2. Enables `clk_peri`
3. Sets enable bits in the control register
4. Enables the FIFOs
5. Sets the baud rate divisors
6. Sets the format

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_uart/uart.c Lines 42 - 92

```

42 uint uart_init(uart_inst_t *uart, uint baudrate) {
43     invalid_params_if(HARDWARE_UART, uart != uart0 && uart != uart1);
44
45     if (uart_clock_get_hz(uart) == 0) {
46         return 0;
47     }
48
49     uart_reset(uart);
50     uart_unreset(uart);
51
52 #if PICO_UART_ENABLE_CRLF_SUPPORT
53     uart_set_translate_crlf(uart, PICO_UART_DEFAULT_CRLF);
54 #endif
55
56     // Any LCR writes need to take place before enabling the UART
57     uint baud = uart_set_baudrate(uart, baudrate);
58
59     // inline the uart_set_format() call, as we don't need the CR disable/re-enable
60     // protection, and also many people will never call it again, so having
61     // the generic function is not useful, and much bigger than this inlined
62     // code which is only a handful of instructions.
63     //
64     // The UART_UARTLCR_H_FEN_BITS setting is combined as well as it is the same register
65 #if 0
66     uart_set_format(uart, 8, 1, UART_PARITY_NONE);
67     // Enable FIFOs (must be before setting UARLEN, as this is an LCR access)
68     hw_set_bits(&uart_get_hw(uart)->lcr_h, UART_UARTLCR_H_FEN_BITS);
69 #else
70     uint data_bits = 8;
71     uint stop_bits = 1;
72     uint parity = UART_PARITY_NONE;
73     hw_write_masked(&uart_get_hw(uart)->lcr_h,
74         ((data_bits - 5u) << UART_UARTLCR_H_WLEN_LSB) |
75             ((stop_bits - 1u) << UART_UARTLCR_H_STP2_LSB) |
76             (bool_to_bit(parity != UART_PARITY_NONE) << UART_UARTLCR_H_PEN_LSB) |
77             (bool_to_bit(parity == UART_PARITY_EVEN) << UART_UARTLCR_H_EPS_LSB) |
78             UART_UARTLCR_H_FEN_BITS,
79         UART_UARTLCR_H_WLEN_BITS | UART_UARTLCR_H_STP2_BITS |
80             UART_UARTLCR_H_PEN_BITS | UART_UARTLCR_H_EPS_BITS |
81             UART_UARTLCR_H_FEN_BITS);
82 #endif
83
84     // Enable the UART, both TX and RX
85     uart_get_hw(uart)->cr = UART_UARTCR_UARLEN_BITS | UART_UARTCR_TXE_BITS |
86     UART_UARTCR_RXE_BITS;
87 #if !PICO_UART_NO_DMACR_ENABLE
88     // Always enable DREQ signals -- no harm in this if DMA is not listening
89     uart_get_hw(uart)->dmacr = UART_UARTDMACR_TXDMAE_BITS | UART_UARTDMACR_RXDMAE_BITS;
90 #endif
91     return baud;

```

92 }

12.1.7.1. Baud Rate Calculation

The UART baud rate is derived from dividing `clk_peri`.

If the required baud rate is 115200 and $\text{UARTCLK} = 125\text{MHz}$ then:

$$\text{Baud Rate Divisor} = (125 \times 10^6) / (16 \times 115200) \approx 67.817$$

Therefore, $\text{BRDI} = 67$ and $\text{BRDF} = 0.817$,

Therefore, fractional part, $m = \text{integer}((0.817 \times 64) + 0.5) = 52$

Generated baud rate divider = $67 + 52/64 = 67.8125$

Generated baud rate = $(125 \times 10^6) / (16 \times 67.8125) \approx 115207$

Error = $(\text{abs}(115200 - 115207) / 115200) \times 100 \approx 0.006\%$

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_uart/uart.c Lines 155 - 180

```

155 uint uart_set_baudrate(uart_inst_t *uart, uint baudrate) {
156     invalid_params_if(HARDWARE_UART, baudrate == 0);
157     uint32_t baud_rate_div = (8 * uart_clock_get_hz(uart) / baudrate) + 1;
158     uint32_t baud_ibrd = baud_rate_div >> 7;
159     uint32_t baud_fbrd;
160
161     if (baud_ibrd == 0) {
162         baud_ibrd = 1;
163         baud_fbrd = 0;
164     } else if (baud_ibrd >= 65535) {
165         baud_ibrd = 65535;
166         baud_fbrd = 0;
167     } else {
168         baud_fbrd = (baud_rate_div & 0x7f) >> 1;
169     }
170
171     uart_get_hw(uart)->ibrd = baud_ibrd;
172     uart_get_hw(uart)->fbrd = baud_fbrd;
173
174     // PL011 needs a (dummy) LCR_H write to latch in the divisors.
175     // We don't want to actually change LCR_H contents here.
176     uart_write_lcr_bits_masked(uart, 0, 0);
177
178     // See datasheet
179     return (4 * uart_clock_get_hz(uart)) / (64 * baud_ibrd + baud_fbrd);
180 }
```

12.1.8. List of Registers

The UART0 and UART1 registers start at base addresses of `0x40070000` and `0x40078000` respectively (defined as `UART0_BASE` and `UART1_BASE` in SDK).

Table 981. List of
UART registers

Offset	Name	Info
0x000	UARTDR	Data Register, UARTDR
0x004	UARTRSR	Receive Status Register/Error Clear Register, UARTRSR/UARTECR

Offset	Name	Info
0x018	UARTFR	Flag Register, UARTFR
0x020	UARTILPR	IrDA Low-Power Counter Register, UARTILPR
0x024	UARTIBRD	Integer Baud Rate Register, UARTIBRD
0x028	UARTFBRD	Fractional Baud Rate Register, UARTFBRD
0x02c	UARTLCR_H	Line Control Register, UARTLCR_H
0x030	UARTCR	Control Register, UARTCR
0x034	UARTIFLS	Interrupt FIFO Level Select Register, UARTIFLS
0x038	UARTIMSC	Interrupt Mask Set/Clear Register, UARTIMSC
0x03c	UARTRIS	Raw Interrupt Status Register, UARTRIS
0x040	UARTMIS	Masked Interrupt Status Register, UARTMIS
0x044	UARTICR	Interrupt Clear Register, UARTICR
0x048	UARTDMACR	DMA Control Register, UARTDMACR
0xfe0	UARTPERIPHID0	UARTPeriphID0 Register
0xfe4	UARTPERIPHID1	UARTPeriphID1 Register
0xfe8	UARTPERIPHID2	UARTPeriphID2 Register
0xfc	UARTPERIPHID3	UARTPeriphID3 Register
0xff0	UARTPCELLID0	UARTPCellID0 Register
0xff4	UARTPCELLID1	UARTPCellID1 Register
0xff8	UARTPCELLID2	UARTPCellID2 Register
0ffc	UARTPCELLID3	UARTPCellID3 Register

UART: UARTDR Register

Offset: 0x000

Description

Data Register, UARTDR

Table 982. UARTDR Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	OE	Overrun error. This bit is set to 1 if data is received and the receive FIFO is already full. This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it.	RO	-
10	BE	Break error. This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits). In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received.	RO	-

Bits	Name	Description	Type	Reset
9	PE	Parity error. When set to 1, it indicates that the parity of the received data character does not match the parity that the EPS and SPS bits in the Line Control Register, UARTLCR_H. In FIFO mode, this error is associated with the character at the top of the FIFO.	RO	-
8	FE	Framing error. When set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). In FIFO mode, this error is associated with the character at the top of the FIFO.	RO	-
7:0	DATA	Receive (read) data character. Transmit (write) data character.	RWF	-

UART: UARTRSR Register

Offset: 0x004

Description

Receive Status Register/Error Clear Register, UARTRSR/UARTECR

Table 983. UARTRSR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	OE	Overrun error. This bit is set to 1 if data is received and the FIFO is already full. This bit is cleared to 0 by a write to UARTECR. The FIFO contents remain valid because no more data is written when the FIFO is full, only the contents of the shift register are overwritten. The CPU must now read the data, to empty the FIFO.	WC	0x0
2	BE	Break error. This bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity, and stop bits). This bit is cleared to 0 after a write to UARTECR. In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state) and the next valid start bit is received.	WC	0x0
1	PE	Parity error. When set to 1, it indicates that the parity of the received data character does not match the parity that the EPS and SPS bits in the Line Control Register, UARTLCR_H. This bit is cleared to 0 by a write to UARTECR. In FIFO mode, this error is associated with the character at the top of the FIFO.	WC	0x0
0	FE	Framing error. When set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). This bit is cleared to 0 by a write to UARTECR. In FIFO mode, this error is associated with the character at the top of the FIFO.	WC	0x0

UART: UARTFR Register

Offset: 0x018

Description

Flag Register, UARTFR

Table 984. UARTFR Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	RI	Ring indicator. This bit is the complement of the UART ring indicator, nUARTRI, modem status input. That is, the bit is 1 when nUARTRI is LOW.	RO	-
7	TXFE	Transmit FIFO empty. The meaning of this bit depends on the state of the FEN bit in the Line Control Register, UARTLCR_H. If the FIFO is disabled, this bit is set when the transmit holding register is empty. If the FIFO is enabled, the TXFE bit is set when the transmit FIFO is empty. This bit does not indicate if there is data in the transmit shift register.	RO	0x1
6	RXFF	Receive FIFO full. The meaning of this bit depends on the state of the FEN bit in the UARTLCR_H Register. If the FIFO is disabled, this bit is set when the receive holding register is full. If the FIFO is enabled, the RXFF bit is set when the receive FIFO is full.	RO	0x0
5	TXFF	Transmit FIFO full. The meaning of this bit depends on the state of the FEN bit in the UARTLCR_H Register. If the FIFO is disabled, this bit is set when the transmit holding register is full. If the FIFO is enabled, the TXFF bit is set when the transmit FIFO is full.	RO	0x0
4	RXFE	Receive FIFO empty. The meaning of this bit depends on the state of the FEN bit in the UARTLCR_H Register. If the FIFO is disabled, this bit is set when the receive holding register is empty. If the FIFO is enabled, the RXFE bit is set when the receive FIFO is empty.	RO	0x1
3	BUSY	UART busy. If this bit is set to 1, the UART is busy transmitting data. This bit remains set until the complete byte, including all the stop bits, has been sent from the shift register. This bit is set as soon as the transmit FIFO becomes non-empty, regardless of whether the UART is enabled or not.	RO	0x0
2	DCD	Data carrier detect. This bit is the complement of the UART data carrier detect, nUARTDCD, modem status input. That is, the bit is 1 when nUARTDCD is LOW.	RO	-
1	DSR	Data set ready. This bit is the complement of the UART data set ready, nUARTDSR, modem status input. That is, the bit is 1 when nUARTDSR is LOW.	RO	-
0	CTS	Clear to send. This bit is the complement of the UART clear to send, nUARTCTS, modem status input. That is, the bit is 1 when nUARTCTS is LOW.	RO	-

UART: UARTILPR Register

Offset: 0x020

Description

IrDA Low-Power Counter Register, UARTILPR

Table 985. UARTILPR Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	ILPDVSR	8-bit low-power divisor value. These bits are cleared to 0 at reset.	RW	0x00

UART: UARTIBRD Register

Offset: 0x024

Description

Integer Baud Rate Register, UARTIBRD

Table 986. UARTIBRD Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:0	BAUD_DIVINT	The integer baud rate divisor. These bits are cleared to 0 on reset.	RW	0x0000

UART: UARTFBRD Register

Offset: 0x028

Description

Fractional Baud Rate Register, UARTFBRD

Table 987. UARTFBRD Register

Bits	Name	Description	Type	Reset
31:6	Reserved.	-	-	-
5:0	BAUD_DIVFRAC	The fractional baud rate divisor. These bits are cleared to 0 on reset.	RW	0x00

UART: UARTLCR_H Register

Offset: 0x02c

Description

Line Control Register, UARTLCR_H

Table 988. UARTLCR_H Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	SPS	Stick parity select. 0 = stick parity is disabled 1 = either: * if the EPS bit is 0 then the parity bit is transmitted and checked as a 1 * if the EPS bit is 1 then the parity bit is transmitted and checked as a 0. This bit has no effect when the PEN bit disables parity checking and generation.	RW	0x0
6:5	WLEN	Word length. These bits indicate the number of data bits transmitted or received in a frame as follows: b11 = 8 bits b10 = 7 bits b01 = 6 bits b00 = 5 bits.	RW	0x0

Bits	Name	Description	Type	Reset
4	FEN	Enable FIFOs: 0 = FIFOs are disabled (character mode) that is, the FIFOs become 1-byte-deep holding registers 1 = transmit and receive FIFO buffers are enabled (FIFO mode).	RW	0x0
3	STP2	Two stop bits select. If this bit is set to 1, two stop bits are transmitted at the end of the frame. The receive logic does not check for two stop bits being received.	RW	0x0
2	EPS	Even parity select. Controls the type of parity the UART uses during transmission and reception: 0 = odd parity. The UART generates or checks for an odd number of 1s in the data and parity bits. 1 = even parity. The UART generates or checks for an even number of 1s in the data and parity bits. This bit has no effect when the PEN bit disables parity checking and generation.	RW	0x0
1	PEN	Parity enable: 0 = parity is disabled and no parity bit added to the data frame 1 = parity checking and generation is enabled.	RW	0x0
0	BRK	Send break. If this bit is set to 1, a low-level is continually output on the UARTRXD output, after completing transmission of the current character. For the proper execution of the break command, the software must set this bit for at least two complete frames. For normal use, this bit must be cleared to 0.	RW	0x0

UART: UARTCR Register

Offset: 0x030

Description

Control Register, UARTCR

Table 989. UARTCR Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	CTSEN	CTS hardware flow control enable. If this bit is set to 1, CTS hardware flow control is enabled. Data is only transmitted when the nUARTCTS signal is asserted.	RW	0x0
14	RTSEN	RTS hardware flow control enable. If this bit is set to 1, RTS hardware flow control is enabled. Data is only requested when there is space in the receive FIFO for it to be received.	RW	0x0
13	OUT2	This bit is the complement of the UART Out2 (nUARTOut2) modem status output. That is, when the bit is programmed to a 1, the output is 0. For DTE this can be used as Ring Indicator (RI).	RW	0x0
12	OUT1	This bit is the complement of the UART Out1 (nUARTOut1) modem status output. That is, when the bit is programmed to a 1 the output is 0. For DTE this can be used as Data Carrier Detect (DCD).	RW	0x0

Bits	Name	Description	Type	Reset
11	RTS	Request to send. This bit is the complement of the UART request to send, nUARTRTS, modem status output. That is, when the bit is programmed to a 1 then nUARTRTS is LOW.	RW	0x0
10	DTR	Data transmit ready. This bit is the complement of the UART data transmit ready, nUARTDTR, modem status output. That is, when the bit is programmed to a 1 then nUARTDTR is LOW.	RW	0x0
9	RXE	Receive enable. If this bit is set to 1, the receive section of the UART is enabled. Data reception occurs for either UART signals or SIR signals depending on the setting of the SIREN bit. When the UART is disabled in the middle of reception, it completes the current character before stopping.	RW	0x1
8	TXE	Transmit enable. If this bit is set to 1, the transmit section of the UART is enabled. Data transmission occurs for either UART signals, or SIR signals depending on the setting of the SIREN bit. When the UART is disabled in the middle of transmission, it completes the current character before stopping.	RW	0x1
7	LBE	Loopback enable. If this bit is set to 1 and the SIREN bit is set to 1 and the SIRTEST bit in the Test Control Register, UARTTCR is set to 1, then the nsIROUT path is inverted, and fed through to the SIRIN path. The SIRTEST bit in the test register must be set to 1 to override the normal half-duplex SIR operation. This must be the requirement for accessing the test registers during normal operation, and SIRTEST must be cleared to 0 when loopback testing is finished. This feature reduces the amount of external coupling required during system test. If this bit is set to 1, and the SIRTEST bit is set to 0, the UARTTXD path is fed through to the UARTRXD path. In either SIR mode or UART mode, when this bit is set, the modem outputs are also fed through to the modem inputs. This bit is cleared to 0 on reset, to disable loopback.	RW	0x0
6:3	Reserved.	-	-	-
2	SIRLP	SIR low-power IrDA mode. This bit selects the IrDA encoding mode. If this bit is cleared to 0, low-level bits are transmitted as an active high pulse with a width of 3 / 16th of the bit period. If this bit is set to 1, low-level bits are transmitted with a pulse width that is 3 times the period of the IrLPBaud16 input signal, regardless of the selected bit rate. Setting this bit uses less power, but might reduce transmission distances.	RW	0x0

Bits	Name	Description	Type	Reset
1	SIREN	SIR enable: 0 = IrDA SIR ENDEC is disabled. nSIROUT remains LOW (no light pulse generated), and signal transitions on SIRIN have no effect. 1 = IrDA SIR ENDEC is enabled. Data is transmitted and received on nSIROUT and SIRIN. UARTRXD remains HIGH, in the marking state. Signal transitions on UARTRXD or modem status inputs have no effect. This bit has no effect if the UARTEN bit disables the UART.	RW	0x0
0	UARTEN	UART enable: 0 = UART is disabled. If the UART is disabled in the middle of transmission or reception, it completes the current character before stopping. 1 = the UART is enabled. Data transmission and reception occurs for either UART signals or SIR signals depending on the setting of the SIREN bit.	RW	0x0

UART: UARTIFLS Register

Offset: 0x034

Description

Interrupt FIFO Level Select Register, UARTIFLS

Table 990. UARTIFLS Register

Bits	Name	Description	Type	Reset
31:6	Reserved.	-	-	-
5:3	RXIFLSEL	Receive interrupt FIFO level select. The trigger points for the receive interrupt are as follows: b000 = Receive FIFO becomes \geq 1 / 8 full b001 = Receive FIFO becomes \geq 1 / 4 full b010 = Receive FIFO becomes \geq 1 / 2 full b011 = Receive FIFO becomes \geq 3 / 4 full b100 = Receive FIFO becomes \geq 7 / 8 full b101-b111 = reserved.	RW	0x2
2:0	TXIFLSEL	Transmit interrupt FIFO level select. The trigger points for the transmit interrupt are as follows: b000 = Transmit FIFO becomes \leq 1 / 8 full b001 = Transmit FIFO becomes \leq 1 / 4 full b010 = Transmit FIFO becomes \leq 1 / 2 full b011 = Transmit FIFO becomes \leq 3 / 4 full b100 = Transmit FIFO becomes \leq 7 / 8 full b101-b111 = reserved.	RW	0x2

UART: UARTIMSC Register

Offset: 0x038

Description

Interrupt Mask Set/Clear Register, UARTIMSC

Table 991. UARTIMSC Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	OEIM	Overrun error interrupt mask. A read returns the current mask for the UARTOEINTR interrupt. On a write of 1, the mask of the UARTOEINTR interrupt is set. A write of 0 clears the mask.	RW	0x0

Bits	Name	Description	Type	Reset
9	BEIM	Break error interrupt mask. A read returns the current mask for the UARTBEINTR interrupt. On a write of 1, the mask of the UARTBEINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
8	PEIM	Parity error interrupt mask. A read returns the current mask for the UARTPEINTR interrupt. On a write of 1, the mask of the UARTPEINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
7	FEIM	Framing error interrupt mask. A read returns the current mask for the UARTFEINTR interrupt. On a write of 1, the mask of the UARTFEINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
6	RTIM	Receive timeout interrupt mask. A read returns the current mask for the UARTRTINTR interrupt. On a write of 1, the mask of the UARTRTINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
5	TXIM	Transmit interrupt mask. A read returns the current mask for the UARTRXINTR interrupt. On a write of 1, the mask of the UARTRXINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
4	RXIM	Receive interrupt mask. A read returns the current mask for the UARTRXINTR interrupt. On a write of 1, the mask of the UARTRXINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
3	DSRMIM	nUARTDSR modem interrupt mask. A read returns the current mask for the UARTDSRINTR interrupt. On a write of 1, the mask of the UARTDSRINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
2	DCDMIM	nUARTDCD modem interrupt mask. A read returns the current mask for the UARTDCDINTR interrupt. On a write of 1, the mask of the UARTDCDINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
1	CTSMIM	nUARTCTS modem interrupt mask. A read returns the current mask for the UARTCTSINTR interrupt. On a write of 1, the mask of the UARTCTSINTR interrupt is set. A write of 0 clears the mask.	RW	0x0
0	RIMIM	nUARTRI modem interrupt mask. A read returns the current mask for the UARTRIINTR interrupt. On a write of 1, the mask of the UARTRIINTR interrupt is set. A write of 0 clears the mask.	RW	0x0

UART: UARTRIS Register

Offset: 0x03c

Description

Raw Interrupt Status Register, UARTRIS

Table 992. UARTRIS Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	OERIS	Overrun error interrupt status. Returns the raw interrupt state of the UARTOEINTR interrupt.	RO	0x0
9	BERIS	Break error interrupt status. Returns the raw interrupt state of the UARTBEINTR interrupt.	RO	0x0
8	PERIS	Parity error interrupt status. Returns the raw interrupt state of the UARTPEINTR interrupt.	RO	0x0
7	FERIS	Framing error interrupt status. Returns the raw interrupt state of the UARTFEINTR interrupt.	RO	0x0
6	RTRIS	Receive timeout interrupt status. Returns the raw interrupt state of the UARTRTINTR interrupt.	RO	0x0
5	TXRIS	Transmit interrupt status. Returns the raw interrupt state of the UARTRXINTR interrupt.	RO	0x0
4	RXRIS	Receive interrupt status. Returns the raw interrupt state of the UARTRXINTR interrupt.	RO	0x0
3	DSRRMIS	nUARTDSR modem interrupt status. Returns the raw interrupt state of the UARTDSRINTR interrupt.	RO	-
2	DCDRMIS	nUARTDCD modem interrupt status. Returns the raw interrupt state of the UARTDCDINTR interrupt.	RO	-
1	CTSRMIS	nUARTCTS modem interrupt status. Returns the raw interrupt state of the UARTCTSINTR interrupt.	RO	-
0	RIRMIS	nUARTRI modem interrupt status. Returns the raw interrupt state of the UARTRIINTR interrupt.	RO	-

UART: UARTMIS Register

Offset: 0x040

Description

Masked Interrupt Status Register, UARTMIS

Table 993. UARTMIS Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	OEMIS	Overrun error masked interrupt status. Returns the masked interrupt state of the UARTOEINTR interrupt.	RO	0x0
9	BEMIS	Break error masked interrupt status. Returns the masked interrupt state of the UARTBEINTR interrupt.	RO	0x0
8	PEMIS	Parity error masked interrupt status. Returns the masked interrupt state of the UARTPEINTR interrupt.	RO	0x0
7	FEMIS	Framing error masked interrupt status. Returns the masked interrupt state of the UARTFEINTR interrupt.	RO	0x0
6	RTMIS	Receive timeout masked interrupt status. Returns the masked interrupt state of the UARTRTINTR interrupt.	RO	0x0
5	TXMIS	Transmit masked interrupt status. Returns the masked interrupt state of the UARTRXINTR interrupt.	RO	0x0

Bits	Name	Description	Type	Reset
4	RXMIS	Receive masked interrupt status. Returns the masked interrupt state of the UARTRXINTR interrupt.	RO	0x0
3	DSRMMIS	nUARTDSR modem masked interrupt status. Returns the masked interrupt state of the UARDSRINTR interrupt.	RO	-
2	DCDMMIS	nUARTDCD modem masked interrupt status. Returns the masked interrupt state of the UARDCDINTR interrupt.	RO	-
1	CTSMMIS	nUARTCTS modem masked interrupt status. Returns the masked interrupt state of the UARTCTSINTR interrupt.	RO	-
0	RIMMIS	nUARTRI modem masked interrupt status. Returns the masked interrupt state of the UARTRIINTR interrupt.	RO	-

UART: UARTICR Register

Offset: 0x044

Description

Interrupt Clear Register, UARTICR

Table 994. UARTICR Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	OEIC	Overrun error interrupt clear. Clears the UARTOEINTR interrupt.	WC	-
9	BEIC	Break error interrupt clear. Clears the UARTBEINTR interrupt.	WC	-
8	PEIC	Parity error interrupt clear. Clears the UARTPEINTR interrupt.	WC	-
7	FEIC	Framing error interrupt clear. Clears the UARTFEINTR interrupt.	WC	-
6	RTIC	Receive timeout interrupt clear. Clears the UARTRTINTR interrupt.	WC	-
5	TXIC	Transmit interrupt clear. Clears the UARTTXINTR interrupt.	WC	-
4	RXIC	Receive interrupt clear. Clears the UARTRXINTR interrupt.	WC	-
3	DSRMIC	nUARTDSR modem interrupt clear. Clears the UARDSRINTR interrupt.	WC	-
2	DCDMIC	nUARTDCD modem interrupt clear. Clears the UARDCDINTR interrupt.	WC	-
1	CTSMIC	nUARTCTS modem interrupt clear. Clears the UARTCTSINTR interrupt.	WC	-
0	RIMIC	nUARTRI modem interrupt clear. Clears the UARTRIINTR interrupt.	WC	-

UART: UARTRDMACR Register

Offset: 0x048

Description

DMA Control Register, UARTRDMACR

Table 995.
UARTDMACR Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	DMAONERR	DMA on error. If this bit is set to 1, the DMA receive request outputs, UARTRXDMASREQ or UARTRXDMABREQ, are disabled when the UART error interrupt is asserted.	RW	0x0
1	TXDMAE	Transmit DMA enable. If this bit is set to 1, DMA for the transmit FIFO is enabled.	RW	0x0
0	RXDMAE	Receive DMA enable. If this bit is set to 1, DMA for the receive FIFO is enabled.	RW	0x0

UART: UARTPERIPHID0 Register

Offset: 0xfe0

Description

UARTPeriphID0 Register

Table 996.
UARTPERIPHID0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PARTNUMBER0	These bits read back as 0x11	RO	0x11

UART: UARTPERIPHID1 Register

Offset: 0xfe4

Description

UARTPeriphID1 Register

Table 997.
UARTPERIPHID1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DESIGNERO	These bits read back as 0x1	RO	0x1
3:0	PARTNUMBER1	These bits read back as 0x0	RO	0x0

UART: UARTPERIPHID2 Register

Offset: 0xfe8

Description

UARTPeriphID2 Register

Table 998.
UARTPERIPHID2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	This field depends on the revision of the UART: r1p0 0x0 r1p1 0x1 r1p3 0x2 r1p4 0x2 r1p5 0x3	RO	0x3
3:0	DESIGNER1	These bits read back as 0x4	RO	0x4

UART: UARTPERIPHID3 Register

Offset: 0fec

Description

UARTPeriphID3 Register

Table 999.
UARTPERIPHID3
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	CONFIGURATION	These bits read back as 0x00	RO	0x00

UART: UARTPCELLID0 Register

Offset: 0xff0

Description

UARTPCELLID0 Register

Table 1000.
UARTPCELLID0
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	UARTPCELLID0	These bits read back as 0x0D	RO	0x0D

UART: UARTPCELLID1 Register

Offset: 0xff4

Description

UARTPCELLID1 Register

Table 1001.
UARTPCELLID1
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	UARTPCELLID1	These bits read back as 0xF0	RO	0xf0

UART: UARTPCELLID2 Register

Offset: 0xff8

Description

UARTPCELLID2 Register

Table 1002.
UARTPCELLID2
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	UARTPCELLID2	These bits read back as 0x05	RO	0x05

UART: UARTPCELLID3 Register

Offset: 0ffc

Description

UARTPCELLID3 Register

Table 1003.
UARTPCELLID3
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	UARTPCELLID3	These bits read back as 0xB1	RO	0xb1

12.2. I2C

Synopsys Documentation

Synopsys Proprietary. Used with permission.

I2C is a commonly used 2-wire interface that can be used to connect devices for low speed data transfer using clock `SCL` and data `SDA` wires.

RP2350 has two identical instances of an I2C controller. The external pins of each controller are connected to GPIO pins as defined in the GPIO muxing [table](#) in [Section 9.4](#). The muxing options give some IO flexibility.

12.2.1. Features

Each I2C controller is based on a configuration of the Synopsys [DW_apb_i2c \(v2.03a\)](#) IP. The following features are supported:

- Master or Slave (Default to Master mode)
- Standard mode, Fast mode or Fast mode plus
- Default slave address `0x055`
- Supports 10-bit addressing in Master mode
- 16-element transmit buffer
- 16-element receive buffer
- Can be driven from DMA
- Can generate interrupts

12.2.1.1. Standard

The I2C controller was designed for I2C Bus specification, version 6.0, dated April 2014.

12.2.1.2. Clocking

All clocks in the I2C controller are connected to `clk_sys`, including `ic_clk` which is mentioned in later sections. The I2C clock is generated by dividing down this clock, controlled by registers inside the block.

12.2.1.3. IOs

Each controller must connect its clock `SCL` and data `SDA` to one pair of GPIOs. The I2C standard requires that drivers drive a signal low, or when not driven the signal will be pulled high. This applies to SCL and SDA. The GPIO pads should be configured for:

- pull-up enabled
- slew rate limited
- schmitt trigger enabled

NOTE

There should also be external pull-ups on the board as the internal pad pull-ups may not be strong enough to pull up external circuits.

12.2.2. IP Configuration

I2C configuration details (each instance is fully independent):

- 32-bit APB access
- Supports Standard mode, Fast mode or Fast mode plus (not High speed)
- Default slave address of `0x055`
- Master or Slave mode
- Master by default (Slave mode disabled at reset)
- 10-bit addressing supported in master mode (7-bit by default)
- 16 entry transmit buffer
- 16 entry receive buffer
- Allows restart conditions when a master (can be disabled for legacy device support)
- Configurable timing to adjust `TsUDAT/ThDAT`
- General calls responded to on reset
- Interface to DMA
- Single interrupt output
- Configurable timing to adjust clock frequency
- Spike suppression (default 7 `clk_sys` cycles)
- Can NACK after data received by Slave
- Hold transfer when Tx FIFO empty
- Hold bus until space available in Rx FIFO
- Restart detect interrupt in Slave mode
- Optional blocking Master commands (not enabled by default)

12.2.3. I2C Overview

The I2C bus is a 2-wire serial interface, consisting of a serial data line `SDA` and a serial clock `SCL`. These wires carry information between the devices connected to the bus. Each device is recognized by a unique address and can operate as either a "transmitter" or "receiver", depending on the function of the device. Devices can also be considered as masters or slaves when performing data transfers. A master is a device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

NOTE

The I2C block must only be programmed to operate in either master OR slave mode only. Operating as a master and slave simultaneously is not supported.

The I2C block can operate in these modes:

- standard mode (with data rates from 0 to 100kb/s),
- fast mode (with data rates up to 400kb/s),
- fast mode plus (with data rates up to 1000kb/s).

These modes are not supported:

- High-speed mode (with data rates up to 3.4Mb/s),
- Ultra-Fast Speed Mode (with data rates up to 5Mb/s).

NOTE

References to fast mode also apply to fast mode plus, unless specifically stated otherwise.

The I2C block can communicate with devices in one of these modes as long as they are attached to the bus. Additionally, fast mode devices are downward compatible. For instance, fast mode devices can communicate with standard mode devices at up to 100kb/s over the I2C bus system. However, standard mode devices are not upward compatible and should not be incorporated in a fast-mode I2C bus system as they cannot follow the higher transfer rate; unpredictable states would occur.

The following devices commonly use high-speed mode:

- LCD displays
- high-bit count ADCs
- high capacity EEPROMs

These devices typically need to transfer large amounts of data.

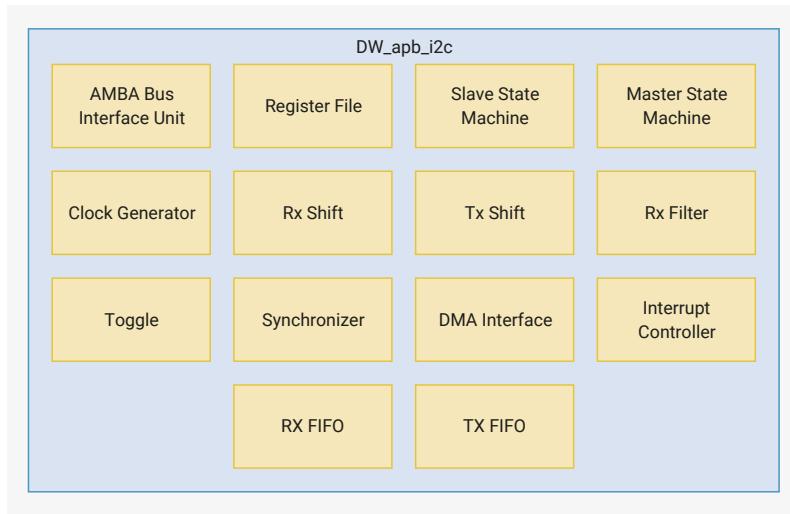
Most maintenance and control applications, the common use for the I2C bus, typically operate at 100kHz in standard and fast modes. Any [DW_apb_i2c](#) device can be attached to an I2C bus. Every device can talk with any master, passing information back and forth. There needs to be at least one master (such as a microcontroller or DSP) on the bus, but there can be multiple masters, which require them to arbitrate for ownership. Multiple masters and arbitration are explained later in this chapter. The I2C block does not support SMBus and PMBus protocols (for System management and Power management).

The [DW_apb_i2c](#) is made up of:

- an AMBA APB slave interface
- an I2C interface
- FIFO logic to maintain coherency between the two interfaces

The blocks of the component are illustrated in [Figure 46](#).

Figure 46. I2C Block diagram



The following define the functions of the blocks in Figure 46:

- **AMBA Bus Interface Unit:** Takes the APB interface signals and translates them into a common generic interface that allows the register file to be bus protocol-agnostic.
- **Register File:** Contains configuration registers and is the interface with software.
- **Slave State Machine:** Follows the protocol for a slave and monitors bus for address match.
- **Master State Machine:** Generates the I2C protocol for the master transfers.
- **Clock Generator:** Calculates the required timing to do the following:
 - Generate the **SCL** clock when configured as a master
 - Check for bus idle
 - Generate a START and a STOP
 - Setup the data and hold the data
- **Rx Shift:** Takes data into the design and extracts it in byte format.
- **Tx Shift:** Presents data supplied by CPU for transfer on the I2C bus.
- **Rx Filter:** Detects the events in the bus; for example, start, stop and arbitration lost.
- **Toggle:** Generates pulses on both sides and toggles to transfer signals across clock domains.
- **Synchronizer:** Transfers signals from one clock domain to another.
- **DMA Interface:** Generates the handshaking signals to the central DMA controller in order to automate the data transfer without CPU intervention.
- **Interrupt Controller:** Generates the raw interrupt and interrupt flags, allowing them to be set and cleared.
- **Rx FIFO/Tx FIFO:** Holds the Rx FIFO and Tx FIFO register banks and controllers, along with their status levels.

12.2.4. I2C Terminology

This section defines key terms used in various parts of the I2C.

12.2.4.1. I2C Bus Terms

The following terms relate to how the role of the I2C device and how it interacts with other I2C devices on the bus.

Transmitter

the device that sends data to the bus. A transmitter can either be a device that initiates the data transmission to the bus (a master-transmitter) or the device that responds to a request from the master to send data to the bus (a slave-transmitter).

Receiver

the device that receives data from the bus. A receiver can either be a device that receives data on its own request (a master-receiver) or a device that receives data in response to a request from the master (a slave-receiver).

Master

the component that initializes a transfer (START command), generates the clock **SCL** signal and terminates the transfer (STOP command). A master can be either a transmitter or a receiver.

Slave

the device addressed by the master. A slave can be either receiver or transmitter.

Multi-master

the ability for more than one master to co-exist on the bus at the same time without collision or data loss.

Arbitration

the predefined procedure that authorizes only one master at a time to take control of the bus. For more information about this behaviour, refer to [Section 12.2.8](#).

Synchronization

the predefined procedure that synchronizes the clock signals provided by two or more masters. For more information about this feature, refer to [Section 12.2.9](#).

SDA

the data signal line (Serial Data).

SCL

the clock signal line (Serial Clock).

12.2.4.2. Bus Transfer Terms

The following terms are specific to data transfers that occur to and from the I2C bus.

START (RESTART)

data transfer begins with a START or RESTART condition. The level of the **SDA** data line changes from high to low, while the **SCL** clock line remains high. When this occurs, the bus becomes busy.

NOTE

START and RESTART conditions are functionally identical.

STOP

data transfer is terminated by a STOP condition. This occurs when the level on the **SDA** data line passes from the low state to the high state, while the **SCL** clock line remains high. When the data transfer has been terminated, the bus is free or idle once again. The bus stays busy if a RESTART is generated instead of a STOP condition.

12.2.5. I2C Behaviour

The **DW_apb_i2c** can be controlled via software to be one of the following:

- An I2C master only, communicating with other I2C slaves

- An I2C slave only, communicating with one or more I2C masters.

The master is responsible for generating the clock and controlling the transfer of data. The slave is responsible for either transmitting or receiving data to and from the master. The acknowledgement of data is sent by the device that is receiving data, which can be either a master or a slave. As mentioned previously, the I2C protocol also allows multiple masters to reside on the I2C bus and uses an arbitration procedure to determine bus ownership.

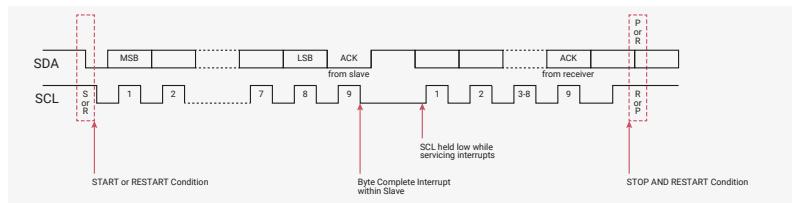
Each slave has a unique address determined by the system designer. When a master wants to communicate with a slave:

1. The master transmits a START/RESTART condition that is then followed by the slave's address and a control bit (R/W) to determine if the master wants to transmit data or receive data from the slave.
2. The slave then sends an acknowledge (ACK) pulse after the address.

When the master (master-transmitter) writes to the slave (slave-receiver), the receiver gets one byte of data. This transaction continues until the master terminates the transmission with a STOP condition.

When the master reads from a slave (master-receiver), the slave transmits (slave-transmitter) a byte of data to the master. The master then acknowledges the transaction with the ACK pulse. This transaction continues until the master terminates the transmission by not acknowledging (NACK) the transaction after the last byte is received, and then the master issues a STOP condition or addresses another slave after issuing a RESTART condition. This behaviour is illustrated in [Figure 47](#).

Figure 47. Data transfer on the I2C Bus



The [DW_apb_i2c](#) is a synchronous serial interface. The [SDA](#) line is a bidirectional signal that changes only while the [SCL](#) line is low except for STOP, START, and RESTART conditions. The output drivers are open-drain or open-collector to perform wire-AND functions on the bus. The maximum number of devices on the bus is limited by only the maximum capacitance specification of 400 pF. Data is transmitted in byte packages.

The I2C protocols implemented in [DW_apb_i2c](#) are described in more details in [Section 12.2.6](#).

12.2.5.1. START and STOP Generation

When operating as an I2C master, putting data into the Tx FIFO causes the [DW_apb_i2c](#) to generate a START condition on the I2C bus. Writing a 1 to [IC_DATA_CMD](#).STOP causes the [DW_apb_i2c](#) to generate a STOP condition on the I2C bus; a STOP condition is not issued if this bit is not set, even if the Tx FIFO is empty.

When operating as a slave, the [DW_apb_i2c](#) does not generate START and STOP conditions, as per the protocol. However, if a read request is made to the [DW_apb_i2c](#), it holds the [SCL](#) line low until read data has been supplied to it. This stalls the I2C bus until read data is provided to the slave [DW_apb_i2c](#), or the [DW_apb_i2c](#) slave is disabled by writing a 0 to [IC_ENABLE](#).ENABLE.

12.2.5.2. Combined Formats

The [DW_apb_i2c](#) supports mixed read and write combined format transactions in both 7-bit and 10-bit addressing modes. The [DW_apb_i2c](#) does not support mixed address and mixed address format - that is, a 7-bit address transaction followed by a 10-bit address transaction or vice versa-combined format transactions. To initiate combined format transfers, [IC_CON](#).IC_RESTART_EN should be set to 1. With this value set and operating as a master, when the [DW_apb_i2c](#) completes an I2C transfer, it checks the Tx FIFO and executes the next transfer. If the direction of this transfer differs from the previous transfer, the combined format is used to issue the transfer. If the Tx FIFO is empty when the current I2C transfer completes:

- [IC_DATA_CMD](#).STOP is checked and:
 - If set to 1, a STOP bit is issued.
 - If set to 0, the [SCL](#) is held low until the next command is written to the Tx FIFO.

For more details, refer to [Section 12.2.7](#).

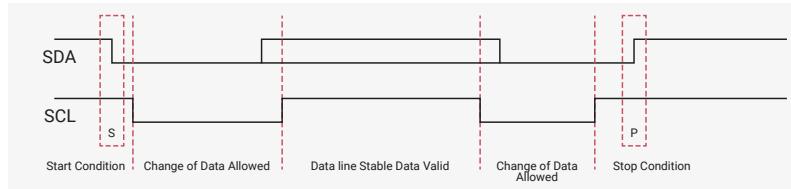
12.2.6. I2C Protocols

This section defines protocols used in the [DW_apb_i2c](#).

12.2.6.1. START and STOP Conditions

When the bus is idle, both the [SCL](#) and [SDA](#) signals are pulled high through external pull-up resistors on the bus. When the master wants to start a transmission on the bus, the master issues a **START condition**: a high-to-low transition of the [SDA](#) signal while [SCL](#) is 1. When the master wants to terminate the transmission, the master issues a **STOP condition**: a low-to-high transition of the [SDA](#) signal while [SCL](#) is 1. [Figure 48](#) shows the timing of the START and STOP conditions. When data is being transmitted on the bus, the [SDA](#) signal must be stable when [SCL](#) is set to 1.

Figure 48. I2C START and STOP Condition



NOTE

The signal transitions for the START/STOP conditions, as depicted in [Figure 48](#), reflect those observed at the output signals of the master driving the I2C bus. Care should be taken when observing the [SDA/SCL](#) signals at the input signals of slaves, because unequal line delays may result in an incorrect [SDA/SCL](#) timing relationship.

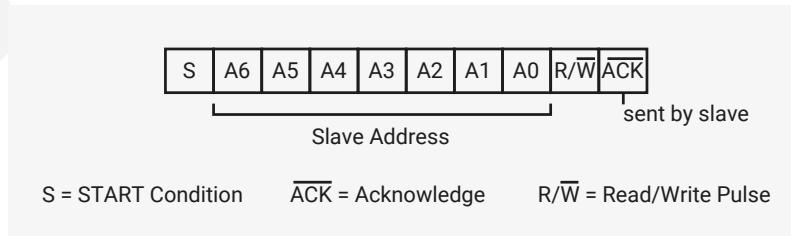
12.2.6.2. Addressing Slave Protocol

There are two address formats: 7-bit and 10-bit.

12.2.6.2.1. 7-bit Address Format

In the 7-bit address format, the first seven bits (bits 7:1) of the first byte set the slave address and the LSB bit (bit 0) defines the R/W status, as shown in [Figure 49](#). When bit 0 is set to 0, the master writes to the slave. When bit 0 is set to 1, the master reads from the slave.

Figure 49. I2C 7-bit Address Format



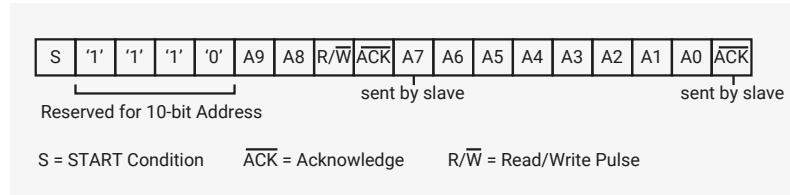
12.2.6.2.2. 10-bit Address Format

The 10-bit address format transfers two bytes for each 10-bit address.

- In the first byte, the first five bits (bits 7:3) indicate a 10-bit transfer. The next two bits (bits 2:1) contain bits 9:8 of the slave address. The LSB bit (bit 0) defines the R/W status.
- The second byte contains bits 7:0 of the slave address.

Figure 50 shows the 10-bit address format:

Figure 50. 10-bit Address Format



This table defines the special purpose and reserved first byte addresses.

Table 1004.
I2C/SMBus Definition
of Bits in First Byte

Slave Address	R/W Bit	Description
0000 000	0	General Call Address. DW_apb_i2c places the data in the receive buffer and issues a General Call interrupt.
0000 000	1	START byte. For more details, refer to Section 12.2.6.4 .
0000 001	X	CBUS address. DW_apb_i2c ignores these accesses.
0000 010	X	Reserved.
0000 011	X	Reserved.
0000 1XX	X	High-speed master code (for more information, refer to Section 12.2.8).
1111 1XX	X	Reserved.
1111 0XX	X	10-bit slave addressing.
0001 000	X	SMBus Host. (not supported)
0001 100	X	SMBus Alert Response Address. (not supported)
1100 001	X	SMBus Device Default Address. (not supported)

[DW_apb_i2c](#) does not restrict you from using reserved addresses. However, if you use these reserved addresses, you may experience incompatibilities with I2C components.

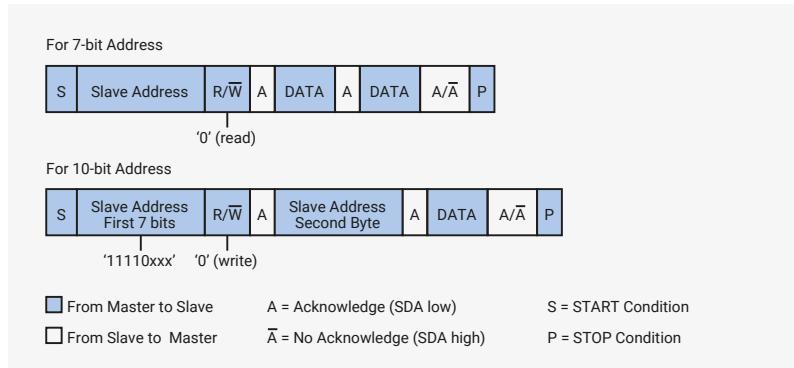
12.2.6.3. Transmitting and Receiving Protocol

The master can initiate data transmission and reception to and from the bus, acting as either a master-transmitter or master-receiver. A slave responds to requests from the master to either transmit data or receive data to/from the bus, acting as either a slave-transmitter or slave-receiver, respectively.

12.2.6.3.1. Master-Transmitter and Slave-Receiver

All data is transmitted in byte format, with no limit on the number of bytes transferred per data transfer. After the master sends the address and R/W bit or the master transmits a byte of data to the slave, the slave-receiver must respond with the acknowledge signal (ACK). When no slave-receiver responds with an ACK pulse, the master aborts the transfer by issuing a STOP condition. The slave must leave the SDA line high so that the master can abort the transfer. If the master-transmitter is transmitting data as shown in [Figure 51](#), the slave-receiver responds to the master-transmitter with an acknowledge pulse after every byte of data is received.

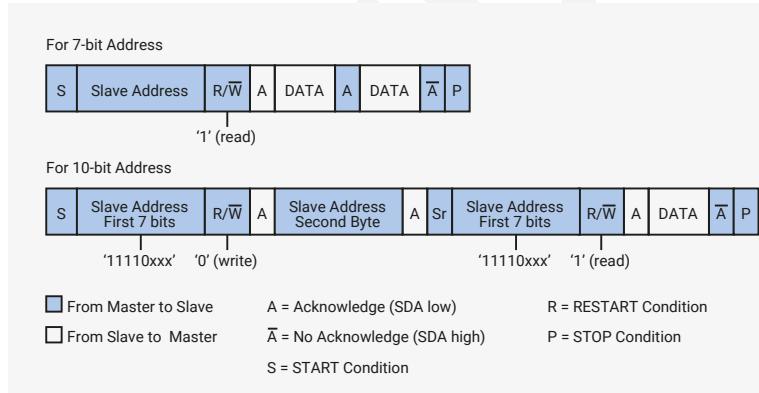
Figure 51. I2C Master-Transmitter Protocol



12.2.6.3.2. Master-Receiver and Slave-Transmitter

If the master is receiving data as shown in [Figure 52](#) the master responds to the slave-transmitter with an acknowledge pulse after receiving each byte of data, except for the last byte. This is the way the master-receiver notifies the slave-transmitter that this is the last byte. The slave-transmitter relinquishes the SDA line after detecting No Acknowledge (NACK) so that the master can issue a STOP condition.

Figure 52. I2C Master-Receiver Protocol



When a master does not want to relinquish the bus with a STOP condition, the master can issue a RESTART condition. This is identical to a START condition except it occurs after the ACK pulse. Operating in master mode, the [DW_apb_i2c](#) can then communicate with the same slave using a transfer of a different direction. For a description of the combined format transactions that the [DW_apb_i2c](#) supports, see [Section 12.2.5.2](#).

i NOTE

The [DW_apb_i2c](#) must be completely disabled before the target slave address register ([IC_TAR](#)) can be reprogrammed.

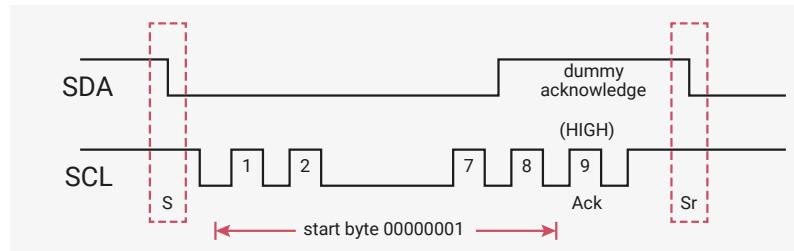
12.2.6.4. START BYTE Transfer Protocol

The START BYTE transfer protocol is designed for systems that do not have an on-board dedicated I2C hardware module. When the [DW_apb_i2c](#) is addressed as a slave, it always samples the I2C bus at the highest speed supported so that it never requires a START BYTE transfer. However, when [DW_apb_i2c](#) is a master, it supports the generation of START

BYTE transfers at the beginning of every transfer in case a slave device requires it.

This protocol consists of the transmission of seven zeros, followed by a one, as illustrated in [Figure 53](#). This allows the processor polling the bus to under-sample the address phase until zero is detected. Once the microcontroller detects a zero, it switches from the under sampling rate to the correct rate of the master.

Figure 53. I2C Start Byte Transfer



The START BYTE procedure is as follows:

1. Master generates a START condition.
2. Master transmits the START byte (0000 0001).
3. Master transmits the ACK clock pulse. (Present only to conform with the byte handling format used on the bus)
4. No slave sets the ACK signal to zero.
5. Master generates a RESTART (R) condition.

Hardware receivers do not respond to the START BYTE procedure because it uses a reserved address and resets after the RESTART condition generates.

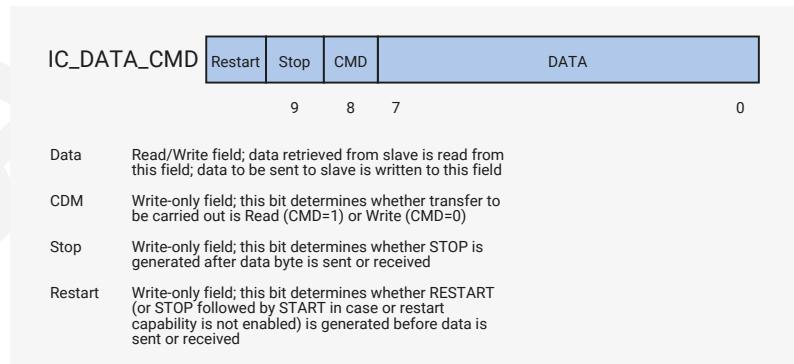
12.2.7. Tx FIFO Management and START, STOP and RESTART Generation

When operating as a master, the `DW_apb_i2c` component supports the mode of Tx (transmit) FIFO management illustrated in [Figure 54](#).

12.2.7.1. Tx FIFO Management

The component does not generate a STOP if the Tx FIFO becomes empty; in this situation the component holds the `SCL` line low, stalling the bus until a new entry is available in the Tx FIFO. A STOP condition is generated only when the user specifically requests it by setting bit nine (Stop bit) of the command written to `IC_DATA_CMD` register. [Figure 54](#) shows the bits in the `IC_DATA_CMD` register.

Figure 54.
`IC_DATA_CMD`
Register



[Figure 55](#) illustrates the behaviour of the `DW_apb_i2c` when the Tx FIFO becomes empty while operating as a master transmitter, as well as the generation of a STOP condition.

Figure 55. Master Transmitter - Tx FIFO Empties/STOP Generation

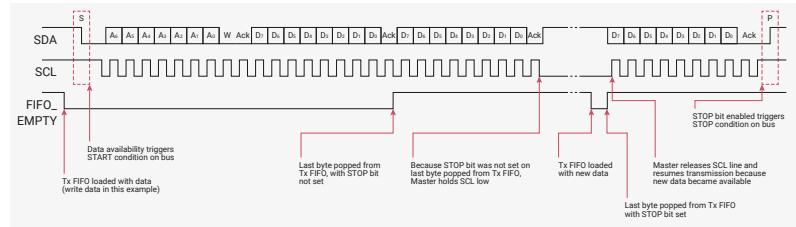


Figure 56 illustrates the behaviour of the `DW_apb_i2c` when the Tx FIFO becomes empty while operating as a master receiver, as well as the generation of a STOP condition.

Figure 56. Master Receiver - Tx FIFO Empties/STOP Generation

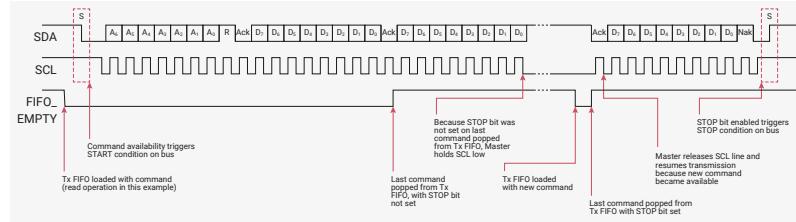


Figure 57 and Figure 58 illustrate configurations where the user can control the generation of RESTART conditions on the I2C bus. If bit 10 (Restart) of the `IC_DATA_CMD` register is set and the restart capability is enabled (`IC_RESTART_EN=1`), a RESTART is generated before the data byte is written to or read from the slave. If the restart capability is not enabled, a STOP followed by a START is generated in place of the RESTART. Figure 57 illustrates this situation during operation as a master transmitter.

Figure 57. Master Transmitter - Restart Bit of `IC_DATA_CMD` Is Set

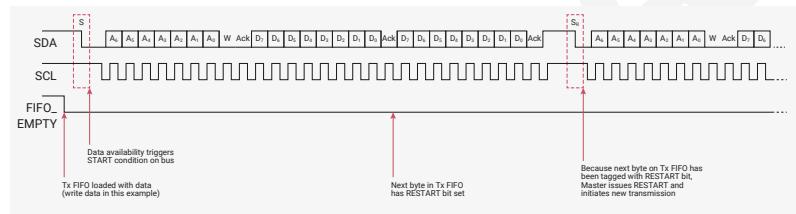


Figure 58 illustrates the same situation, but during operation as a master receiver.

Figure 58. Master Receiver - Restart Bit of `IC_DATA_CMD` Is Set

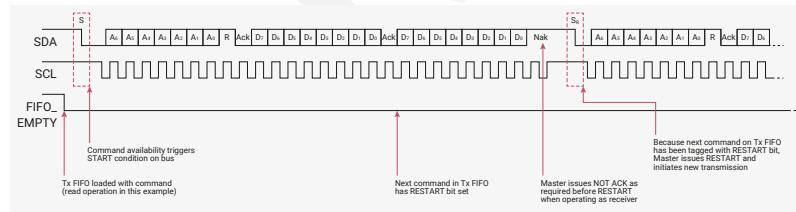


Figure 59 illustrates operation as a master transmitter where the Stop bit of the `IC_DATA_CMD` register is set and the Tx FIFO is not empty.

Figure 59. Master Transmitter - Stop Bit of `IC_DATA_CMD` Set/Tx FIFO not empty

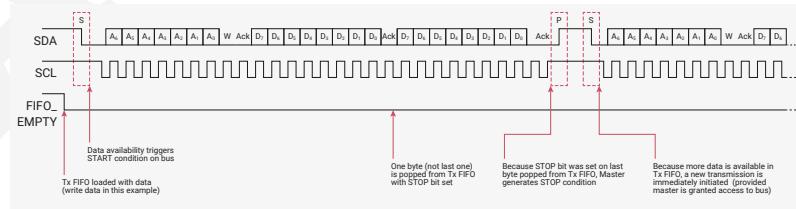


Figure 60 illustrates operation as a master transmitter where the first byte loaded into the Tx FIFO is allowed to go empty with the Restart bit set.

Figure 60. Master Transmitter - First Byte Loaded Into Tx FIFO Allowed to Empty, Restart Bit Set

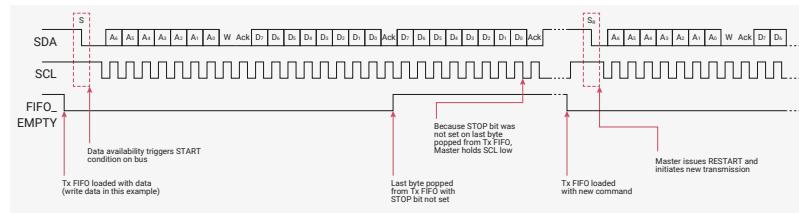


Figure 61 illustrates operation as a master receiver where the Stop bit of the IC_DATA_CMD register is set and the Tx FIFO is not empty.

Figure 61. Master Receiver - Stop Bit of IC_DATA_CMD Set/Tx FIFO Not Empty

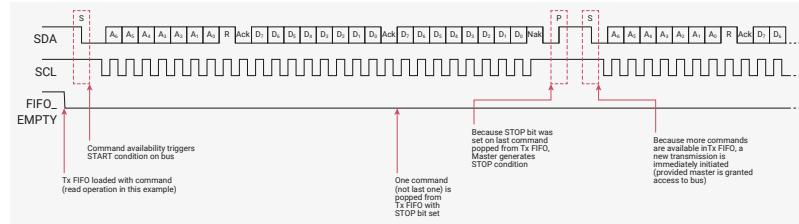
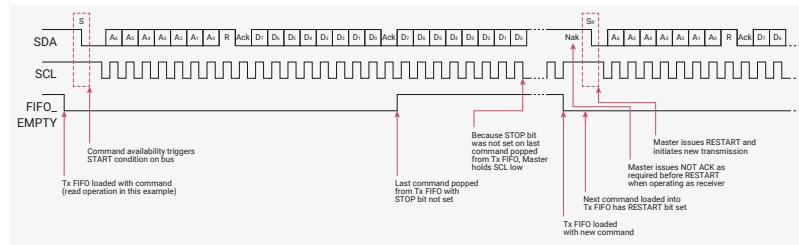


Figure 62 illustrates operation as a master receiver where the first command loaded after the Tx FIFO is allowed to empty and the Restart bit is set.

Figure 62. Master Receiver - First Command Loaded After Tx FIFO Allowed to Empty/Restart Bit Set



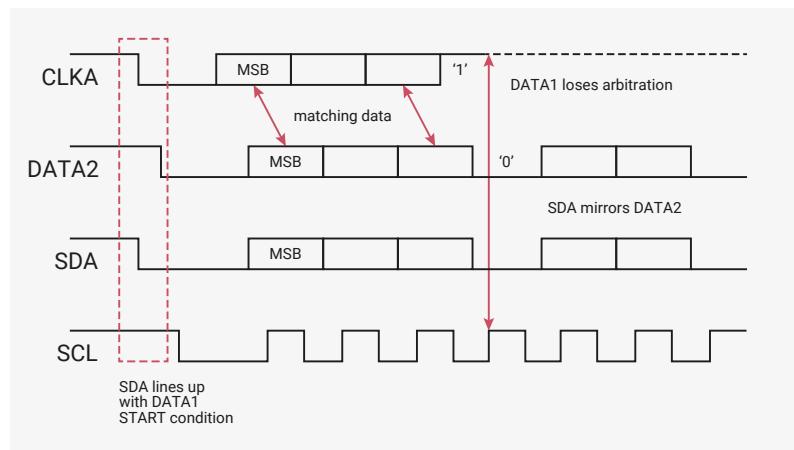
12.2.8. Multiple Master Arbitration

The **DW_apb_i2c** bus protocol allows multiple masters to reside on the same bus. If there are two masters on the same I2C bus, there is an arbitration procedure if both try to take control of the bus at the same time by generating a START condition at the same time. Once a master (for example, a microcontroller) has control of the bus, no other master can take control until the first master sends a STOP condition and places the bus in an idle state.

Arbitration takes place on the **SDA** line, while the **SCL** line is set to 1. The master, which transmits a one while the other master transmits zero, loses arbitration and turns off its data output stage. The master that lost arbitration can continue to generate clocks until the end of the byte transfer. If both masters address the same slave device, the arbitration could go into the data phase.

Upon detecting that it has lost arbitration to another master, the `DW_apb_i2c` stops generating `SCL` by disabling the output driver. [Figure 63](#) illustrates the timing of two masters arbitrating on the bus.

Figure 63. Multiple Master Arbitration



Control of the bus is determined by address or master code and data sent by competing masters, so there is no central master nor any order of priority on the bus.

Arbitration is not allowed between the following conditions:

- A RESTART condition and a data bit
- A STOP condition and a data bit
- A RESTART condition and a STOP condition

NOTE

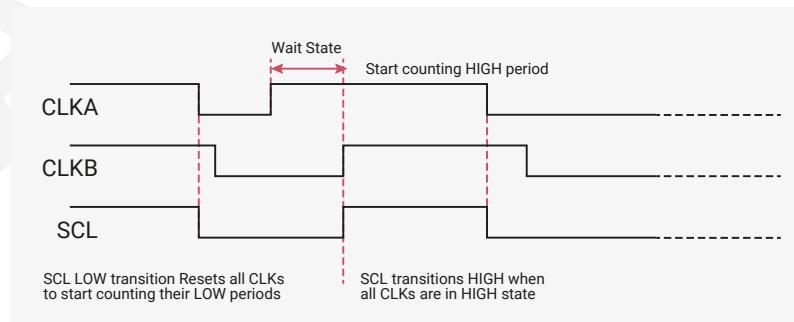
Slaves do not participate in the arbitration process.

12.2.9. Clock Synchronization

When two or more masters try to transfer information on the bus at the same time, they must arbitrate and synchronize the **SCL** clock. All masters generate their own clock to transfer messages. Data is valid only during the high period of **SCL** clock. Clock synchronization is performed using the wired-AND connection to the **SCL** signal. When the master transitions the **SCL** clock to zero, the master starts counting the low time of the **SCL** clock and transitions the **SCL** clock signal to one at the beginning of the next clock period. However, if another master is holding the **SCL** line to 0, then the master goes into a HIGH wait state until the **SCL** clock line transitions to one.

All masters then count off their high time, and the master with the shortest high time transitions the **SCL** line to zero. The masters then count out their low time and the one with the longest low time forces the other masters into a HIGH wait state. Therefore, a synchronized **SCL** clock is generated, which is illustrated in Figure 64. Optionally, slaves may hold the **SCL** line low to slow down the timing on the I2C bus.

Figure 64. Multi-Master Clock Synchronization



12.2.10. Operation Modes

This section provides information about operation modes.

NOTE

Only set the `DW_apb_i2c` to operate as an I2C Master or an I2C Slave. Never set the `DW_apb_i2c` to operate as both simultaneously. To avoid this, never simultaneously set `IC_CON.IC_SLAVE_DISABLE` and `IC_CON.MASTER_MODE` to zero and one, respectively.

12.2.10.1. Slave Mode Operation

This section discusses slave mode procedures.

12.2.10.1.1. Initial Configuration

To use the `DW_apb_i2c` as a slave, perform the following steps:

1. Disable the `DW_apb_i2c` by writing a 0 to `IC_ENABLE.ENABLE`.
2. Write to the `IC_SAR` register (bits 9:0) to set the slave address. This is the address to which the `DW_apb_i2c` responds.
3. Write to the `IC_CON` register to specify which type of addressing is supported (7-bit or 10-bit by setting bit 3). Enable the `DW_apb_i2c` in slave-only mode by writing a 0 into bit six (`IC_CON.IC_SLAVE_DISABLE`) and a 0 to bit zero (`IC_CON.MASTER_MODE`).

NOTE

Slaves and masters can use different addressing settings. For instance, a slave can be programmed with 7-bit addressing and a master with 10-bit addressing, and vice versa.

4. Enable the `DW_apb_i2c` by writing a 1 to `IC_ENABLE.ENABLE`.

NOTE

Depending on the reset values chosen, steps two and three may not be necessary because the reset values can be configured. For instance, if the device is only going to be a master, there would be no need to set the slave address because you can configure `DW_apb_i2c` to have the slave disabled after reset and to enable the master after reset. The values stored are static and do not need to be reprogrammed if the `DW_apb_i2c` is disabled.

WARNING

Only bring the `DW_apb_i2c` Slave out of reset when the I2C bus is IDLE. De-asserting the reset when a transfer is ongoing on the bus causes internal synchronization flip-flops used to synchronize `SDA` and `SCL` to toggle from a reset value of one to the actual value on the bus. This can result in `SDA` toggling from one to zero while `SCL` is one, thereby causing a false START condition to be detected by the `DW_apb_i2c` Slave. This scenario can also be avoided by configuring the `DW_apb_i2c` with `IC_SLAVE_DISABLE = 1` and `MASTER_MODE = 1` so that the Slave interface is disabled after reset. It can then be enabled by programming `IC_CON[0] = 0` and `IC_CON[6] = 0` after the internal `SDA` and `SCL` have synchronized to the value on the bus; this takes approximately six `ic_clk` cycles after reset de-assertion.

12.2.10.1.2. Slave-Transmitter Operation for a Single Byte

When another I2C master device on the bus addresses the `DW_apb_i2c` and requests data, the `DW_apb_i2c` acts as a slave-transmitter. The following steps occur:

1. The other I2C master device initiates an I2C transfer with an address that matches the slave address in the [IC_SAR](#) register of the [DW_apb_i2c](#).
2. The [DW_apb_i2c](#) acknowledges the sent address and recognizes the direction of the transfer to indicate that it is acting as a slave-transmitter.
3. The [DW_apb_i2c](#) asserts the [RD_REQ](#) interrupt (bit five of the [IC_RAW_INTR_STAT](#) register) and holds the [SCL](#) line low. It remains in a wait state until software responds. If the [RD_REQ](#) interrupt has been masked, due to [IC_INTR_MASK.M_RD_REQ](#) being set to zero, use a hardware and/or software timing routine to instruct the CPU to perform periodic reads of the [IC_RAW_INTR_STAT](#) register.
 - o Reads that indicate [IC_RAW_INTR_STAT.RD_REQ](#) being set to one must be treated as the equivalent of the [RD_REQ](#) interrupt being asserted.
 - o Software must then act to satisfy the I2C transfer.
 - o The timing interval used should be in the order of 10 times the fastest [SCL](#) clock period the [DW_apb_i2c](#) can handle. For example, for 400kb/s, the timing interval is 25 μ s.

 NOTE

The value of 10 is recommended here because this is approximately the amount of time required for a single byte of data transferred on the I2C bus.

4. If there is any data remaining in the Tx FIFO before receiving the read request, the [DW_apb_i2c](#) asserts a [TX_ABRT](#) interrupt (bit six of the [IC_RAW_INTR_STAT](#) register) to flush the old data from the Tx FIFO. If the [TX_ABRT](#) interrupt has been masked, due to [IC_INTR_MASK.M_TX_ABRT](#) being set to zero, re-use the timing routine described in the previous step to read the [IC_RAW_INTR_STAT](#) register.

 NOTE

Because the [DW_apb_i2c](#)'s Tx FIFO is forced into a flushed/reset state whenever a [TX_ABRT](#) event occurs, software must release the [DW_apb_i2c](#) from this state by reading the [IC_CLR_TX_ABRT](#) register before attempting to write into the Tx FIFO. See register [IC_RAW_INTR_STAT](#) for more details.

- o Reads that indicate bit six ([R_TX_ABRT](#)) being set to one must be treated as the equivalent of the [TX_ABRT](#) interrupt being asserted.
 - o There is no further action required from software.
 - o The timing interval used should be similar to that described in the previous step for the [IC_RAW_INTR_STAT.RD_REQ](#) register.
5. Software writes to the [IC_DATA_CMD](#) register with the data to be written (by writing a 0 in bit 8).
 6. Software must clear the [RD_REQ](#) and [TX_ABRT](#) interrupts (bits five and six, respectively) of the [IC_RAW_INTR_STAT](#) register before proceeding. If the [RD_REQ](#) or [TX_ABRT](#) interrupts have been masked, then clearing of the [IC_RAW_INTR_STAT](#) register will have already been performed when either the [R_RD_REQ](#) or [R_TX_ABRT](#) bit has been read as one.
 7. The [DW_apb_i2c](#) releases the [SCL](#) and transmits the byte.
 8. The master may hold the I2C bus by issuing a RESTART condition or release the bus by issuing a STOP condition.

NOTE

Slave-Transmitter Operation for a single byte is not applicable in Ultra-Fast mode, since this mode does not support read transfers.

12.2.10.1.3. Slave-Receiver Operation for a Single Byte

When another I2C master device on the bus addresses the `DW_apb_i2c` and is sending data, the `DW_apb_i2c` acts as a slave-receiver and the following steps occur:

1. The other I2C master device initiates an I2C transfer with an address that matches the `DW_apb_i2c`'s slave address in the `IC_SAR` register.
2. The `DW_apb_i2c` acknowledges the sent address and recognizes the direction of the transfer to indicate that the `DW_apb_i2c` is acting as a slave-receiver.
3. `DW_apb_i2c` receives the transmitted byte and places it in the receive buffer.

NOTE

If the Rx (receive) FIFO is completely filled with data when a byte is pushed, then the `DW_apb_i2c` slave holds the I2C `SCL` line low until the Rx FIFO has some space, and then continues with the next read request.

4. `DW_apb_i2c` asserts the `RX_FULL` interrupt `IC_RAW_INTR_STAT.RX_FULL`. If the `RX_FULL` interrupt has been masked, due to setting `IC_INTR_MASK.M_RX_FULL` to zero or setting `IC_TX_TL` to a value larger than zero, you should implement a timing routine (described in [Section 12.2.10.1.2](#)) for periodic reads of the `IC_STATUS` register. This timing routine should treat reads of the `IC_STATUS` register, with bit 3 (`RFNE`) set at one as the equivalent of an `RX_FULL` interrupt.
5. Software may read the byte from the `IC_DATA_CMD` register (bits 7:0).
6. The other master device may hold the I2C bus by issuing a RESTART condition, or release the bus by issuing a STOP condition.

12.2.10.1.4. Slave-Transfer Operation For Bulk Transfers

In the standard I2C protocol, all transactions are single byte transactions; the programmer responds to a remote master read request by writing one byte into the slave's Tx FIFO. When a slave (slave-transmitter) receives a read request (`RD_REQ`) from the remote master (master-receiver), at a minimum there should be at least one entry placed into the slave-transmitter's Tx FIFO.

`DW_apb_i2c` handles more data in the Tx FIFO. This enables subsequent read requests to take data without raising an interrupt. This eliminates latencies incurred between interrupts. This mode only occurs when `DW_apb_i2c` acts as a slave-transmitter. If the remote master acknowledges the data sent by the slave-transmitter and there is no data in the slave's Tx FIFO, the `DW_apb_i2c` holds the I2C `SCL` line low while it raises the read request interrupt (`RD_REQ`) and waits for a data write into the Tx FIFO.

If the `RD_REQ` interrupt is masked by setting `IC_INTR_STAT.R_RD_REQ` to zero, use a timing routine to activate periodic reads of the `IC_RAW_INTR_STAT` register. Reads of `IC_RAW_INTR_STAT` that return bit five (`RD_REQ`) set to one must be treated as the equivalent of `RD_REQ`. This timing routine is similar to that described in [Section 12.2.10.1.2](#).

The `RD_REQ` interrupt is raised upon a read request. Always clear this interrupt when exiting the interrupt service handling routine (ISR). The ISR allows you to either write one byte or more than one byte into the Tx FIFO. The master can request additional data at the end of a transmission by acknowledging the last byte. In this scenario, the slave must raise `RD_REQ` again.

If you know in advance that the remote master requests a packet of `n` bytes, you can write `n` byte to the Tx FIFO. Then, when another master addresses `DW_apb_i2c` and requests data, the remote master will receive a continuous stream of data. This happens because the `DW_apb_i2c` slave continues to send data to the remote master as long as the remote

master acknowledges the data sent and there is data available in the Tx FIFO. There is no need to hold the **SCL** line low or to issue **RD_REQ** again.

If the remote master doesn't read all of the bytes from the Tx FIFO, the **DW_apb_i2c** ignores the excess bytes with the following procedure:

- The **DW_apb_i2c** clears the Tx FIFO.
- The **DW_apb_i2c** generates a transmit abort (**TX_ABRT**) event.

At the time an ACK/NACK is expected, if a NACK is received, then the remote master has all the data it wants. At this time, a flag is raised within the slave's state machine to clear the leftover data in the Tx FIFO. This flag is transferred to the processor bus clock domain where the FIFO exists and the contents of the Tx FIFO is cleared at that time.

12.2.10.2. Master Mode Operation

This section discusses master mode procedures.

12.2.10.2.1. Initial Configuration

To use the **DW_apb_i2c** as a master, perform the following steps:

1. Disable the **DW_apb_i2c** by writing zero to **IC_ENABLE.ENABLE**.
 2. Write to the **IC_CON** register to set the maximum speed mode supported (bits 2:1) and the desired speed of the **DW_apb_i2c** master-initiated transfers, either 7-bit or 10-bit addressing (bit 4). Ensure that bit six (**IC_SLAVE_DISABLE**) is written with a 1 and bit zero (**MASTER_MODE**) is written with a 1.
- NOTE**
- Slaves and masters can use different addressing settings. For instance, a slave can be programmed with 7-bit addressing and a master with 10-bit addressing, and vice versa.
3. Write the address of the I2C device to be addressed to bits 9:0 of the **IC_TAR** register. This register also determines whether the I2C will perform a General Call or a START BYTE command.
 4. Enable the **DW_apb_i2c** by writing a one to **IC_ENABLE.ENABLE**.
 5. Write the transfer direction and the data to be sent to the **IC_DATA_CMD** register. This step generates the START condition and the address byte on the **DW_apb_i2c**. Once **DW_apb_i2c** is enabled and there is data in the Tx FIFO, **DW_apb_i2c** starts reading the data.

NOTE

If you write to the **IC_DATA_CMD** register before enabling the **DW_apb_i2c**, the data and commands are lost: the buffers are kept cleared when **DW_apb_i2c** is disabled.

The values stored are static and do not need to be reprogrammed when the **DW_apb_i2c** is disabled except for transfer direction and data. As a result, you may not need to perform steps two, three, four, and five if you already configured the reset values.

12.2.10.2.2. Master Transmit and Master Receive

The **DW_apb_i2c** supports switching back and forth between reading and writing dynamically. To transmit data, write data to the lower byte of the I2C Rx/Tx Data Buffer and Command Register (**IC_DATA_CMD**). For I2C write operations, write zero to the CMD bit [8]. Subsequently, to issue a read command, write a one to the CMD bit and write **don't care** to the lower byte of the **IC_DATA_CMD** register. The **DW_apb_i2c** master continues to initiate transfers as long as there are commands present in the Tx FIFO. If the Tx FIFO becomes empty, the master performs one of the following actions

based on the value of `IC_DATA_CMD`:

- If set to one, it issues a STOP condition after completing the current transfer.
- If set to zero, it holds `SCL` low until next command is written to the Tx FIFO.

For more details, refer to [Section 12.2.7](#).

12.2.10.3. Disabling `DW_apb_i2c`

The `IC_ENABLE_STATUS` register allows software to unambiguously determine when the I2C hardware has completely shut down.

i NOTE

Earlier versions of `DW_apb_i2c` required the programmer to monitor two registers: (`IC_STATUS` and `IC_RAW_INTR_STAT`). RP2350 only requires the programmer to monitor `IC_ENABLE_STATUS`.

To shut down I2C hardware, write a zero to `IC_ENABLE`.`ENABLE`. The `DW_apb_i2c` master can be disabled only if the command currently processing when the de-assertion occurs has the STOP bit set to one. If you attempt to disable the `DW_apb_i2c` master while processing a command without the STOP bit set, the `DW_apb_i2c` master continues to remain active, holding the `SCL` line low until a new command is received in the Tx FIFO.

To relinquish the I2C bus and disable `DW_apb_i2c` while the `DW_apb_i2c` master is processing a command *without* the STOP bit set, issue an `ABORT` request.

12.2.10.3.1. Procedure

1. Define a timer interval ($t_{i2c,poll}$) equal to the 10 times the signalling period for the highest I2C transfer speed used in the system and supported by `DW_apb_i2c`. For example, if the highest I2C transfer mode is 400kb/s, $t_{i2c,poll}$ is 25 μ s.
2. Define a maximum time-out parameter, `MAX_T_POLL_COUNT`, such that if any repeated polling operation exceeds this maximum value, an error is reported.
3. Execute a blocking thread, process, or function that prevents any further I2C master transactions from starting from software, but allows any pending transfers to be completed.

i NOTE

This step can be ignored if `DW_apb_i2c` is programmed to operate as an I2C slave only.

1. The variable `POLL_COUNT` is initialized to zero.
2. Set bit zero of the `IC_ENABLE` register to zero.
3. Read the `IC_ENABLE_STATUS` register and test the `IC_EN` bit (bit 0). Increment `POLL_COUNT` by one. If `POLL_COUNT >= MAX_T_POLL_COUNT`, exit with the relevant error code.
4. If `IC_ENABLE_STATUS[0]` is one, sleep for $t_{i2c,poll}$ and proceed to the previous step. Otherwise, exit with a relevant success code.

12.2.10.4. Aborting I2C Transfers

The `ABORT` control bit of the `IC_ENABLE` register allows the software to relinquish the I2C bus before completing the issued transfer commands from the Tx FIFO. In response to an `ABORT` request, the controller issues the STOP condition over the I2C bus, followed by a Tx FIFO flush. Aborting the transfer is allowed only in master mode of operation.

12.2.10.4.1. Procedure

1. Stop filling the Tx FIFO ([IC_DATA_CMD](#)) with new commands.
2. When operating in DMA mode, disable the transmit DMA by setting [TDMAE](#) to zero.
3. Set [IC_ENABLE.ABORT](#) to one.
4. Wait for the [M_TX_ABRT](#) interrupt.
5. Read the [IC_TX_ABRT_SOURCE](#) register to identify the source as [ABRT_USER_ABRT](#).

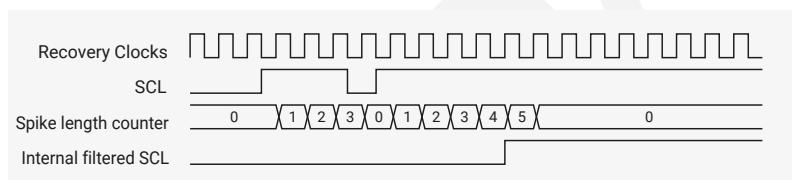
12.2.11. Spike Suppression

The [DW_apb_i2c](#) contains programmable spike suppression logic that matches requirements imposed by the I2C Bus Specification for SS/FS modes. This logic is based on counters that monitor the input signals ([SCL](#) and [SDA](#)), checking if they remain stable for a predetermined amount of [ic_clk](#) cycles before they are sampled internally. There is one separate counter for each signal ([SCL](#) and [SDA](#)). The number of [ic_clk](#) cycles can be programmed by the user. The value should account for the frequency of [ic_clk](#) and the relevant spike length specification. Each counter starts whenever its input signal changes value. Depending on the behaviour of the input signal, one of the following scenarios occurs:

- The input signal remains unchanged until the counter reaches its count limit value. When this happens, the counter resets and stops, and the internal version of the signal updates to the input value.
- The input signal changes again before the counter reaches its count limit value. When this happens, the counter resets and stops, but the internal version of the signal does not update.

The timing diagram in [Figure 65](#) illustrates the behaviour described above.

Figure 65. Spike Suppression Example



NOTE

There is a 2-stage synchronizer on the [SCL](#) input. For the sake of simplicity, this synchronization delay was not included in the timing diagram in [Figure 65](#).

The I2C Bus Specification calls for different maximum spike lengths according to the operating mode (50ns for SS and FS). Register [IC_FS_SPKLEN](#) holds the maximum spike length for SS and FS modes.

This register is 8 bits wide and accessible through the APB interface for reads and writes. However, you can only write to this register when the [DW_apb_i2c](#) is disabled. The minimum value that can be programmed into these registers is one; attempting to program a value smaller than one results in the value one being written.

The default value for these registers is based on the value of 100ns for [ic_clk](#) period, so should be updated for the [clk_sys](#) period in use on RP2350.

NOTE

- Because the minimum value that can be programmed into the `IC_FS_SPKLEN` register is one, the spike length specification can be exceeded for low frequencies of `ic_clk`. Consider the simple example of a 10MHz (100ns period) `ic_clk`; in this case, the minimum spike length that can be programmed is 100ns, which means that spikes up to this length are suppressed.
- Standard synchronization logic (two flip-flops in series) is implemented upstream of the spike suppression logic and is not affected in any way by the contents of the spike length registers or the operation of the spike suppression logic; the two operations (synchronization and spike suppression) are completely independent. Because the `SCL` and `SDA` inputs are asynchronous to `ic_clk`, there is one `ic_clk` cycle uncertainty in the sampling of these signals. Depending on when they occur relative to the rising edge of `ic_clk`, spikes of the same original length might show a difference of one `ic_clk` cycle after being sampled.
- Spike suppression is symmetrical; the behaviour is exactly the same for transitions from zero to one and from one to zero.

12.2.12. Fast Mode Plus Operation

In fast mode plus, the `DW_apb_i2c` extends fast mode operation to be support speeds up to 1000kb/s. To enable the `DW_apb_i2c` for fast mode plus operation, perform the following steps before initiating any data transfer:

1. Set `ic_clk` frequency greater than or equal to 32MHz (refer to [Section 12.2.14.2.1](#)).
2. Program the `IC_CON` register `[2:1] = 2'b10` for fast mode or fast mode plus.
3. Program `IC_FS_SCL_LCNT` and `IC_FS_SCL_HCNT` registers to meet the fast mode plus `SCL` (refer to [Section 12.2.14](#)).
4. Program the `IC_FS_SPKLEN` register to suppress the maximum spike of 50ns.
5. Program the `IC_SDA_SETUP` register to meet the minimum data setup time (tSU; DAT).

12.2.13. Bus Clear Feature

`DW_apb_i2c` supports the bus clear feature that provides graceful recovery of data `SDA` and clock `SCL` lines during unlikely events in which either the clock or data line is stuck at LOW.

12.2.13.1. SDA Line Stuck at LOW Recovery

In case of `SDA` line stuck at LOW, the master performs the following actions to recover as shown in [Figure 66](#) and [Figure 67](#):

1. Master sends a maximum of nine clock pulses to recover the bus LOW within those nine clocks.
 - o The number of clock pulses will vary with the number of bits that remain to be sent by the slave. As the maximum number of bits is nine, master sends up to nine clock pulses and allows the slave to recover.
 - o The master attempts to assert a Logic 1 on the `SDA` line and check whether `SDA` is recovered. If the `SDA` is not recovered, it will continue to send a maximum of nine `SCL` clocks.
2. If `SDA` line is recovered within nine clock pulses, the master will send STOP to release the bus.
3. If `SDA` line is not recovered even after the ninth clock pulse, you must hardware reset the system.

Figure 66. SDA Recovery with 9 SCL Clocks

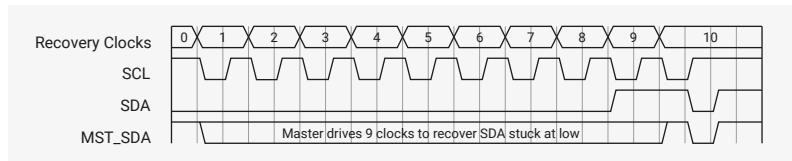
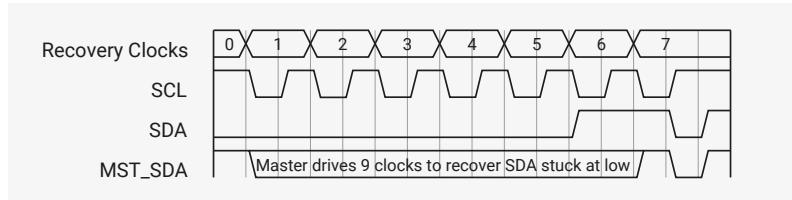


Figure 67. SDA Recovery with 6 SCL Clocks



12.2.13.2. SCL Line is Stuck at LOW

In the unlikely event (due to an electric failure of a circuit) where the clock (**SCL**) is stuck to LOW, there is no effective method to overcome this problem. Instead, reset the bus using the hardware reset signal.

12.2.14. IC_CLK Frequency Configuration

When the **DW_apb_i2c** is configured as a Standard (SS), Fast (FS), or Fast-Mode Plus (FM+), the ***CNT** registers must be set before any I2C bus transaction can take place in order to ensure proper I/O timing. The ***CNT** registers are:

- [IC_SS_SCL_HCNT](#)
- [IC_SS_SCL_LCNT](#)
- [IC_FS_SCL_HCNT](#)
- [IC_FS_SCL_LCNT](#)

NOTE

The tBUF timing and setup/hold time of START, STOP and RESTART registers uses ***HCNT**/***LCNT** register settings for the corresponding speed mode.

NOTE

It is not necessary to program any of the ***CNT** registers if the **DW_apb_i2c** is enabled to operate only as an I2C slave, since these registers are used only to determine the **SCL** timing requirements for operation as an I2C master.

Table 1005 lists the derivation of I2C timing parameters from the ***CNT** programming registers.

Table 1005. Derivation of I2C Timing Parameters from ***CNT** Registers

Timing Parameter	Symbol	Standard Speed	Fast Speed / Fast Speed Plus
LOW period of the SCL clock	tLOW	IC_SS_SCL_LCNT	IC_FS_SCL_LCNT
HIGH period of the SCL clock	tHIGH	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT
Setup time for a repeated START condition	tSU;STA	IC_SS_SCL_LCNT	IC_FS_SCL_HCNT
Hold time (repeated) START condition	tHD;STA	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT
Setup time for STOP condition	tSU;STO	IC_SS_SCL_HCNT	IC_FS_SCL_HCNT

Timing Parameter	Symbol	Standard Speed	Fast Speed / Fast Speed Plus
Bus free time between a STOP and a START condition	tBUF	IC_SS_SCL_LCNT	IC_FS_SCL_LCNT
Spike length	tSP	IC_FS_SPKLEN	IC_FS_SPKLEN
Data hold time	tHD;DAT	IC_SDA_HOLD	IC_SDA_HOLD
Data setup time	tSU;DAT	IC_SDA_SETUP	IC_SDA_SETUP

12.2.14.1. Minimum High and Low Counts in SS, FS, and FM+ Modes.

When the `DW_apb_i2c` operates as an I2C master, in both transmit and receive transfers:

- `IC_SS_SCL_LCNT` and `IC_FS_SCL_LCNT` register values must be larger than `IC_FS_SPKLEN` + 7.
- `IC_SS_SCL_HCNT` and `IC_FS_SCL_HCNT` register values must be larger than `IC_FS_SPKLEN` + 5.

Details regarding the `DW_apb_i2c` high and low counts are as follows:

- The minimum value of `IC_*_SPKLEN + 7` for the `*_LCNT` registers is due to the time required for the `DW_apb_i2c` to drive `SDA` after a negative edge of `SCL`.
- The minimum value of `IC_*_SPKLEN + 5` for the `*_HCNT` registers is due to the time required for the `DW_apb_i2c` to sample `SDA` during the high period of `SCL`.
- The `DW_apb_i2c` adds one cycle to the programmed `*_LCNT` value in order to generate the low period of the `SCL` clock; this is due to the counting logic for `SCL` low counting to $(*_\text{LCNT} + 1)$.
- The `DW_apb_i2c` adds `IC_*_SPKLEN + 7` cycles to the programmed `*_HCNT` value in order to generate the high period of the `SCL` clock, due to the following factors:
 - The counting logic for `SCL` high counts to $(*_\text{HCNT} + 1)$.
 - The digital filtering applied to the `SCL` line incurs a delay of `SPKLEN + 2 ic_clk` cycles, where `SPKLEN` is `IC_FS_SPKLEN` if the component is operating in SS or FS.
 - Whenever `SCL` is driven one to zero by the `DW_apb_i2c` (completing the `SCL` high time) an internal logic latency of three `ic_clk` cycles is incurred. Consequently, the minimum `SCL` low time of which the `DW_apb_i2c` is capable is nine `ic_clk` periods ($7 + 1 + 1$), while the minimum `SCL` high time is thirteen `ic_clk` periods ($6 + 1 + 3 + 3$).

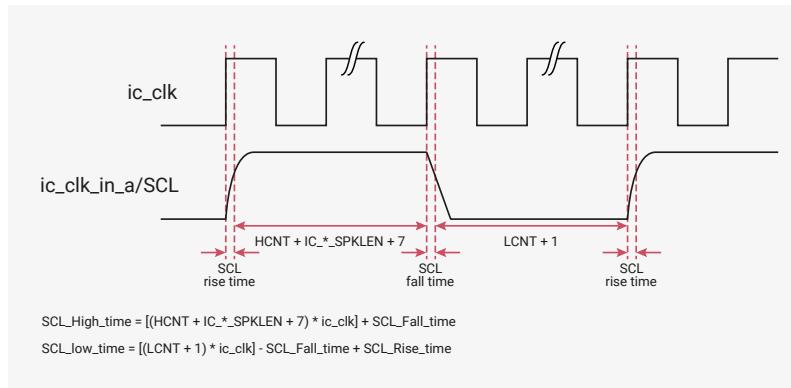
NOTE

The total high time and low time of `SCL` generated by the `DW_apb_i2c` master is also influenced by the rise time and fall time of the `SCL` line, as shown in the illustration and equations in Figure 68. `SCL` rise and fall time parameters vary depending on external factors such as:

- Characteristics of the IO driver
- Pull-up resistor value
- Total capacitance on `SCL` line

These characteristics are beyond the control of the `DW_apb_i2c`.

Figure 68. Impact of SCL Rise Time and Fall Time on Generated SCL



12.2.14.2. Minimum IC_CLK Frequency

This section describes the minimum `ic_clk` frequencies that the `DW_apb_i2c` supports for each speed mode, and the associated high and low count values. In slave mode, `IC_SDA_HOLD` (Thd:dat) and `IC_SDA_SETUP` (Tsu:dat) need to be programmed to satisfy the I2C protocol timing requirements. The following examples are for the case where `IC_FS_SPKLEN` is programmed to two.

12.2.14.2.1. Standard Mode (SM), Fast Mode (FM), and Fast Mode Plus (FM+)

This section details how to derive a minimum `ic_clk` value for standard and fast modes of the `DW_apb_i2c`. Although the following method shows how to do fast mode calculations, you can also use the same method in order to do calculations for standard mode and fast mode plus.

NOTE

The following computations do not consider the `SCL_Rise_time` and `SCL_Fall_time`.

Given conditions and calculations for the minimum `DW_apb_i2c ic_clk` value in fast mode:

- Fast mode has data rate of 400kb/s; implies `SCL` period of $1/400\text{kHz} = 2.5\mu\text{s}$
- Minimum `hcnt` value of 14 as a seed value; `IC_HCNT_FS = 14`
- Protocol minimum `SCL` high and low times:
 - `MIN_SCL_LOWtime_FS = 1300ns`
 - `MIN_SCL_HIGHTime_FS = 600ns`

Derived equations:

$$\text{SCL_PERIOD_FS} / (\text{IC_HCNT_FS} + \text{IC_LCNT_FS}) = \text{IC_CLK_PERIOD}$$

$$\text{IC_LCNT_FS} \times \text{IC_CLK_PERIOD} = \text{MIN_SCL_LOWtime_FS}$$

Combined, the previous equations produce the following:

$$\text{IC_LCNT_FS} \times (\text{SCL_PERIOD_FS} / (\text{IC_LCNT_FS} + \text{IC_HCNT_FS})) = \text{MIN_SCL_LOWtime_FS}$$

Solving for `IC_LCNT_FS`:

$$IC_LCNT_FS \times (2.5\mu s / (IC_LCNT_FS + 14)) = 1.3\mu s$$

The previous equation gives:

$$IC_LCNT_FS = \text{roundup}(15.166) = 16$$

These calculations produce $IC_LCNT_FS = 16$ and $IC_HCNT_FS = 14$, giving an ic_clk value of:

$$2.5\mu s / (16 + 14) = 83.3\text{ns} = 12\text{MHz}$$

Testing these results shows that protocol requirements are satisfied.

Table 1006 lists the minimum ic_clk values for all modes with high and low count values.

Table 1006. ic_clk in Relation to High and Low Counts

Speed Mode	ic_clkfreq (MHz)	Minimum Value of IC_*_SPKLEN	SCL Low Time in 'ic_clk's	SCL Low Program Value	SCL Low Time	SCL High Time in 'ic_clk's	SCL High Program Value	SCL High Time
SS	2.7	1	13	12	4.7μs	14	6	5.2μs
FS	12.0	1	16	15	1.33μs	14	6	1.16μs
FM+	32	2	16	15	500ns	16	7	500ns

- The $IC_*_SCL_LCNT$ and $IC_*_SCL_HCNT$ registers are programmed using the SCL low and high program values in Table 1006, which are calculated using SCL low count minus one, and SCL high counts minus eight, respectively. The values in Table 1006 are based on $IC_SDA_RX_HOLD = 0$. The maximum $IC_SDA_RX_HOLD$ value depends on the IC_*_CNT registers in Master mode.
- In order to compute the HCNT and LCNT considering RC timings, use the following equations:
 - $IC_HCNT_* = [(HCNT + IC_*_SPKLEN + 7) * ic_clk] + SCL_Fall_time$
 - $IC_LCNT_* = [(LCNT + 1) * ic_clk] - SCL_Fall_time + SCL_Rise_time$

12.2.14.3. Calculating High and Low Counts

The calculations below show how to calculate SCL high and low counts for each speed mode in the DW_apb_i2c . For the calculations to work, the ic_clk frequencies used must not be less than the minimum ic_clk frequencies specified in Table 1006.

The default ic_clk period value is set to 100ns, so default SCL high and low count values are calculated for each speed mode based on this clock. These values need updating according to the guidelines below.

The equation to calculate the proper number of ic_clk signals required for setting the proper SCL clocks high and low times is as follows:

$$IC_xCNT = (\text{ROUNDUP}(\text{MIN_SCL_xxxtimes*OSCFREQ}, 0))$$

MIN_SCL_HIGHtime = Minimum High Period

MIN_SCL_HIGHtime = 4000ns for 100kb/s,
600ns for 400kb/s,
260ns for 1000kb/s,

MIN_SCL_LOWtime = Minimum Low Period

MIN_SCL_LOWtime = 4700ns for 100kb/s,

```

1300ns for 400kb/s,
500ns for 1000kb/s,
OSCFREQ = ic_clk Clock Frequency (Hz).

```

For example:

```

OSCFREQ = 100MHz
I2Cmode = fast, 400kb/s
MIN_SCL_HIGHTime = 600ns.
MIN_SCL_LOWtime = 1300ns.

IC_xCNT = (ROUNDUP(MIN_SCL_HIGH_LOWtime*OSCFREQ, 0))

IC_HCNT = (ROUNDUP(600ns * 100MHz, 0))
IC_HCNTSCL PERIOD = 60
IC_LCNT = (ROUNDUP(1300ns * 100MHz, 0))
IC_LCNTSCL PERIOD = 130
Actual MIN_SCL_HIGHTime = 60*(1/100MHz) = 600ns
Actual MIN_SCL_LOWtime = 130*(1/100MHz) = 1300ns

```

12.2.15. DMA Controller Interface

The [DW_apb_i2c](#) has built-in DMA capability; it has a handshaking interface to the DMA Controller to request and control transfers. The APB bus is used to perform data transfers to and from the DMA. DMA transfers use single accesses, since the data rate is relatively low.

12.2.15.1. Enabling the DMA Controller Interface

To enable the DMA Controller interface on the [DW_apb_i2c](#), you must write the DMA Control Register ([IC_DMA_CR](#)). Writing a one into the [TDMAE](#) bit field of [IC_DMA_CR](#) register enables the [DW_apb_i2c](#) transmit handshaking interface. Writing a one into the [RDMAE](#) bit field of the [IC_DMA_CR](#) register enables the [DW_apb_i2c](#) receive handshaking interface.

12.2.15.2. Overview of Operation

The DMA Controller is programmed with the number of data items (transfer count) that are to be transmitted or received by [DW_apb_i2c](#).

The transfer is broken into single transfers on the bus, each initiated by a request from the [DW_apb_i2c](#).

For example, where the transfer count programmed into the DMA Controller is four. The DMA transfer consists of a series of four single transactions. If the [DW_apb_i2c](#) makes a transmit request to this channel, a single data item is written to the [DW_apb_i2c](#) Tx FIFO. Similarly, if the [DW_apb_i2c](#) makes a receive request to this channel, a single data item is read from the [DW_apb_i2c](#) Rx FIFO. Four separate requests must be made to this DMA channel before all four data items are written or read.

12.2.15.3. Watermark Levels

In [DW_apb_i2c](#) the registers for setting watermarks to allow DMA bursts do not need to be set to anything other than their reset value. Specifically, [IC_DMA_TDLR](#) and [IC_DMA_RDLR](#) can be left at reset values of zero. This is because only single transfers are needed due to the low bandwidth of I2C relative to system bandwidth. Because the DMA controller normally has the highest priority on the system bus, transfers complete quickly.

12.2.16. Operation of Interrupt Registers

Table 1007 lists the operation of the `DW_apb_i2c` interrupt registers and how they are set and cleared. Some bits are set by hardware and cleared by software, whereas other bits are set and cleared by hardware.

Table 1007. Clearing and Setting of Interrupt Registers

Interrupt Bit Fields	Set by Hardware/Cleared by Software	Set and Cleared by Hardware
RESTART_DET	Y	N
GEN_CALL	Y	N
START_DET	Y	N
STOP_DET	Y	N
ACTIVITY	Y	N
RX_DONE	Y	N
TX_ABRT	Y	N
RD_REQ	Y	N
TX_EMPTY	N	Y
TX_OVER	Y	N
RX_FULL	N	Y
RX_OVER	Y	N
RX_UNDER	Y	N

12.2.17. List of Registers

The I2C0 and I2C1 registers start at base addresses of `0x40090000` and `0x40098000` respectively (defined as `I2C0_BASE` and `I2C1_BASE` in SDK).

NOTE

You may see references to configuration constants in the I2C register descriptions; these are **fixed** values, set at hardware design time. A full list of their values can be found in https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2350/hardware_regs/include/hardware/regs/i2c.h

Table 1008. List of I2C registers

Offset	Name	Info
0x00	IC_CON	I2C Control Register
0x04	IC_TAR	I2C Target Address Register
0x08	IC_SAR	I2C Slave Address Register
0x10	IC_DATA_CMD	I2C Rx/Tx Data Buffer and Command Register
0x14	IC_SS_SCL_HCNT	Standard Speed I2C Clock SCL High Count Register
0x18	IC_SS_SCL_LCNT	Standard Speed I2C Clock SCL Low Count Register
0x1c	IC_FS_SCL_HCNT	Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register
0x20	IC_FS_SCL_LCNT	Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register
0x2c	IC_INTR_STAT	I2C Interrupt Status Register
0x30	IC_INTR_MASK	I2C Interrupt Mask Register

Offset	Name	Info
0x34	IC_RAW_INTR_STAT	I2C Raw Interrupt Status Register
0x38	IC_RX_TL	I2C Receive FIFO Threshold Register
0x3c	IC_TX_TL	I2C Transmit FIFO Threshold Register
0x40	IC_CLR_INTR	Clear Combined and Individual Interrupt Register
0x44	IC_CLR_RX_UNDER	Clear RX_UNDER Interrupt Register
0x48	IC_CLR_RX_OVER	Clear RX_OVER Interrupt Register
0x4c	IC_CLR_TX_OVER	Clear TX_OVER Interrupt Register
0x50	IC_CLR_RD_REQ	Clear RD_REQ Interrupt Register
0x54	IC_CLR_TX_ABRT	Clear TX_ABRT Interrupt Register
0x58	IC_CLR_RX_DONE	Clear RX_DONE Interrupt Register
0x5c	IC_CLR_ACTIVITY	Clear ACTIVITY Interrupt Register
0x60	IC_CLR_STOP_DET	Clear STOP_DET Interrupt Register
0x64	IC_CLR_START_DET	Clear START_DET Interrupt Register
0x68	IC_CLR_GEN_CALL	Clear GEN_CALL Interrupt Register
0x6c	IC_ENABLE	I2C ENABLE Register
0x70	IC_STATUS	I2C STATUS Register
0x74	IC_TXFLR	I2C Transmit FIFO Level Register
0x78	IC_RXFLR	I2C Receive FIFO Level Register
0x7c	IC_SDA_HOLD	I2C SDA Hold Time Length Register
0x80	IC_TX_ABRT_SOURCE	I2C Transmit Abort Source Register
0x84	IC_SLV_DATA_NACK_ONLY	Generate Slave Data NACK Register
0x88	IC_DMA_CR	DMA Control Register
0x8c	IC_DMA_TDLR	DMA Transmit Data Level Register
0x90	IC_DMA_RDLR	DMA Transmit Data Level Register
0x94	IC_SDA_SETUP	I2C SDA Setup Register
0x98	IC_ACK_GENERAL_CALL	I2C ACK General Call Register
0x9c	IC_ENABLE_STATUS	I2C Enable Status Register
0xa0	IC_FS_SPKLEN	I2C SS, FS or FM+ spike suppression limit
0xa8	IC_CLR_RESTART_DET	Clear RESTART_DET Interrupt Register
0xf4	IC_COMP_PARAM_1	Component Parameter Register 1
0xf8	IC_COMP_VERSION	I2C Component Version Register
0xfc	IC_COMP_TYPE	I2C Component Type Register

I2C: IC_CON Register

Offset: 0x00

Description

I2C Control Register. This register can be written only when the DW_apb_i2c is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.

Read/Write Access: - bit 10 is read only. - bit 11 is read only - bit 16 is read only - bit 17 is read only - bits 18 and 19 are read only.

Table 1009. IC_CON Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	STOP_DET_IF_MASTER_ACTIVE	Master issues the STOP_DET interrupt irrespective of whether master is active or not	RO	0x0
9	RX_FIFO_FULL_HOLD_CTRL	This bit controls whether DW_apb_i2c should hold the bus when the Rx FIFO is physically full to its RX_BUFFER_DEPTH, as described in the IC_RX_FULL_HLD_BUS_EN parameter. Reset value: 0x0. 0x0 → Overflow when RX_FIFO is full 0x1 → Hold bus when RX_FIFO is full	RW	0x0
8	TX_EMPTY_CTRL	This bit controls the generation of the TX_EMPTY interrupt, as described in the IC_RAW_INTR_STAT register. Reset value: 0x0. 0x0 → Default behaviour of TX_EMPTY interrupt 0x1 → Controlled generation of TX_EMPTY interrupt	RW	0x0
7	STOP_DET_IFADDRESSED	In slave mode: - 1'b1: issues the STOP_DET interrupt only when it is addressed. - 1'b0: issues the STOP_DET irrespective of whether it's addressed or not. Reset value: 0x0 NOTE: During a general call address, this slave does not issue the STOP_DET interrupt if STOP_DET_IF_ADDRESSED = 1'b1, even if the slave responds to the general call address by generating ACK. The STOP_DET interrupt is generated only when the transmitted address matches the slave address (SAR). 0x0 → slave issues STOP_DET intr always 0x1 → slave issues STOP_DET intr only if addressed	RW	0x0
6	IC_SLAVE_DISABLE	This bit controls whether I2C has its slave disabled, which means once the presetn signal is applied, then this bit is set and the slave is disabled. If this bit is set (slave is disabled), DW_apb_i2c functions only as a master and does not perform any action that requires a slave. NOTE: Software should ensure that if this bit is written with 0, then bit 0 should also be written with a 0. 0x0 → Slave mode is enabled 0x1 → Slave mode is disabled	RW	0x1

Bits	Name	Description	Type	Reset
5	IC_RESTART_EN	<p>Determines whether RESTART conditions may be sent when acting as a master. Some older slaves do not support handling RESTART conditions; however, RESTART conditions are used in several DW_apb_i2c operations. When RESTART is disabled, the master is prohibited from performing the following functions:</p> <ul style="list-style-type: none"> - Sending a START BYTE - Performing any high-speed mode operation - High-speed mode operation - Performing direction changes in combined format mode - Performing a read operation with a 10-bit address <p>By replacing RESTART condition followed by a STOP and a subsequent START condition, split operations are broken down into multiple DW_apb_i2c transfers. If the above operations are performed, it will result in setting bit 6 (TX_ABRT) of the IC_RAW_INTR_STAT register.</p> <p>Reset value: ENABLED 0x0 → Master restart disabled 0x1 → Master restart enabled</p>	RW	0x1
4	IC_10BITADDR_MASTER	<p>Controls whether the DW_apb_i2c starts its transfers in 7- or 10-bit addressing mode when acting as a master.</p> <ul style="list-style-type: none"> - 0: 7-bit addressing - 1: 10-bit addressing <p>0x0 → Master 7Bit addressing mode 0x1 → Master 10Bit addressing mode</p>	RW	0x0
3	IC_10BITADDR_SLAVE	<p>When acting as a slave, this bit controls whether the DW_apb_i2c responds to 7- or 10-bit addresses.</p> <ul style="list-style-type: none"> - 0: 7-bit addressing. The DW_apb_i2c ignores transactions that involve 10-bit addressing; for 7-bit addressing, only the lower 7 bits of the IC_SAR register are compared. - 1: 10-bit addressing. The DW_apb_i2c responds to only 10-bit addressing transfers that match the full 10 bits of the IC_SAR register. <p>0x0 → Slave 7Bit addressing 0x1 → Slave 10Bit addressing</p>	RW	0x0

Bits	Name	Description	Type	Reset
2:1	SPEED	<p>These bits control at which speed the DW_apb_i2c operates; its setting is relevant only if one is operating the DW_apb_i2c in master mode. Hardware protects against illegal values being programmed by software. These bits must be programmed appropriately for slave mode also, as it is used to capture correct value of spike filter as per the speed mode.</p> <p>This register should be programmed only with a value in the range of 1 to IC_MAX_SPEED_MODE; otherwise, hardware updates this register with the value of IC_MAX_SPEED_MODE.</p> <p>1: standard mode (100 kbit/s)</p> <p>2: fast mode (<=400 kbit/s) or fast mode plus (<=1000Kbit/s)</p> <p>3: high speed mode (3.4 Mbit/s)</p> <p>Note: This field is not applicable when IC_ULTRA_FAST_MODE=1 0x1 → Standard Speed mode of operation 0x2 → Fast or Fast Plus mode of operation 0x3 → High Speed mode of operation</p>	RW	0x2
0	MASTER_MODE	<p>This bit controls whether the DW_apb_i2c master is enabled.</p> <p>NOTE: Software should ensure that if this bit is written with '1' then bit 6 should also be written with a '1'. 0x0 → Master mode is disabled 0x1 → Master mode is enabled</p>	RW	0x1

I2C: IC_TAR Register

Offset: 0x04

Description

I2C Target Address Register

This register is 12 bits wide, and bits 31:12 are reserved. This register can be written to only when IC_ENABLE[0] is set to 0.

Note: If the software or application is aware that the DW_apb_i2c is not using the TAR address for the pending commands in the Tx FIFO, then it is possible to update the TAR address even while the Tx FIFO has entries (IC_STATUS[2]= 0). - It is not necessary to perform any write to this register if DW_apb_i2c is enabled as an I2C slave only.

Table 1010. IC_TAR Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
11	SPECIAL	This bit indicates whether software performs a Device-ID or General Call or START BYTE command. - 0: ignore bit 10 GC_OR_START and use IC_TAR normally - 1: perform special I2C command as specified in Device_ID or GC_OR_START bit Reset value: 0x0 0x0 → Disables programming of GENERAL_CALL or START_BYTE transmission 0x1 → Enables programming of GENERAL_CALL or START_BYTE transmission	RW	0x0
10	GC_OR_START	If bit 11 (SPECIAL) is set to 1 and bit 13(Device-ID) is set to 0, then this bit indicates whether a General Call or START byte command is to be performed by the DW_apb_i2c. - 0: General Call Address - after issuing a General Call, only writes may be performed. Attempting to issue a read command results in setting bit 6 (TX_ABRT) of the IC_RAW_INTR_STAT register. The DW_apb_i2c remains in General Call mode until the SPECIAL bit value (bit 11) is cleared. - 1: START BYTE Reset value: 0x0 0x0 → GENERAL_CALL byte transmission 0x1 → START byte transmission	RW	0x0
9:0	IC_TAR	This is the target address for any master transaction. When transmitting a General Call, these bits are ignored. To generate a START BYTE, the CPU needs to write only once into these bits. If the IC_TAR and IC_SAR are the same, loopback exists but the FIFOs are shared between master and slave, so full loopback is not feasible. Only one direction loopback mode is supported (simplex), not duplex. A master cannot transmit to itself; it can transmit to only a slave.	RW	0x055

I2C: IC_SAR Register

Offset: 0x08

Description

I2C Slave Address Register

Table 1011. IC_SAR Register

Bits	Name	Description	Type	Reset
31:10	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
9:0	IC_SAR	<p>The IC_SAR holds the slave address when the I2C is operating as a slave. For 7-bit addressing, only IC_SAR[6:0] is used.</p> <p>This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.</p> <p>Note: The default values cannot be any of the reserved address locations: that is, 0x00 to 0x07, or 0x78 to 0x7f. The correct operation of the device is not guaranteed if you program the IC_SAR or IC_TAR to a reserved value. Refer to Table 1004 for a complete list of these reserved values.</p>	RW	0x055

I2C: IC_DATA_CMD Register

Offset: 0x10

Description

I2C Rx/Tx Data Buffer and Command Register; this is the register the CPU writes to when filling the TX FIFO and the CPU reads from when retrieving bytes from RX FIFO.

The size of the register changes as follows:

Write: - 11 bits when IC_EMPTYFIFO_HOLD_MASTER_EN=1 - 9 bits when IC_EMPTYFIFO_HOLD_MASTER_EN=0 Read: - 12 bits when IC_FIRST_DATA_BYTE_STATUS = 1 - 8 bits when IC_FIRST_DATA_BYTE_STATUS = 0 Note: In order for the DW_apb_i2c to continue acknowledging reads, a read command should be written for every byte that is to be received; otherwise the DW_apb_i2c will stop acknowledging.

Table 1012.
IC_DATA_CMD
Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	FIRST_DATA_BYT E	<p>Indicates the first data byte received after the address phase for receive transfer in Master receiver or Slave receiver mode.</p> <p>Reset value : 0x0</p> <p>NOTE: In case of APB_DATA_WIDTH=8,</p> <ol style="list-style-type: none"> 1. The user has to perform two APB Reads to IC_DATA_CMD in order to get status on 11 bit. 2. In order to read the 11 bit, the user has to perform the first data byte read [7:0] (offset 0x10) and then perform the second read [15:8] (offset 0x11) in order to know the status of 11 bit (whether the data received in previous read is a first data byte or not). 3. The 11th bit is an optional read field, user can ignore 2nd byte read [15:8] (offset 0x11) if not interested in FIRST_DATA_BYTE status. <p>0x0 → Sequential data byte received 0x1 → Non sequential data byte received</p>	RO	0x0

Bits	Name	Description	Type	Reset
10	RESTART	<p>This bit controls whether a RESTART is issued before the byte is sent or received.</p> <p>1 - If IC_RESTART_EN is 1, a RESTART is issued before the data is sent/received (according to the value of CMD), regardless of whether or not the transfer direction is changing from the previous command; if IC_RESTART_EN is 0, a STOP followed by a START is issued instead.</p> <p>0 - If IC_RESTART_EN is 1, a RESTART is issued only if the transfer direction is changing from the previous command; if IC_RESTART_EN is 0, a STOP followed by a START is issued instead.</p> <p>Reset value: 0x0 0x0 → Don't Issue RESTART before this command 0x1 → Issue RESTART before this command</p>	SC	0x0
9	STOP	<p>This bit controls whether a STOP is issued after the byte is sent or received.</p> <p>- 1 - STOP is issued after this byte, regardless of whether or not the Tx FIFO is empty. If the Tx FIFO is not empty, the master immediately tries to start a new transfer by issuing a START and arbitrating for the bus. - 0 - STOP is not issued after this byte, regardless of whether or not the Tx FIFO is empty. If the Tx FIFO is not empty, the master continues the current transfer by sending/receiving data bytes according to the value of the CMD bit. If the Tx FIFO is empty, the master holds the SCL line low and stalls the bus until a new command is available in the Tx FIFO.</p> <p>Reset value: 0x0 0x0 → Don't Issue STOP after this command 0x1 → Issue STOP after this command</p>	SC	0x0

Bits	Name	Description	Type	Reset
8	CMD	<p>This bit controls whether a read or a write is performed. This bit does not control the direction when the DW_apb_i2con acts as a slave. It controls only the direction when it acts as a master.</p> <p>When a command is entered in the TX FIFO, this bit distinguishes the write and read commands. In slave-receiver mode, this bit is a 'don't care' because writes to this register are not required. In slave-transmitter mode, a '0' indicates that the data in IC_DATA_CMD is to be transmitted.</p> <p>When programming this bit, you should remember the following: attempting to perform a read operation after a General Call command has been sent results in a TX_ABRT interrupt (bit 6 of the IC_RAW_INTR_STAT register), unless bit 11 (SPECIAL) in the IC_TAR register has been cleared. If a '1' is written to this bit after receiving a RD_REQ interrupt, then a TX_ABRT interrupt occurs.</p> <p>Reset value: 0x0 0x0 → Master Write Command 0x1 → Master Read Command</p>	SC	0x0
7:0	DAT	<p>This register contains the data to be transmitted or received on the I2C bus. If you are writing to this register and want to perform a read, bits 7:0 (DAT) are ignored by the DW_apb_i2c. However, when you read this register, these bits return the value of data received on the DW_apb_i2c interface.</p> <p>Reset value: 0x0</p>	RW	0x00

I2C: IC_SS_SCL_HCNT Register

Offset: 0x14

Description

Standard Speed I2C Clock SCL High Count Register

Table 1013.
IC_SS_SCL_HCNT
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
15:0	IC_SS_SCL_HCNT	<p>This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high-period count for standard speed. For more information, refer to 'IC_CLK Frequency Configuration'.</p> <p>This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in 6 being set. For designs with APB_DATA_WIDTH = 8, the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed.</p> <p>NOTE: This register must not be programmed to a value higher than 65525, because DW_apb_i2c uses a 16-bit counter to flag an I2C bus idle condition when this counter reaches a value of IC_SS_SCL_HCNT + 10.</p>	RW	0x0028

I2C: IC_SS_SCL_LCNT Register

Offset: 0x18

Description

Standard Speed I2C Clock SCL Low Count Register

Table 1014.
IC_SS_SCL_LCNT
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:0	IC_SS_SCL_LCNT	<p>This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for standard speed. For more information, refer to 'IC_CLK Frequency Configuration'.</p> <p>This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 8; hardware prevents values less than this being written, and if attempted, results in 8 being set. For designs with APB_DATA_WIDTH = 8, the order of programming is important to ensure the correct operation of DW_apb_i2c. The lower byte must be programmed first, and then the upper byte is programmed.</p>	RW	0x002f

I2C: IC_FS_SCL_HCNT Register

Offset: 0x1c

Description

Fast Mode or Fast Mode Plus I2C Clock SCL High Count Register

Table 1015.
IC_FS_SCL_HCNT
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:0	IC_FS_SCL_HCNT	<p>This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock high-period count for fast mode or fast mode plus. It is used in high-speed mode to send the Master Code and START BYTE or General CALL. For more information, refer to 'IC_CLK Frequency Configuration'.</p> <p>This register goes away and becomes read-only returning 0s if IC_MAX_SPEED_MODE = standard. This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 6; hardware prevents values less than this being written, and if attempted results in 6 being set. For designs with APB_DATA_WIDTH == 8 the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed.</p>	RW	0x0006

I2C: IC_FS_SCL_LCNT Register

Offset: 0x20

Description

Fast Mode or Fast Mode Plus I2C Clock SCL Low Count Register

Table 1016.
IC_FS_SCL_LCNT
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
15:0	IC_FS_SCL_LCNT	<p>This register must be set before any I2C bus transaction can take place to ensure proper I/O timing. This register sets the SCL clock low period count for fast speed. It is used in high-speed mode to send the Master Code and START BYTE or General CALL. For more information, refer to 'IC_CLK Frequency Configuration'.</p> <p>This register goes away and becomes read-only returning 0s if IC_MAX_SPEED_MODE = standard.</p> <p>This register can be written only when the I2C interface is disabled, which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect.</p> <p>The minimum valid value is 8; hardware prevents values less than this being written, and if attempted results in 8 being set. For designs with APB_DATA_WIDTH = 8 the order of programming is important to ensure the correct operation of the DW_apb_i2c. The lower byte must be programmed first. Then the upper byte is programmed. If the value is less than 8 then the count value gets changed to 8.</p>	RW	0x000d

I2C: IC_INTR_STAT Register

Offset: 0x2c

Description

I2C Interrupt Status Register

Each bit in this register has a corresponding mask bit in the IC_INTR_MASK register. These bits are cleared by reading the matching interrupt clear register. The unmasked raw versions of these bits are available in the IC_RAW_INTR_STAT register.

Table 1017.
IC_INTR_STAT
Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	R_RESTART_DET	<p>See IC_RAW_INTR_STAT for a detailed description of R_RESTART_DET bit.</p> <p>Reset value: 0x0</p> <p>0x0 → R_RESTART_DET interrupt is inactive</p> <p>0x1 → R_RESTART_DET interrupt is active</p>	RO	0x0
11	R_GEN_CALL	<p>See IC_RAW_INTR_STAT for a detailed description of R_GEN_CALL bit.</p> <p>Reset value: 0x0</p> <p>0x0 → R_GEN_CALL interrupt is inactive</p> <p>0x1 → R_GEN_CALL interrupt is active</p>	RO	0x0

Bits	Name	Description	Type	Reset
10	R_START_DET	See IC_RAW_INTR_STAT for a detailed description of R_START_DET bit. Reset value: 0x0 0x0 → R_START_DET interrupt is inactive 0x1 → R_START_DET interrupt is active	RO	0x0
9	R_STOP_DET	See IC_RAW_INTR_STAT for a detailed description of R_STOP_DET bit. Reset value: 0x0 0x0 → R_STOP_DET interrupt is inactive 0x1 → R_STOP_DET interrupt is active	RO	0x0
8	R_ACTIVITY	See IC_RAW_INTR_STAT for a detailed description of R_ACTIVITY bit. Reset value: 0x0 0x0 → R_ACTIVITY interrupt is inactive 0x1 → R_ACTIVITY interrupt is active	RO	0x0
7	R_RX_DONE	See IC_RAW_INTR_STAT for a detailed description of R_RX_DONE bit. Reset value: 0x0 0x0 → R_RX_DONE interrupt is inactive 0x1 → R_RX_DONE interrupt is active	RO	0x0
6	R_TX_ABRT	See IC_RAW_INTR_STAT for a detailed description of R_TX_ABRT bit. Reset value: 0x0 0x0 → R_TX_ABRT interrupt is inactive 0x1 → R_TX_ABRT interrupt is active	RO	0x0
5	R_RD_REQ	See IC_RAW_INTR_STAT for a detailed description of R_RD_REQ bit. Reset value: 0x0 0x0 → R_RD_REQ interrupt is inactive 0x1 → R_RD_REQ interrupt is active	RO	0x0
4	R_TX_EMPTY	See IC_RAW_INTR_STAT for a detailed description of R_TX_EMPTY bit. Reset value: 0x0 0x0 → R_TX_EMPTY interrupt is inactive 0x1 → R_TX_EMPTY interrupt is active	RO	0x0
3	R_TX_OVER	See IC_RAW_INTR_STAT for a detailed description of R_TX_OVER bit. Reset value: 0x0 0x0 → R_TX_OVER interrupt is inactive 0x1 → R_TX_OVER interrupt is active	RO	0x0

Bits	Name	Description	Type	Reset
2	R_RX_FULL	See IC_RAW_INTR_STAT for a detailed description of R_RX_FULL bit. Reset value: 0x0 0x0 → R_RX_FULL interrupt is inactive 0x1 → R_RX_FULL interrupt is active	RO	0x0
1	R_RX_OVER	See IC_RAW_INTR_STAT for a detailed description of R_RX_OVER bit. Reset value: 0x0 0x0 → R_RX_OVER interrupt is inactive 0x1 → R_RX_OVER interrupt is active	RO	0x0
0	R_RX_UNDER	See IC_RAW_INTR_STAT for a detailed description of R_RX_UNDER bit. Reset value: 0x0 0x0 → RX_UNDER interrupt is inactive 0x1 → RX_UNDER interrupt is active	RO	0x0

I2C: IC_INTR_MASK Register

Offset: 0x30

Description

I2C Interrupt Mask Register.

These bits mask their corresponding interrupt status bits. This register is active low; a value of 0 masks the interrupt, whereas a value of 1 unmasks the interrupt.

Table 1018.
IC_INTR_MASK
Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	M_RESTART_DET	This bit masks the R_RESTART_DET interrupt in IC_INTR_STAT register. Reset value: 0x0 0x0 → RESTART_DET interrupt is masked 0x1 → RESTART_DET interrupt is unmasked	RW	0x0
11	M_GEN_CALL	This bit masks the R_GEN_CALL interrupt in IC_INTR_STAT register. Reset value: 0x1 0x0 → GEN_CALL interrupt is masked 0x1 → GEN_CALL interrupt is unmasked	RW	0x1
10	M_START_DET	This bit masks the R_START_DET interrupt in IC_INTR_STAT register. Reset value: 0x0 0x0 → START_DET interrupt is masked 0x1 → START_DET interrupt is unmasked	RW	0x0

Bits	Name	Description	Type	Reset
9	M_STOP_DET	<p>This bit masks the R_STOP_DET interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x0 0x0 → STOP_DET interrupt is masked 0x1 → STOP_DET interrupt is unmasked</p>	RW	0x0
8	M_ACTIVITY	<p>This bit masks the R_ACTIVITY interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x0 0x0 → ACTIVITY interrupt is masked 0x1 → ACTIVITY interrupt is unmasked</p>	RW	0x0
7	M_RX_DONE	<p>This bit masks the R_RX_DONE interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → RX_DONE interrupt is masked 0x1 → RX_DONE interrupt is unmasked</p>	RW	0x1
6	M_TX_ABRT	<p>This bit masks the R_TX_ABRT interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → TX_ABORT interrupt is masked 0x1 → TX_ABORT interrupt is unmasked</p>	RW	0x1
5	M_RD_REQ	<p>This bit masks the R_RD_REQ interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → RD_REQ interrupt is masked 0x1 → RD_REQ interrupt is unmasked</p>	RW	0x1
4	M_TX_EMPTY	<p>This bit masks the R_TX_EMPTY interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → TX_EMPTY interrupt is masked 0x1 → TX_EMPTY interrupt is unmasked</p>	RW	0x1
3	M_TX_OVER	<p>This bit masks the R_TX_OVER interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → TX_OVER interrupt is masked 0x1 → TX_OVER interrupt is unmasked</p>	RW	0x1
2	M_RX_FULL	<p>This bit masks the R_RX_FULL interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → RX_FULL interrupt is masked 0x1 → RX_FULL interrupt is unmasked</p>	RW	0x1

Bits	Name	Description	Type	Reset
1	M_RX_OVER	<p>This bit masks the R_RX_OVER interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → RX_OVER interrupt is masked 0x1 → RX_OVER interrupt is unmasked</p>	RW	0x1
0	M_RX_UNDER	<p>This bit masks the R_RX_UNDER interrupt in IC_INTR_STAT register.</p> <p>Reset value: 0x1 0x0 → RX_UNDER interrupt is masked 0x1 → RX_UNDER interrupt is unmasked</p>	RW	0x1

I2C: IC_RAW_INTR_STAT Register

Offset: 0x34

Description

I2C Raw Interrupt Status Register

Unlike the IC_INTR_STAT register, these bits are not masked so they always show the true status of the DW_apb_i2c.

Table 1019.
IC_RAW_INTR_STAT Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	RESTART_DET	<p>Indicates whether a RESTART condition has occurred on the I2C interface when DW_apb_i2c is operating in Slave mode and the slave is being addressed. Enabled only when IC_SLV_RESTART_DET_EN=1.</p> <p>Note: However, in high-speed mode or during a START BYTE transfer, the RESTART comes before the address field as per the I2C protocol. In this case, the slave is not the addressed slave when the RESTART is issued, therefore DW_apb_i2c does not generate the RESTART_DET interrupt.</p> <p>Reset value: 0x0 0x0 → RESTART_DET interrupt is inactive 0x1 → RESTART_DET interrupt is active</p>	RO	0x0
11	GEN_CALL	<p>Set only when a General Call address is received and it is acknowledged. It stays set until it is cleared either by disabling DW_apb_i2c or when the CPU reads bit 0 of the IC_CLR_GEN_CALL register. DW_apb_i2c stores the received data in the Rx buffer.</p> <p>Reset value: 0x0 0x0 → GEN_CALL interrupt is inactive 0x1 → GEN_CALL interrupt is active</p>	RO	0x0

Bits	Name	Description	Type	Reset
10	START_DET	<p>Indicates whether a START or RESTART condition has occurred on the I2C interface regardless of whether DW_apb_i2c is operating in slave or master mode.</p> <p>Reset value: 0x0 0x0 → START_DET interrupt is inactive 0x1 → START_DET interrupt is active</p>	RO	0x0
9	STOP_DET	<p>Indicates whether a STOP condition has occurred on the I2C interface regardless of whether DW_apb_i2c is operating in slave or master mode.</p> <p>In Slave Mode: - If IC_CON[7]=1'b1 (STOP_DET_IFADDRESSED), the STOP_DET interrupt will be issued only if slave is addressed. Note: During a general call address, this slave does not issue a STOP_DET interrupt if STOP_DET_IF_ADDRESSED=1'b1, even if the slave responds to the general call address by generating ACK. The STOP_DET interrupt is generated only when the transmitted address matches the slave address (SAR). - If IC_CON[7]=1'b0 (STOP_DET_IFADDRESSED), the STOP_DET interrupt is issued irrespective of whether it is being addressed. In Master Mode: - If IC_CON[10]=1'b1 (STOP_DET_IF_MASTER_ACTIVE), the STOP_DET interrupt will be issued only if Master is active. - If IC_CON[10]=1'b0 (STOP_DET_IFADDRESSED), the STOP_DET interrupt will be issued irrespective of whether master is active or not.</p> <p>Reset value: 0x0 0x0 → STOP_DET interrupt is inactive 0x1 → STOP_DET interrupt is active</p>	RO	0x0
8	ACTIVITY	<p>This bit captures DW_apb_i2c activity and stays set until it is cleared. There are four ways to clear it: - Disabling the DW_apb_i2c - Reading the IC_CLR_ACTIVITY register - Reading the IC_CLR_INTR register - System reset Once this bit is set, it stays set unless one of the four methods is used to clear it. Even if the DW_apb_i2c module is idle, this bit remains set until cleared, indicating that there was activity on the bus.</p> <p>Reset value: 0x0 0x0 → RAW_INTR_ACTIVITY interrupt is inactive 0x1 → RAW_INTR_ACTIVITY interrupt is active</p>	RO	0x0
7	RX_DONE	<p>When the DW_apb_i2c is acting as a slave-transmitter, this bit is set to 1 if the master does not acknowledge a transmitted byte. This occurs on the last byte of the transmission, indicating that the transmission is done.</p> <p>Reset value: 0x0 0x0 → RX_DONE interrupt is inactive 0x1 → RX_DONE interrupt is active</p>	RO	0x0

Bits	Name	Description	Type	Reset
6	TX_ABRT	<p>This bit indicates if DW_apb_i2c, as an I2C transmitter, is unable to complete the intended actions on the contents of the transmit FIFO. This situation can occur both as an I2C master or an I2C slave, and is referred to as a 'transmit abort'. When this bit is set to 1, the IC_TX_ABRT_SOURCE register indicates the reason why the transmit abort takes places.</p> <p>Note: The DW_apb_i2c flushes/resets/empties the TX_FIFO and RX_FIFO whenever there is a transmit abort caused by any of the events tracked by the IC_TX_ABRT_SOURCE register. The FIFOs remains in this flushed state until the register IC_CLR_TX_ABRT is read. Once this read is performed, the Tx FIFO is then ready to accept more data bytes from the APB interface.</p> <p>Reset value: 0x0 0x0 → TX_ABRT interrupt is inactive 0x1 → TX_ABRT interrupt is active</p>	RO	0x0
5	RD_REQ	<p>This bit is set to 1 when DW_apb_i2c is acting as a slave and another I2C master is attempting to read data from DW_apb_i2c. The DW_apb_i2c holds the I2C bus in a wait state (SCL=0) until this interrupt is serviced, which means that the slave has been addressed by a remote master that is asking for data to be transferred. The processor must respond to this interrupt and then write the requested data to the IC_DATA_CMD register. This bit is set to 0 just after the processor reads the IC_CLR_RD_REQ register.</p> <p>Reset value: 0x0 0x0 → RD_REQ interrupt is inactive 0x1 → RD_REQ interrupt is active</p>	RO	0x0
4	TX_EMPTY	<p>The behavior of the TX_EMPTY interrupt status differs based on the TX_EMPTY_CTRL selection in the IC_CON register. - When TX_EMPTY_CTRL = 0: This bit is set to 1 when the transmit buffer is at or below the threshold value set in the IC_TX_TL register. - When TX_EMPTY_CTRL = 1: This bit is set to 1 when the transmit buffer is at or below the threshold value set in the IC_TX_TL register and the transmission of the address/data from the internal shift register for the most recently popped command is completed. It is automatically cleared by hardware when the buffer level goes above the threshold. When IC_ENABLE[0] is set to 0, the TX FIFO is flushed and held in reset. There the TX FIFO looks like it has no data within it, so this bit is set to 1, provided there is activity in the master or slave state machines. When there is no longer any activity, then with ic_en=0, this bit is set to 0.</p> <p>Reset value: 0x0. 0x0 → TX_EMPTY interrupt is inactive 0x1 → TX_EMPTY interrupt is active</p>	RO	0x0

Bits	Name	Description	Type	Reset
3	TX_OVER	<p>Set during transmit if the transmit buffer is filled to IC_TX_BUFFER_DEPTH and the processor attempts to issue another I2C command by writing to the IC_DATA_CMD register. When the module is disabled, this bit keeps its level until the master or slave state machines go into idle, and when ic_en goes to 0, this interrupt is cleared.</p> <p>Reset value: 0x0 0x0 → TX_OVER interrupt is inactive 0x1 → TX_OVER interrupt is active</p>	RO	0x0
2	RX_FULL	<p>Set when the receive buffer reaches or goes above the RX_TL threshold in the IC_RX_TL register. It is automatically cleared by hardware when buffer level goes below the threshold. If the module is disabled (IC_ENABLE[0]=0), the RX FIFO is flushed and held in reset; therefore the RX FIFO is not full. So this bit is cleared once the IC_ENABLE bit 0 is programmed with a 0, regardless of the activity that continues.</p> <p>Reset value: 0x0 0x0 → RX_FULL interrupt is inactive 0x1 → RX_FULL interrupt is active</p>	RO	0x0
1	RX_OVER	<p>Set if the receive buffer is completely filled to IC_RX_BUFFER_DEPTH and an additional byte is received from an external I2C device. The DW_apb_i2c acknowledges this, but any data bytes received after the FIFO is full are lost. If the module is disabled (IC_ENABLE[0]=0), this bit keeps its level until the master or slave state machines go into idle, and when ic_en goes to 0, this interrupt is cleared.</p> <p>Note: If bit 9 of the IC_CON register (RX_FIFO_FULL_HLD_CTRL) is programmed to HIGH, then the RX_OVER interrupt never occurs, because the Rx FIFO never overflows.</p> <p>Reset value: 0x0 0x0 → RX_OVER interrupt is inactive 0x1 → RX_OVER interrupt is active</p>	RO	0x0
0	RX_UNDER	<p>Set if the processor attempts to read the receive buffer when it is empty by reading from the IC_DATA_CMD register. If the module is disabled (IC_ENABLE[0]=0), this bit keeps its level until the master or slave state machines go into idle, and when ic_en goes to 0, this interrupt is cleared.</p> <p>Reset value: 0x0 0x0 → RX_UNDER interrupt is inactive 0x1 → RX_UNDER interrupt is active</p>	RO	0x0

I2C: IC_RX_TL Register

Offset: 0x38

Description

I2C Receive FIFO Threshold Register

Table 1020. IC_RX_TL Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	RX_TL	<p>Receive FIFO Threshold Level.</p> <p>Controls the level of entries (or above) that triggers the RX_FULL interrupt (bit 2 in IC_RAW_INTR_STAT register). The valid range is 0-255, with the additional restriction that hardware does not allow this value to be set to a value larger than the depth of the buffer. If an attempt is made to do that, the actual value set will be the maximum depth of the buffer. A value of 0 sets the threshold for 1 entry, and a value of 255 sets the threshold for 256 entries.</p>	RW	0x00

I2C: IC_TX_TL Register

Offset: 0x3c

Description

I2C Transmit FIFO Threshold Register

Table 1021. IC_TX_TL Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	TX_TL	<p>Transmit FIFO Threshold Level.</p> <p>Controls the level of entries (or below) that trigger the TX_EMPTY interrupt (bit 4 in IC_RAW_INTR_STAT register). The valid range is 0-255, with the additional restriction that it may not be set to value larger than the depth of the buffer. If an attempt is made to do that, the actual value set will be the maximum depth of the buffer. A value of 0 sets the threshold for 0 entries, and a value of 255 sets the threshold for 255 entries.</p>	RW	0x00

I2C: IC_CLR_INTR Register

Offset: 0x40

Description

Clear Combined and Individual Interrupt Register

Table 1022. IC_CLR_INTR Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
0	CLR_INTR	Read this register to clear the combined interrupt, all individual interrupts, and the IC_TX_ABRT_SOURCE register. This bit does not clear hardware clearable interrupts but software clearable interrupts. Refer to Bit 9 of the IC_TX_ABRT_SOURCE register for an exception to clearing IC_TX_ABRT_SOURCE. Reset value: 0x0	RO	0x0

I2C: IC_CLR_RX_UNDER Register

Offset: 0x44

Description

Clear RX_UNDER Interrupt Register

Table 1023.
IC_CLR_RX_UNDER Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_RX_UNDER	Read this register to clear the RX_UNDER interrupt (bit 0) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_RX_OVER Register

Offset: 0x48

Description

Clear RX_OVER Interrupt Register

Table 1024.
IC_CLR_RX_OVER Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_RX_OVER	Read this register to clear the RX_OVER interrupt (bit 1) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_TX_OVER Register

Offset: 0x4c

Description

Clear TX_OVER Interrupt Register

Table 1025.
IC_CLR_TX_OVER
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_TX_OVER	Read this register to clear the TX_OVER interrupt (bit 3) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_RD_REQ Register

Offset: 0x50

Description

Clear RD_REQ Interrupt Register

Table 1026.
IC_CLR_RD_REQ
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_RD_REQ	Read this register to clear the RD_REQ interrupt (bit 5) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_TX_ABRT Register

Offset: 0x54

Description

Clear TX_ABRT Interrupt Register

Table 1027.
IC_CLR_TX_ABRT
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_TX_ABRT	Read this register to clear the TX_ABRT interrupt (bit 6) of the IC_RAW_INTR_STAT register, and the IC_TX_ABRT_SOURCE register. This also releases the TX FIFO from the flushed/reset state, allowing more writes to the TX FIFO. Refer to Bit 9 of the IC_TX_ABRT_SOURCE register for an exception to clearing IC_TX_ABRT_SOURCE. Reset value: 0x0	RO	0x0

I2C: IC_CLR_RX_DONE Register

Offset: 0x58

Description

Clear RX_DONE Interrupt Register

Table 1028.
IC_CLR_RX_DONE
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_RX_DONE	Read this register to clear the RX_DONE interrupt (bit 7) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_ACTIVITY Register

Offset: 0x5c

Description

Clear ACTIVITY Interrupt Register

Table 1029.
IC_CLR_ACTIVITY
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_ACTIVITY	Reading this register clears the ACTIVITY interrupt if the I2C is not active anymore. If the I2C module is still active on the bus, the ACTIVITY interrupt bit continues to be set. It is automatically cleared by hardware if the module is disabled and if there is no further activity on the bus. The value read from this register to get status of the ACTIVITY interrupt (bit 8) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_STOP_DET Register

Offset: 0x60

Description

Clear STOP_DET Interrupt Register

Table 1030.
IC_CLR_STOP_DET
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_STOP_DET	Read this register to clear the STOP_DET interrupt (bit 9) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_START_DET Register

Offset: 0x64

Description

Clear START_DET Interrupt Register

Table 1031.
IC_CLR_START_DET
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_START_DET	Read this register to clear the START_DET interrupt (bit 10) of the IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_CLR_GEN_CALL Register

Offset: 0x68

Description

Clear GEN_CALL Interrupt Register

Table 1032.
IC_CLR_GEN_CALL
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_GEN_CALL	Read this register to clear the GEN_CALL interrupt (bit 11) of IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_ENABLE Register

Offset: 0x6c

Description

I2C Enable Register

Table 1033.
IC_ENABLE Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	TX_CMD_BLOCK	In Master mode: - 1'b1: Blocks the transmission of data on I2C bus even if Tx FIFO has data to transmit. - 1'b0: The transmission of data starts on I2C bus automatically, as soon as the first data is available in the Tx FIFO. Note: To block the execution of Master commands, set the TX_CMD_BLOCK bit only when Tx FIFO is empty (IC_STATUS[2]==1) and Master is in Idle state (IC_STATUS[5] == 0). Any further commands put in the Tx FIFO are not executed until TX_CMD_BLOCK bit is unset. Reset value: IC_TX_CMD_BLOCK_DEFAULT 0x0 → Tx Command execution not blocked 0x1 → Tx Command execution blocked	RW	0x0

Bits	Name	Description	Type	Reset
1	ABORT	<p>When set, the controller initiates the transfer abort. - 0: ABORT not initiated or ABORT done - 1: ABORT operation in progress The software can abort the I2C transfer in master mode by setting this bit. The software can set this bit only when ENABLE is already set; otherwise, the controller ignores any write to ABORT bit. The software cannot clear the ABORT bit once set. In response to an ABORT, the controller issues a STOP and flushes the Tx FIFO after completing the current transfer, then sets the TX_ABORT interrupt after the abort operation. The ABORT bit is cleared automatically after the abort operation.</p> <p>For a detailed description on how to abort I2C transfers, refer to 'Aborting I2C Transfers'.</p> <p>Reset value: 0x0 0x0 → ABORT operation not in progress 0x1 → ABORT operation in progress</p>	RW	0x0
0	ENABLE	<p>Controls whether the DW_apb_i2c is enabled. - 0: Disables DW_apb_i2c (TX and RX FIFOs are held in an erased state) - 1: Enables DW_apb_i2c Software can disable DW_apb_i2c while it is active. However, it is important that care be taken to ensure that DW_apb_i2c is disabled properly. A recommended procedure is described in 'Disabling DW_apb_i2c'.</p> <p>When DW_apb_i2c is disabled, the following occurs: - The TX FIFO and RX FIFO get flushed. - Status bits in the IC_INTR_STAT register are still active until DW_apb_i2c goes into IDLE state. If the module is transmitting, it stops as well as deletes the contents of the transmit buffer after the current transfer is complete. If the module is receiving, the DW_apb_i2c stops the current transfer at the end of the current byte and does not acknowledge the transfer.</p> <p>In systems with asynchronous pclk and ic_clk when IC_CLK_TYPE parameter set to asynchronous (1), there is a two ic_clk delay when enabling or disabling the DW_apb_i2c. For a detailed description on how to disable DW_apb_i2c, refer to 'Disabling DW_apb_i2c'</p> <p>Reset value: 0x0 0x0 → I2C is disabled 0x1 → I2C is enabled</p>	RW	0x0

I2C: IC_STATUS Register

Offset: 0x70

Description

I2C Status Register

This is a read-only register used to indicate the current transfer status and FIFO status. The status register may be read at any time. None of the bits in this register request an interrupt.

When the I2C is disabled by writing 0 in bit 0 of the IC_ENABLE register: - Bits 1 and 2 are set to 1 - Bits 3 and 10 are set to 0 When the master or slave state machines goes to idle and ic_en=0: - Bits 5 and 6 are set to 0

Table 1034.
IC_STATUS Register

Bits	Name	Description	Type	Reset
31:7	Reserved.	-	-	-
6	SLV_ACTIVITY	Slave FSM Activity Status. When the Slave Finite State Machine (FSM) is not in the IDLE state, this bit is set. - 0: Slave FSM is in IDLE state so the Slave part of DW_apb_i2c is not Active - 1: Slave FSM is not in IDLE state so the Slave part of DW_apb_i2c is Active Reset value: 0x0 0x0 → Slave is idle 0x1 → Slave not idle	RO	0x0
5	MST_ACTIVITY	Master FSM Activity Status. When the Master Finite State Machine (FSM) is not in the IDLE state, this bit is set. - 0: Master FSM is in IDLE state so the Master part of DW_apb_i2c is not Active - 1: Master FSM is not in IDLE state so the Master part of DW_apb_i2c is Active Note: IC_STATUS[0]-that is, ACTIVITY bit-is the OR of SLV_ACTIVITY and MST_ACTIVITY bits. Reset value: 0x0 0x0 → Master is idle 0x1 → Master not idle	RO	0x0
4	RFF	Receive FIFO Completely Full. When the receive FIFO is completely full, this bit is set. When the receive FIFO contains one or more empty location, this bit is cleared. - 0: Receive FIFO is not full - 1: Receive FIFO is full Reset value: 0x0 0x0 → Rx FIFO not full 0x1 → Rx FIFO is full	RO	0x0
3	RFNE	Receive FIFO Not Empty. This bit is set when the receive FIFO contains one or more entries; it is cleared when the receive FIFO is empty. - 0: Receive FIFO is empty - 1: Receive FIFO is not empty Reset value: 0x0 0x0 → Rx FIFO is empty 0x1 → Rx FIFO not empty	RO	0x0
2	TFE	Transmit FIFO Completely Empty. When the transmit FIFO is completely empty, this bit is set. When it contains one or more valid entries, this bit is cleared. This bit field does not request an interrupt. - 0: Transmit FIFO is not empty - 1: Transmit FIFO is empty Reset value: 0x1 0x0 → Tx FIFO not empty 0x1 → Tx FIFO is empty	RO	0x1
1	TFNF	Transmit FIFO Not Full. Set when the transmit FIFO contains one or more empty locations, and is cleared when the FIFO is full. - 0: Transmit FIFO is full - 1: Transmit FIFO is not full Reset value: 0x1 0x0 → Tx FIFO is full 0x1 → Tx FIFO not full	RO	0x1

Bits	Name	Description	Type	Reset
0	ACTIVITY	I2C Activity Status. Reset value: 0x0 0x0 → I2C is idle 0x1 → I2C is active	RO	0x0

I2C: IC_TXFLR Register

Offset: 0x74

Description

I2C Transmit FIFO Level Register This register contains the number of valid data entries in the transmit FIFO buffer. It is cleared whenever: - The I2C is disabled - There is a transmit abort - that is, TX_ABRT bit is set in the IC_RAW_INTR_STAT register - The slave bulk transmit mode is aborted The register increments whenever data is placed into the transmit FIFO and decrements when data is taken from the transmit FIFO.

Table 1035. IC_TXFLR Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4:0	TXFLR	Transmit FIFO Level. Contains the number of valid data entries in the transmit FIFO. Reset value: 0x0	RO	0x00

I2C: IC_RXFLR Register

Offset: 0x78

Description

I2C Receive FIFO Level Register This register contains the number of valid data entries in the receive FIFO buffer. It is cleared whenever: - The I2C is disabled - Whenever there is a transmit abort caused by any of the events tracked in IC_TX_ABRT_SOURCE The register increments whenever data is placed into the receive FIFO and decrements when data is taken from the receive FIFO.

Table 1036. IC_RXFLR Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4:0	RXFLR	Receive FIFO Level. Contains the number of valid data entries in the receive FIFO. Reset value: 0x0	RO	0x00

I2C: IC_SDA_HOLD Register

Offset: 0x7c

Description

I2C SDA Hold Time Length Register

The bits [15:0] of this register are used to control the hold time of SDA during transmit in both slave and master mode (after SCL goes from HIGH to LOW).

The bits [23:16] of this register are used to extend the SDA transition (if any) whenever SCL is HIGH in the receiver in either master or slave mode.

Writes to this register succeed only when IC_ENABLE[0]=0.

The values in this register are in units of ic_clk period. The value programmed in IC_SDA_TX_HOLD must be greater than the minimum hold time in each mode (one cycle in master mode, seven cycles in slave mode) for the value to be implemented.

The programmed SDA hold time during transmit (IC_SDA_TX_HOLD) cannot exceed at any time the duration of the low part of scl. Therefore the programmed value cannot be larger than N_SCL_LOW-2, where N_SCL_LOW is the duration of the low part of the scl period measured in ic_clk cycles.

Table 1037.
IC_SDA_HOLD
Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	IC_SDA_RX_HOLD	Sets the required SDA hold time in units of ic_clk period, when DW_apb_i2c acts as a receiver. Reset value: IC_DEFAULT_SDA_HOLD[23:16].	RW	0x00
15:0	IC_SDA_TX_HOLD	Sets the required SDA hold time in units of ic_clk period, when DW_apb_i2c acts as a transmitter. Reset value: IC_DEFAULT_SDA_HOLD[15:0].	RW	0x0001

I2C: IC_TX_ABRT_SOURCE Register

Offset: 0x80

Description

I2C Transmit Abort Source Register

This register has 32 bits that indicate the source of the TX_ABRT bit. Except for Bit 9, this register is cleared whenever the IC_CLR_TX_ABRT register or the IC_CLR_INTR register is read. To clear Bit 9, the source of the ABRT_SBYTE_NORSTRT must be fixed first; RESTART must be enabled (IC_CON[5]=1), the SPECIAL bit must be cleared (IC_TAR[11]), or the GC_OR_START bit must be cleared (IC_TAR[10]).

Once the source of the ABRT_SBYTE_NORSTRT is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the ABRT_SBYTE_NORSTRT is not fixed before attempting to clear this bit, Bit 9 clears for one cycle and is then re-asserted.

Table 1038.
IC_TX_ABRT_SOURCE
Register

Bits	Name	Description	Type	Reset
31:23	TX_FLUSH_CNT	This field indicates the number of Tx FIFO Data Commands which are flushed due to TX_ABRT interrupt. It is cleared whenever I2C is disabled. Reset value: 0x0 Role of DW_apb_i2c: Master-Transmitter or Slave-Transmitter	RO	0x000
22:17	Reserved.	-	-	-
16	ABRT_USER_ABRT	This is a master-mode-only bit. Master has detected the transfer abort (IC_ENABLE[1]) Reset value: 0x0 Role of DW_apb_i2c: Master-Transmitter 0x0 → Transfer abort detected by master- scenario not present 0x1 → Transfer abort detected by master	RO	0x0

Bits	Name	Description	Type	Reset
15	ABRT_SLVRD_INT_X	<p>1: When the processor side responds to a slave mode request for data to be transmitted to a remote master and user writes a 1 in CMD (bit 8) of IC_DATA_CMD register.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Slave-Transmitter</p> <p>0x0 → Slave trying to transmit to remote master in read mode- scenario not present</p> <p>0x1 → Slave trying to transmit to remote master in read mode</p>	RO	0x0
14	ABRT_SLV_ARBL_OST	<p>This field indicates that a Slave has lost the bus while transmitting data to a remote master.</p> <p>IC_TX_ABRT_SOURCE[12] is set at the same time. Note: Even though the slave never 'owns' the bus, something could go wrong on the bus. This is a fail safe check. For instance, during a data transmission at the low-to-high transition of SCL, if what is on the data bus is not what is supposed to be transmitted, then DW_apb_i2c no longer own the bus.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Slave-Transmitter</p> <p>0x0 → Slave lost arbitration to remote master- scenario not present</p> <p>0x1 → Slave lost arbitration to remote master</p>	RO	0x0
13	ABRT_SLVFLUSH_TXFIFO	<p>This field specifies that the Slave has received a read command and some data exists in the TX FIFO, so the slave issues a TX_ABRT interrupt to flush old data in TX FIFO.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Slave-Transmitter</p> <p>0x0 → Slave flushes existing data in TX-FIFO upon getting read command- scenario not present</p> <p>0x1 → Slave flushes existing data in TX-FIFO upon getting read command</p>	RO	0x0
12	ARB_LOST	<p>This field specifies that the Master has lost arbitration, or if IC_TX_ABRT_SOURCE[14] is also set, then the slave transmitter has lost arbitration.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter or Slave-Transmitter</p> <p>0x0 → Master or Slave-Transmitter lost arbitration- scenario not present</p> <p>0x1 → Master or Slave-Transmitter lost arbitration</p>	RO	0x0

Bits	Name	Description	Type	Reset
11	ABRT_MASTER_DIS	<p>This field indicates that the User tries to initiate a Master operation with the Master mode disabled.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter or Master-Receiver</p> <p>0x0 → User initiating master operation when MASTER disabled- scenario not present</p> <p>0x1 → User initiating master operation when MASTER disabled</p>	RO	0x0
10	ABRT_10B_RD_NORSTRT	<p>This field indicates that the restart is disabled (IC_RESTART_EN bit (IC_CON[5]) =0) and the master sends a read command in 10-bit addressing mode.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Receiver</p> <p>0x0 → Master not trying to read in 10Bit addressing mode when RESTART disabled</p> <p>0x1 → Master trying to read in 10Bit addressing mode when RESTART disabled</p>	RO	0x0
9	ABRT_SBYTE_NORSTRT	<p>To clear Bit 9, the source of the ABRT_SBYTE_NORSTRT must be fixed first; restart must be enabled (IC_CON[5]=1), the SPECIAL bit must be cleared (IC_TAR[11]), or the GC_OR_START bit must be cleared (IC_TAR[10]). Once the source of the ABRT_SBYTE_NORSTRT is fixed, then this bit can be cleared in the same manner as other bits in this register. If the source of the ABRT_SBYTE_NORSTRT is not fixed before attempting to clear this bit, bit 9 clears for one cycle and then gets reasserted. When this field is set to 1, the restart is disabled (IC_RESTART_EN bit (IC_CON[5]) =0) and the user is trying to send a START Byte.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master</p> <p>0x0 → User trying to send START byte when RESTART disabled- scenario not present</p> <p>0x1 → User trying to send START byte when RESTART disabled</p>	RO	0x0

Bits	Name	Description	Type	Reset
8	ABRT_HS_NORST_RT	<p>This field indicates that the restart is disabled (IC_RESTART_EN bit (IC_CON[5]) =0) and the user is trying to use the master to transfer data in High Speed mode.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter or Master-Receiver</p> <p>0x0 → User trying to switch Master to HS mode when RESTART disabled- scenario not present</p> <p>0x1 → User trying to switch Master to HS mode when RESTART disabled</p>	RO	0x0
7	ABRT_SBYTE_ACKDET	<p>This field indicates that the Master has sent a START Byte and the START Byte was acknowledged (wrong behavior).</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master</p> <p>0x0 → ACK detected for START byte- scenario not present</p> <p>0x1 → ACK detected for START byte</p>	RO	0x0
6	ABRT_HS_ACKDET	<p>This field indicates that the Master is in High Speed mode and the High Speed Master code was acknowledged (wrong behavior).</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master</p> <p>0x0 → HS Master code ACKed in HS Mode- scenario not present</p> <p>0x1 → HS Master code ACKed in HS Mode</p>	RO	0x0
5	ABRT_GCALL_READ	<p>This field indicates that DW_apb_i2c in the master mode has sent a General Call but the user programmed the byte following the General Call to be a read from the bus (IC_DATA_CMD[9] is set to 1).</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter</p> <p>0x0 → GCALL is followed by read from bus-scenario not present</p> <p>0x1 → GCALL is followed by read from bus</p>	RO	0x0
4	ABRT_GCALL_NOACK	<p>This field indicates that DW_apb_i2c in master mode has sent a General Call and no slave on the bus acknowledged the General Call.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter</p> <p>0x0 → GCALL not ACKed by any slave-scenario not present</p> <p>0x1 → GCALL not ACKed by any slave</p>	RO	0x0

Bits	Name	Description	Type	Reset
3	ABRT_TXDATA_NOACK	<p>This field indicates the master-mode only bit. When the master receives an acknowledgement for the address, but when it sends data byte(s) following the address, it did not receive an acknowledgement from the remote slave(s).</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter 0x0 → Transmitted data non-ACKed by addressed slave-scenario not present 0x1 → Transmitted data not ACKed by addressed slave</p>	RO	0x0
2	ABRT_10ADDR2_NOACK	<p>This field indicates that the Master is in 10-bit address mode and that the second address byte of the 10-bit address was not acknowledged by any slave.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter or Master-Receiver 0x0 → This abort is not generated 0x1 → Byte 2 of 10Bit Address not ACKed by any slave</p>	RO	0x0
1	ABRT_10ADDR1_NOACK	<p>This field indicates that the Master is in 10-bit address mode and the first 10-bit address byte was not acknowledged by any slave.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter or Master-Receiver 0x0 → This abort is not generated 0x1 → Byte 1 of 10Bit Address not ACKed by any slave</p>	RO	0x0
0	ABRT_7B_ADDR_NOACK	<p>This field indicates that the Master is in 7-bit addressing mode and the address sent was not acknowledged by any slave.</p> <p>Reset value: 0x0</p> <p>Role of DW_apb_i2c: Master-Transmitter or Master-Receiver 0x0 → This abort is not generated 0x1 → This abort is generated because of NOACK for 7-bit address</p>	RO	0x0

I2C: IC_SLV_DATA_NACK_ONLY Register

Offset: 0x84

Description

Generate Slave Data NACK Register

The register is used to generate a NACK for the data part of a transfer when DW_apb_i2c is acting as a slave-receiver. This register only exists when the IC_SLV_DATA_NACK_ONLY parameter is set to 1. When this parameter disabled, this register does not exist and writing to the register's address has no effect.

A write can occur on this register if both of the following conditions are met: - DW_apb_i2c is disabled (IC_ENABLE[0] = 0) - Slave part is inactive (IC_STATUS[6] = 0) Note: The IC_STATUS[6] is a register read-back location for the internal slv_activity signal; the user should poll this before writing the ic_slv_data_nack_only bit.

Table 1039.
IC_SLV_DATA_NACK_ONLY Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	NACK	Generate NACK. This NACK generation only occurs when DW_apb_i2c is a slave-receiver. If this register is set to a value of 1, it can only generate a NACK after a data byte is received; hence, the data transfer is aborted and the data received is not pushed to the receive buffer. When the register is set to a value of 0, it generates NACK/ACK, depending on normal criteria. - 1: generate NACK after data byte received - 0: generate NACK/ACK normally Reset value: 0x0 0x0 → Slave receiver generates NACK normally 0x1 → Slave receiver generates NACK upon data reception only	RW	0x0

I2C: IC_DMA_CR Register

Offset: 0x88

Description

DMA Control Register

The register is used to enable the DMA Controller interface operation. There is a separate bit for transmit and receive. This can be programmed regardless of the state of IC_ENABLE.

Table 1040.
IC_DMA_CR Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	TDMAE	Transmit DMA Enable. This bit enables/disables the transmit FIFO DMA channel. Reset value: 0x0 0x0 → transmit FIFO DMA channel disabled 0x1 → Transmit FIFO DMA channel enabled	RW	0x0
0	RDMAE	Receive DMA Enable. This bit enables/disables the receive FIFO DMA channel. Reset value: 0x0 0x0 → Receive FIFO DMA channel disabled 0x1 → Receive FIFO DMA channel enabled	RW	0x0

I2C: IC_DMA_TDLR Register

Offset: 0x8c

Description

DMA Transmit Data Level Register

Table 1041.
IC_DMA_TDLR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3:0	DMATDL	Transmit Data Level. This bit field controls the level at which a DMA request is made by the transmit logic. It is equal to the watermark level; that is, the <code>dma_tx_req</code> signal is generated when the number of valid data entries in the transmit FIFO is equal to or below this field value, and <code>TDMAE</code> = 1. Reset value: 0x0	RW	0x0

I2C: IC_DMA_RDLR Register

Offset: 0x90

Description

I2C Receive Data Level Register

Table 1042.
IC_DMA_RDLR
Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:0	DMARDL	Receive Data Level. This bit field controls the level at which a DMA request is made by the receive logic. The watermark level = <code>DMARDL</code> +1; that is, <code>dma_rx_req</code> is generated when the number of valid data entries in the receive FIFO is equal to or more than this field value + 1, and <code>RDMAE</code> =1. For instance, when <code>DMARDL</code> is 0, then <code>dma_rx_req</code> is asserted when 1 or more data entries are present in the receive FIFO. Reset value: 0x0	RW	0x0

I2C: IC_SDA_SETUP Register

Offset: 0x94

Description

I2C SDA Setup Register

This register controls the amount of time delay (in terms of number of `ic_clk` clock periods) introduced in the rising edge of `SCL` - relative to `SDA` changing - when `DW_apb_i2c` services a read request in a slave-transmitter operation. The relevant I2C requirement is `tSU:DAT` (note 4) as detailed in the I2C Bus Specification. This register must be programmed with a value equal to or greater than 2.

Writes to this register succeed only when `IC_ENABLE[0]` = 0.

Note: The length of setup time is calculated using $[(IC_SDA_SETUP - 1) * (ic_clk_period)]$, so if the user requires 10 `ic_clk` periods of setup time, they should program a value of 11. The `IC_SDA_SETUP` register is only used by the `DW_apb_i2c` when operating as a slave transmitter.

Table 1043.
IC_SDA_SETUP
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	SDA_SETUP	SDA Setup. It is recommended that if the required delay is 1000ns, then for an ic_clk frequency of 10 MHz, IC_SDA_SETUP should be programmed to a value of 11. IC_SDA_SETUP must be programmed with a minimum value of 2.	RW	0x64

I2C: IC_ACK_GENERAL_CALL Register

Offset: 0x98

Description

I2C ACK General Call Register

The register controls whether DW_apb_i2c responds with a ACK or NACK when it receives an I2C General Call address.

This register is applicable only when the DW_apb_i2c is in slave mode.

Table 1044.
IC_ACK_GENERAL_CA
LL Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	ACK_GEN_CALL	ACK General Call. When set to 1, DW_apb_i2c responds with a ACK (by asserting ic_data_oe) when it receives a General Call. Otherwise, DW_apb_i2c responds with a NACK (by negating ic_data_oe). 0x0 → Generate NACK for a General Call 0x1 → Generate ACK for a General Call	RW	0x1

I2C: IC_ENABLE_STATUS Register

Offset: 0x9c

Description

I2C Enable Status Register

The register is used to report the DW_apb_i2c hardware status when the IC_ENABLE[0] register is set from 1 to 0; that is, when DW_apb_i2c is disabled.

If IC_ENABLE[0] has been set to 1, bits 2:1 are forced to 0, and bit 0 is forced to 1.

If IC_ENABLE[0] has been set to 0, bits 2:1 is only be valid as soon as bit 0 is read as '0'.

Note: When IC_ENABLE[0] has been set to 0, a delay occurs for bit 0 to be read as 0 because disabling the DW_apb_i2c depends on I2C bus activities.

Table 1045.
IC_ENABLE_STATUS
Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
2	SLV_RX_DATA_LOST	<p>Slave Received Data Lost. This bit indicates if a Slave-Receiver operation has been aborted with at least one data byte received from an I2C transfer due to the setting bit 0 of IC_ENABLE from 1 to 0. When read as 1, DW_apb_i2c is deemed to have been actively engaged in an aborted I2C transfer (with matching address) and the data phase of the I2C transfer has been entered, even though a data byte has been responded with a NACK.</p> <p>Note: If the remote I2C master terminates the transfer with a STOP condition before the DW_apb_i2c has a chance to NACK a transfer, and IC_ENABLE[0] has been set to 0, then this bit is also set to 1.</p> <p>When read as 0, DW_apb_i2c is deemed to have been disabled without being actively involved in the data phase of a Slave-Receiver transfer.</p> <p>Note: The CPU can safely read this bit when IC_EN (bit 0) is read as 0.</p> <p>Reset value: 0x0 0x0 → Slave RX Data is not lost 0x1 → Slave RX Data is lost</p>	RO	0x0

Bits	Name	Description	Type	Reset
1	SLV_DISABLED_W HILE_BUSY	<p>Slave Disabled While Busy (Transmit, Receive). This bit indicates if a potential or active Slave operation has been aborted due to the setting bit 0 of the IC_ENABLE register from 1 to 0. This bit is set when the CPU writes a 0 to the IC_ENABLE register while:</p> <p>(a) DW_apb_i2c is receiving the address byte of the Slave-Transmitter operation from a remote master;</p> <p>OR,</p> <p>(b) address and data bytes of the Slave-Receiver operation from a remote master.</p> <p>When read as 1, DW_apb_i2c is deemed to have forced a NACK during any part of an I2C transfer, irrespective of whether the I2C address matches the slave address set in DW_apb_i2c (IC_SAR register) OR if the transfer is completed before IC_ENABLE is set to 0 but has not taken effect.</p> <p>Note: If the remote I2C master terminates the transfer with a STOP condition before the DW_apb_i2c has a chance to NACK a transfer, and IC_ENABLE[0] has been set to 0, then this bit will also be set to 1.</p> <p>When read as 0, DW_apb_i2c is deemed to have been disabled when there is master activity, or when the I2C bus is idle.</p> <p>Note: The CPU can safely read this bit when IC_EN (bit 0) is read as 0.</p> <p>Reset value: 0x0 0x0 → Slave is disabled when it is idle 0x1 → Slave is disabled when it is active</p>	RO	0x0
0	IC_EN	<p>ic_en Status. This bit always reflects the value driven on the output port ic_en. - When read as 1, DW_apb_i2c is deemed to be in an enabled state. - When read as 0, DW_apb_i2c is deemed completely inactive. Note: The CPU can safely read this bit anytime. When this bit is read as 0, the CPU can safely read SLV_RX_DATA_LOST (bit 2) and SLV_DISABLED_WHILE_BUSY (bit 1).</p> <p>Reset value: 0x0 0x0 → I2C disabled 0x1 → I2C enabled</p>	RO	0x0

I2C: IC_FS_SPKLEN Register

Offset: 0xa0

Description

I2C SS, FS or FM+ spike suppression limit

This register is used to store the duration, measured in ic_clk cycles, of the longest spike that is filtered out by the spike suppression logic when the component is operating in SS, FS or FM+ modes. The relevant I2C requirement is tSP (table 4) as detailed in the I2C Bus Specification. This register must be programmed with a minimum value of 1.

Table 1046.
IC_FS_SPKLEN
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	IC_FS_SPKLEN	This register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration, measured in ic_clk cycles, of the longest spike in the SCL or SDA lines that will be filtered out by the spike suppression logic. This register can be written only when the I2C interface is disabled which corresponds to the IC_ENABLE[0] register being set to 0. Writes at other times have no effect. The minimum valid value is 1; hardware prevents values less than this being written, and if attempted results in 1 being set. or more information, refer to 'Spike Suppression'.	RW	0x07

I2C: IC_CLR_RESTART_DET Register

Offset: 0xa8

Description

Clear RESTART_DET Interrupt Register

Table 1047.
IC_CLR_RESTART_DET
Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLR_RESTART_DET	Read this register to clear the RESTART_DET interrupt (bit 12) of IC_RAW_INTR_STAT register. Reset value: 0x0	RO	0x0

I2C: IC_COMP_PARAM_1 Register

Offset: 0xf4

Description

Component Parameter Register 1

Note This register is not implemented and therefore reads as 0. If it was implemented it would be a constant read-only register that contains encoded information about the component's parameter settings. Fields shown below are the settings for those parameters

Table 1048.
IC_COMP_PARAM_1
Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	TX_BUFFER_DEPT_H	TX Buffer Depth = 16	RO	0x00
15:8	RX_BUFFER_DEPT_H	RX Buffer Depth = 16	RO	0x00
7	ADD_ENCODED_PARAMS	Encoded parameters not visible	RO	0x0
6	HAS_DMA	DMA handshaking signals are enabled	RO	0x0
5	INTR_IO	COMBINED Interrupt outputs	RO	0x0

Bits	Name	Description	Type	Reset
4	HC_COUNT_VALUES	Programmable count values for each mode.	RO	0x0
3:2	MAX_SPEED_MODE	MAX SPEED MODE = FAST MODE	RO	0x0
1:0	APB_DATA_WIDTH	APB data bus width is 32 bits	RO	0x0

I2C: IC_COMP_VERSION Register

Offset: 0xf8

Description

I2C Component Version Register

Table 1049.
IC_COMP_VERSION
Register

Bits	Name	Description	Type	Reset
31:0	IC_COMP_VERSION		RO	0x3230312a

I2C: IC_COMP_TYPE Register

Offset: 0xfc

Description

I2C Component Type Register

Table 1050.
IC_COMP_TYPE
Register

Bits	Name	Description	Type	Reset
31:0	IC_COMP_TYPE	Designware Component Type number = 0x44_57_01_40. This assigned unique hex value is constant and is derived from the two ASCII letters 'DW' followed by a 16-bit unsigned number.	RO	0x44570140

12.3. SPI

Arm Documentation

Excerpted from the [ARM PrimeCell Synchronous Serial Port \(PL022\) Technical Reference Manual](#). Used with permission.

RP2350 has two identical SPI controllers, both based on an Arm Primecell Synchronous Serial Port (SSP) (PL022) (Revision r1p4). This is distinct from the QSPI memory interface covered in [Section 12.14](#).

Each controller supports the following features:

- Master or Slave modes
 - Motorola SPI-compatible interface
 - Texas Instruments synchronous serial interface
 - National Semiconductor Microwire interface
- 8-location Tx and Rx FIFOs
- Interrupt generation to service FIFOs or indicate error conditions

- Can be driven from DMA
- Programmable clock rate
- Programmable data size 4-16 bits

Each controller can be connected to a number of GPIO pins as defined in the GPIO muxing [Table 646](#) in [Section 9.4](#). Connections to the GPIO muxing use a prefix that contains the SPI instance name `spi0_` or `spi1_`, and include the following:

- clock `sclk` (connects to `SSPCLKOUT` in the following sections when the controller is operating in master mode, or `SSPCLKIN` when in slave mode)
- active low chip select or frame sync `ss_n` (referred to as `SSPFSSOUT` in the following sections)
- transmit data `tx` (referred to as `SSPTXD` in the following sections; `nSSPOE` is not connected to the `tx` pad, so the SPI controller does not tristate output data)
- receive data `rd` (referred to as `SSPRXD` in the following sections)

The SPI Tx pin function is wired to always assert the pad output enable, and is not driven from `nSSPOE`. When multiple SPI slaves share a bus, software must switch the output enable. This could be done by toggling the `oeover` field of the relevant `iobank0.ctrl` register, or by switching GPIO function.

The SPI uses `clk_peri` as its reference clock for SPI timing, and is referred to as `SSPCLK` in the following sections. `clk_sys` is used as the bus clock, and is referred to as PCLK in the following sections (also see [Figure 30](#)).

12.3.1. Overview

The PrimeCell SSP is a master or slave interface for synchronous serial communication with peripheral devices that have Motorola SPI, National Semiconductor Microwire, or Texas Instruments synchronous serial interfaces.

The PrimeCell SSP performs serial-to-parallel conversion on data received from a peripheral device. The CPU accesses data, control, and status information through the AMBA APB interface. The transmit and receive paths are buffered with internal FIFO memories, enabling up to eight 16-bit values to be stored independently in both transmit and receive modes. Serial data transmits on `SSPTXD` and is received on `SSPRXD`.

The PrimeCell SSP includes a programmable bit rate clock divider and prescaler to generate the serial output clock, `SSPCLKOUT`, from the input clock, `SSPCLK`. Bit rates are supported to 2MHz and higher, subject to choice of frequency for `SSPCLK`, and the maximum bit rate is determined by peripheral devices.

You can use the control registers `SSPCR0` and `SSPCR1` to program the PrimeCell SSP operating mode, frame format, and size.

The following individually maskable interrupts are generated:

- `SSPTXINTR` requests servicing of the transmit buffer
- `SSPRXINTR` requests servicing of the receive buffer
- `SSPRORINTR` indicates an overrun condition in the receive FIFO
- `SSPRTINTR` indicates that a timeout period expired while data was present in the receive FIFO.

A single combined interrupt is asserted if any of the individual interrupts are asserted and unmasked. This interrupt is connected to the processor interrupt controllers in RP2350.

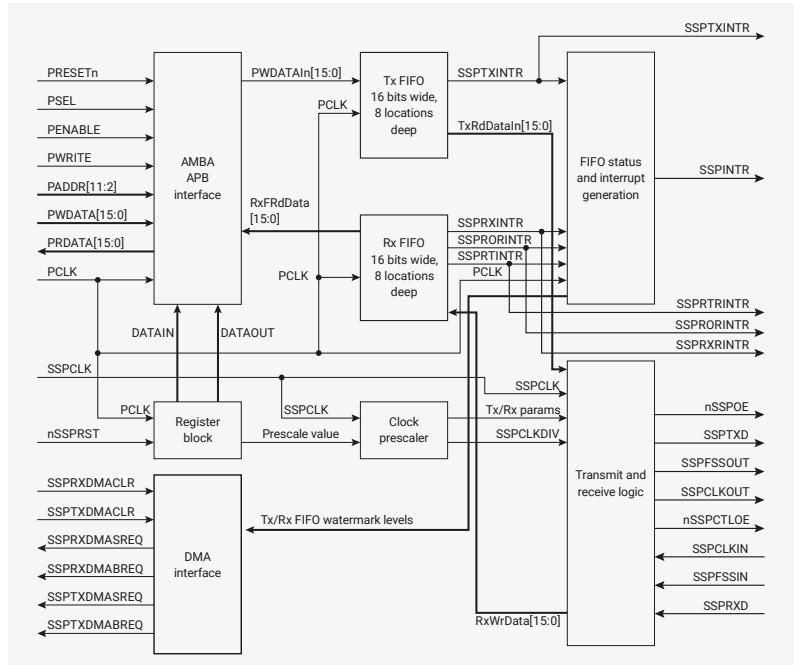
In addition to the above interrupts, a set of DMA signals are provided for interfacing with a DMA controller.

Depending on the operating mode selected, the `SSPFSSOUT` output operates as:

- an active-HIGH frame synchronization output for Texas Instruments synchronous serial frame format
- an active-LOW slave select for SPI and Microwire.

12.3.2. Functional Description

Figure 69. PrimeCell
SSP block diagram.
For clarity, does not
show the test logic.



12.3.2.1. AMBA APB interface

The AMBA APB interface generates read and write decodes for accesses to status and control registers, and transmit and receive FIFO memories.

12.3.2.2. Register block

The register block stores data written, or to be read, across the AMBA APB interface.

12.3.2.3. Clock prescaler

When configured as a master, an internal prescaler, comprising two free-running reloadable serially linked counters, provides the serial output clock **SSPCLKOUT**.

You can program the clock prescaler, using the **SSPCPSR** register, to divide **SSPCLK** by a factor of 2-254 in steps of two. By not utilizing the least significant bit of the **SSPCPSR** register, division by an odd number is not possible which ensures that a symmetrical, equal mark space ratio, clock is generated. See **SSPCPSR**.

The output of the prescaler is divided again by a factor of 1-256, by programming the **SSPCR0** control register, to give the final master output clock **SSPCLKOUT**.

NOTE

The PCLK and SSPCLK clock inputs in [Figure 69](#) are connected to the `clk_sys` and `clk_peri` system-level clock nets on RP2350, respectively. By default, `clk_peri` attaches directly to the system clock. However, you can detach it to maintain constant SPI frequency if the system clock is varied dynamically. See [Figure 30](#) for an overview of the RP2350 clock architecture.

12.3.2.4. Transmit FIFO

The common transmit (Tx) FIFO is a 16-bit wide, 8-location deep memory buffer. CPU data written across the AMBA APB interface is stored in the buffer until read out by the transmit logic.

When configured as a master or a slave, parallel data is written into the transmit FIFO prior to serial conversion, and transmission to the attached slave or master respectively, through the `SSPTXD` pin.

12.3.2.5. Receive FIFO

The common receive (Rx) FIFO is a 16-bit wide, 8-location deep memory buffer. Received data from the serial interface is stored in the buffer until read out by the CPU across the AMBA APB interface.

When configured as a master or slave, serial data received through the `SSPRXD` pin is registered prior to parallel loading into the attached slave or master receive FIFO respectively.

12.3.2.6. Transmit and receive logic

When configured as a master, the clock for the attached slaves is derived from a divided-down version of `SSPCLK` through the previously described prescaler operations. The master transmit logic successively reads a value from its transmit FIFO and performs parallel to serial conversion on it. Then, the serial data stream and frame control signal, synchronized to `SSPCLKOUT`, outputs through the `SSPTXD` pin to the attached slaves. The master receive logic performs serial to parallel conversion on the incoming synchronous `SSPRXD` data stream, extracting and storing values into its receive FIFO for subsequent reading through the APB interface.

When configured as a slave, the `SSPCLKIN` clock is provided by an attached master and used to time transmission and reception sequences. The slave transmit logic, under control of the master clock, successively:

1. Reads a value from its transmit FIFO.
2. Performs parallel to serial conversion.
3. Outputs the serial data stream and frame control signal through the slave `SSPTXD` pin.

The slave receive logic performs serial to parallel conversion on the incoming `SSPRXD` data stream, extracting and storing values into its receive FIFO, for subsequent reading through the APB interface.

12.3.2.7. Interrupt generation logic

The PrimeCell SSP generates four individual maskable, active-HIGH interrupts. A combined interrupt output is generated as an OR function of the individual interrupt requests.

The transmit and receive dynamic data-flow interrupts, `SSPTXINTR` and `SSPRXINTR`, are separated from the status interrupts so that data can be read or written in response to the FIFO trigger levels.

12.3.2.8. DMA interface

The PrimeCell SSP provides an interface to connect to a DMA controller, see [Section 12.3.3.16](#).

12.3.2.9. Synchronizing registers and logic

The PrimeCell SSP supports both asynchronous and synchronous operation of the clocks, **PCLK** and **SSPCLK**. Synchronization registers and handshaking logic have been implemented, and are active at all times. Synchronization of control signals is performed on both directions of data flow, that is:

- from the **PCLK** to the **SSPCLK** domain
- from the **SSPCLK** to the **PCLK** domain.

12.3.3. Operation

12.3.3.1. Interface reset

The PrimeCell SSP is reset by the global reset signal, **PRESETn**, and a block-specific reset signal, **nSSPRST**. The device reset controller asserts **nSSPRST** asynchronously and negates it synchronously to **SSPCLK**.

12.3.3.2. Configuring the SSP

Following reset, the PrimeCell SSP logic is disabled and must be configured when in this state. It is necessary to program control registers **SSPCR0** and **SSPCR1** to configure the peripheral as a master or slave operating under one of the following protocols:

- Motorola SPI
- Texas Instruments SSI
- National Semiconductor

The bit rate, derived from the external **SSPCLK**, requires the programming of the clock prescale register **SSPCPSR**.

12.3.3.3. Enable PrimeCell SSP operation

You can either prime the transmit FIFO, by writing up to eight 16-bit values when the PrimeCell SSP is disabled, or permit the transmit FIFO service request to interrupt the CPU. Once enabled, transmission or reception of data begins on the transmit, **SSPTXD**, and receive, **SSPRXD**, pins.

12.3.3.4. Clock ratios

There is a constraint on the ratio of the frequencies of **PCLK** to **SSPCLK**. The frequency of **SSPCLK** must be less than or equal to that of **PCLK**. This ensures that control signals from the **SSPCLK** domain to the **PCLK** domain are guaranteed to get synchronized before one frame duration:

$$F_{SSPCLK} \leq F_{PCLK}.$$

In the slave mode of operation, the **SSPCLKIN** signal from the external master is double-synchronized and then delayed to detect an edge. It takes three **SSPCLKs** to detect an edge on **SSPCLKIN**. **SSPTXD** has less setup time to the falling edge of **SSPCLKIN** on which the master is sampling the line.

The setup and hold times on **SSPRXD**, with reference to **SSPCLKIN**, must be more conservative to ensure that it is at the right value when the actual sampling occurs within the **SSPMS**. To ensure correct device operation, **SSPCLK** must be at least 12

times faster than the maximum expected frequency of **SSPCLKIN**.

The frequency selected for **SSPCLK** must accommodate the desired range of bit clock rates. The ratio of minimum **SSPCLK** frequency to **SSPCLKOUT** maximum frequency in the case of the slave mode is 12, and for the master mode, it is two.

For example, at the maximum **SSPCLK** (`clk_peri`) frequency on RP2350 of 150MHz, the maximum peak bit rate in master mode is 70.5Mb/s. This is achieved with the **SSPCPSR** register programmed with a value of 2, and the **SCR[7:0]** field in the **SSPCR0** register programmed with a value of 0.

In slave mode, the same maximum **SSPCLK** frequency of 150MHz can achieve a peak bit rate of $150 / 12 = 12.5$ Mb/s. The **SSPCPSR** register can be programmed with a value of 12, and the **SCR[7:0]** field in the **SSPCR0** register can be programmed with a value of 0. Similarly, the ratio of **SSPCLK** maximum frequency to **SSPCLKOUT** minimum frequency is 254×256 .

The minimum frequency of **SSPCLK** is governed by the following inequalities, both of which must be satisfied:

$$F_{SSPCLK}(\min) \geq 2 \times F_{SSPCLKOUT}(\max), \text{ for master mode}$$

$$F_{SSPCLK}(\min) \geq 12 \times F_{SSPCLKIN}(\max), \text{ for slave mode.}$$

The maximum frequency of **SSPCLK** is governed by the following inequalities, both of which must be satisfied:

$$F_{SSPCLK}(\max) \leq 254 \times 256 \times F_{SSPCLKOUT}(\min), \text{ for master mode}$$

$$F_{SSPCLK}(\max) \leq 254 \times 256 \times F_{SSPCLKIN}(\min), \text{ for slave mode.}$$

12.3.3.5. Programming the **SSPCR0** Control Register

The **SSPCR0** register is used to:

- program the serial clock rate
- select one of the three protocols
- select the data word size, where applicable.

The Serial Clock Rate (SCR) value, in conjunction with the **SSPCPSR** clock prescale divisor value, **CPSDVS**, is used to derive the PrimeCell SSP transmit and receive bit rate from the external **SSPCLK**.

The frame format is programmed through the **FRF** bits, and the data word size through the **DSS** bits.

Bit phase and polarity, applicable to Motorola SPI format only, are programmed through the **SPH** and **SPO** bits.

12.3.3.6. Programming the **SSPCR1** Control Register

The **SSPCR1** register is used to:

- select master or slave mode
- enable a loop back test feature
- enable the PrimeCell SSP peripheral.

To configure the PrimeCell SSP as a master, clear the **SSPCR1** register master or slave selection bit, **MS**, to 0. This is the default value on reset.

Setting the **SSPCR1** register **MS** bit to 1 configures the PrimeCell SSP as a slave. When configured as a slave, use the **SSPCR1** slave mode **SSPTXD** output disable bit (**SOD**) to enable or disable of the PrimeCell SSP **SSPTXD** signal. You can use this in some multi-slave environments where masters might parallel broadcast.

To enable the PrimeCell SSP, set the Synchronous Serial Port Enable (**SSE**) bit to 1.

12.3.3.6.1. Bit rate generation

The serial bit rate is derived by dividing down the input clock, **SSPCLK**. The clock is first divided by an even prescale value **CPSDVSR** in the range 2-254, and is programmed in **SSPCPSR**. The clock is divided again by a value in the range 1-256, that is $1 + SCR$, where **SCR** is the value programmed in **SSPCR0**.

The following equation defines the frequency of the output signal bit clock, **SSPCLKOUT**:

$$F_{SSPCLKOUT} = \frac{F_{SSPCLK}}{CPSDVSR \times (1 + SCR)}$$

For example, if **SSPCLK** is 125MHz, and **CPSDVSR** = 2, then **SSPCLKOUT** has a frequency range from 244kHz - 62.5MHz.

12.3.3.7. Frame format

Each data frame is between 4-16 bits long, depending on the size of data programmed, and is transmitted starting with the MSB. You can select the following basic frame types:

- Texas Instruments synchronous serial
- Motorola SPI
- National Semiconductor Microwire.

For all formats, the serial clock, **SSPCLKOUT**, is held inactive while the PrimeCell SSP is idle, and transitions at the programmed frequency only during active transmission or reception of data. The idle state of **SSPCLKOUT** is utilized to provide a receive timeout indication that occurs when the receive FIFO still contains data after a timeout period.

For Motorola SPI and National Semiconductor Microwire frame formats, the serial frame, **SSPFSSOUT**, pin is active-LOW, and is asserted, pulled-down, during the entire transmission of the frame.

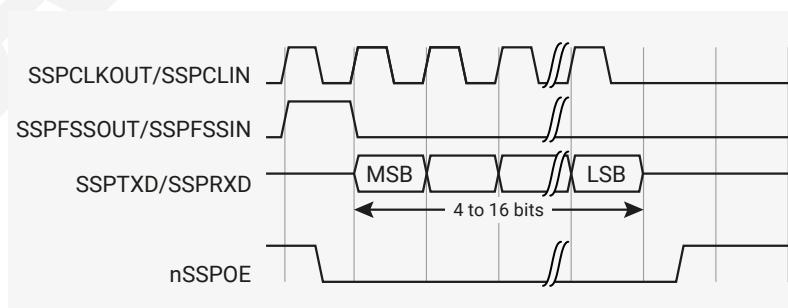
For Texas Instruments synchronous serial frame format, the **SSPFSSOUT** pin is pulsed for one serial clock period, starting at its rising edge, prior to the transmission of each frame. For this frame format, both the PrimeCell SSP and the off-chip slave device drive their output data on the rising edge of **SSPCLKOUT**, and latch data from the other device on the falling edge.

Unlike the full-duplex transmission of the other two frame formats, the National Semiconductor Microwire format uses a special master-slave messaging technique that operates at half-duplex. In this mode, when a frame begins, an 8-bit control message is transmitted to the off-chip slave. During this transmit, the SSS receives no incoming data. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the requested data. The returned data can be 4-16 bits in length, making the total frame length in the range 13-25 bits.

12.3.3.8. Texas Instruments synchronous serial frame format

Figure 70 shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

Figure 70. Texas Instruments synchronous serial frame format, single transfer



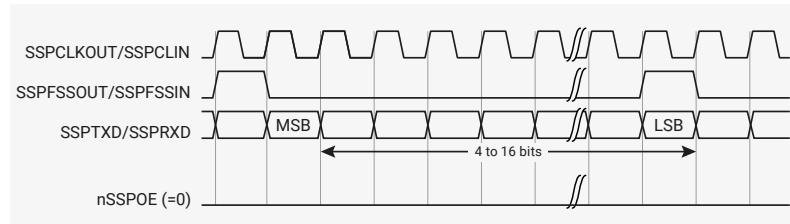
In this mode, **SSPCLKOUT** and **SSPFSSOUT** are forced LOW, and the transmit data line, **SSPTXD** is tristated whenever the PrimeCell SSP is idle. When the bottom entry of the transmit FIFO contains data, **SSPFSSOUT** is pulsed HIGH for one **SSPCLKOUT** period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the

transmit logic. On the next rising edge of **SSPCLKOUT**, the MSB of the 4-bit to 16-bit data frame is shifted out on the **SSPTXD** pin. In a similar way, the MSB of the received data is shifted onto the **SSPRXD** pin by the off-chip serial slave device.

Both the PrimeCell SSP and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each **SSPCLKOUT**. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of **PCLK** after the LSB has been latched.

Figure 71 shows the Texas Instruments synchronous serial frame format when back-to-back frames are transmitted.

Figure 71. Texas Instruments synchronous serial frame format, continuous transfer



12.3.3.9. Motorola SPI frame format

The Motorola SPI interface is a four-wire interface where the **SSPFSSOUT** signal behaves as a slave select. The main feature of the Motorola SPI format is that you can program the inactive state and phase of the **SSPCLKOUT** signal using the **SPO** and **SPH** bits of the **SSPSCR0** control register.

12.3.3.9.1. **SPO**, clock polarity

When the **SPO** clock polarity control bit is LOW, it produces a steady state LOW value on the **SSPCLKOUT** pin. If the **SPO** clock polarity control bit is HIGH, a steady state HIGH value is placed on the **SSPCLKOUT** pin when data is not being transferred.

12.3.3.9.2. **SPH**, clock phase

The **SPH** control bit selects the clock edge that captures data and enables it to change state. It has the most impact on the first bit transmitted by either permitting or not permitting a clock transition before the first data capture edge.

When the **SPH** phase control bit is LOW, data is captured on the first clock edge transition.

When the **SPH** clock phase control bit is HIGH, data is captured on the second clock edge transition.

12.3.3.10. Motorola SPI Format with **SPO=0, SPH=0**

Figure 72 and Figure 73 shows a continuous transmission signal sequence for Motorola SPI frame format with **SPO=0, SPH=0**. Figure 72 shows a single transmission signal sequence for Motorola SPI frame format with **SPO=0, SPH=0**.

Figure 72. Motorola SPI frame format, single transfer, with **SPO=0** and **SPH=0**

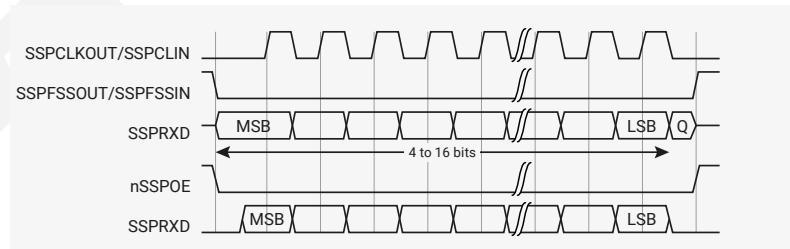
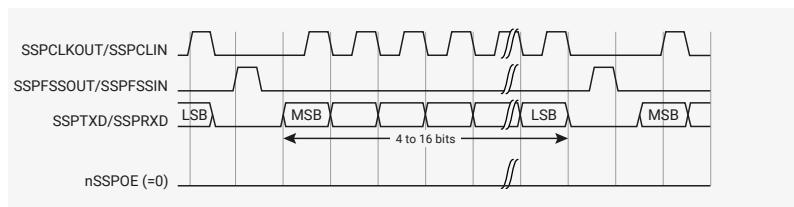


Figure 73 shows a continuous transmission signal sequence for Motorola SPI frame format with **SPO=0, SPH=0**.

Figure 73. Motorola SPI frame format, single transfer, with $SPO=0$ and $SPH=0$



In this configuration, during idle periods:

- the **SSPCLKOUT** signal is forced LOW
- the **SSPFSSOUT** signal is forced HIGH
- the transmit data line **SSPTXD** is arbitrarily forced LOW
- the **nSSPOE** pad enable signal is forced HIGH (this is not connected to the pad in RP2350)
- when the PrimeCell SSP is configured as a master, the **nSSPCTLOE** line is driven LOW, enabling the **SSPCLKOUT** pad, active-LOW enable
- when the PrimeCell SSP is configured as a slave, the **nSSPCTLOE** line is driven HIGH, disabling the **SSPCLKOUT** pad, active-LOW enable

If the PrimeCell SSP is enable, and there is valid data within the transmit FIFO, the start of transmission is signified by the **SSPFSSOUT** master signal being driven LOW. This causes slave data to be enabled onto the **SSPRXD** input line of the master. The **nSSPOE** line is driven LOW, enabling the master **SSPTXD** output pad.

One-half **SSPCLKOUT** period later, valid master data is transferred to the **SSPTXD** pin. Now that both the master and slave data have been set, the **SSPCLKOUT** master clock pin goes HIGH after one additional half **SSPCLKOUT** period.

The data is now captured on the rising and propagated on the falling edges of the **SSPCLKOUT** signal.

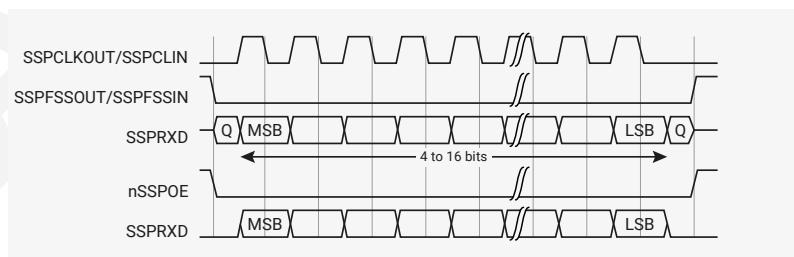
In the case of a single word transmission, after all bits of the data word have been transferred, the **SSPFSSOUT** line is returned to its idle HIGH state one **SSPCLKOUT** period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the **SSPFSSOUT** signal pulse HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not permit it to be altered if the **SPH** bit is logic zero. Therefore, the master device must raise the **SSPFSSIN** pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the **SSPFSSOUT** pin is returned to its idle state one **SSPCLKOUT** period after the last bit has been captured.

12.3.3.11. Motorola SPI Format with $SPO=0$, $SPH=1$

Figure 74 shows the transfer signal sequence for Motorola SPI format with $SPO=0$, $SPH=1$, and it covers both single and continuous transfers.

Figure 74. Motorola SPI frame format with $SPO=0$ and $SPH=1$, single and continuous transfers



In this configuration, during idle periods:

- the **SSPCLKOUT** signal is forced LOW
- The **SSPFSSOUT** signal is forced HIGH
- the transmit data line **SSPTXD** is arbitrarily forced LOW

- the **nSSPOE** pad enable signal is forced HIGH (not connected to the pad in RP2350)
- when the PrimeCell SSP is configured as a master, the **nSSPCTLOE** line is driven LOW, enabling the **SSPCLKOUT** pad, active-LOW enable
- when the PrimeCell SSP is configured as a slave, the **nSSPCTLOE** line is driven HIGH, disabling the **SSPCLKOUT** pad, active-LOW enable

If the PrimeCell SSP is enabled, and there is valid data within the transmit FIFO, the start of transmission is signified by the **SSPFSSOUT** master signal being driven LOW. The **nSSPOE** line is driven LOW, enabling the master **SSPTXD** output pad. After an additional one half **SSPCLKOUT** period, both master and slave valid data is enabled onto their respective transmission lines. At the same time, the **SSPCLKOUT** is enabled with a rising edge transition.

Data is then captured on the falling edges and propagated on the rising edges of the **SSPCLKOUT** signal.

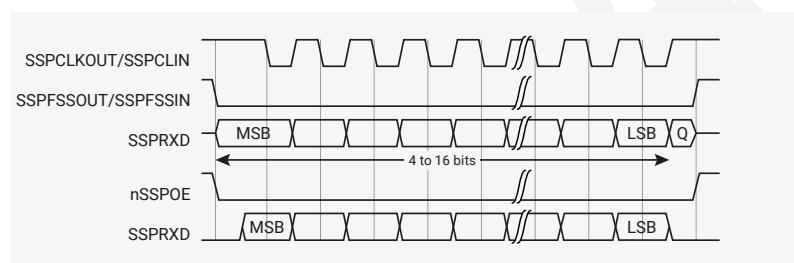
In the case of a single word transfer, after all bits have been transferred, the **SSPFSSOUT** line is returned to its idle HIGH state one **SSPCLKOUT** period after the last bit has been captured. For continuous back-to-back transfers, the **SSPFSSOUT** pin is held LOW between successive data words and termination is the same as that of the single word transfer.

12.3.3.12. Motorola SPI Format with **SPO=1, SPH=0**

[Figure 75](#) and [Figure 76](#) show single and continuous transmission signal sequences for Motorola SPI format with **SPO=1, SPH=0**.

[Figure 75](#) shows a single transmission signal sequence for Motorola SPI format with **SPO=1, SPH=0**.

Figure 75. Motorola SPI frame format, single transfer, with **SPO=1** and **SPH=0**

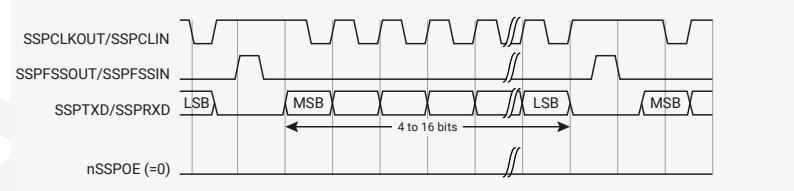


[Figure 76](#) shows a continuous transmission signal sequence for Motorola SPI format with **SPO=1, SPH=0**.

NOTE

In [Figure 75](#), **Q** is an undefined signal.

Figure 76. Motorola SPI frame format, continuous transfer, with **SPO=1** and **SPH=0**



In this configuration, during idle periods:

- the **SSPCLKOUT** signal is forced HIGH
- the **SSPFSSOUT** signal is forced HIGH
- the transmit data line **SSPTXD** is arbitrarily forced LOW
- the **nSSPOE** pad enable signal is forced HIGH (not connected to the pad in RP2350)
- when the PrimeCell SSP is configured as a master, the **nSSPCTLOE** line is driven LOW, enabling the **SSPCLKOUT** pad, active-LOW enable

- when the PrimeCell SSP is configured as a slave, the **nSSPCTLOE** line is driven HIGH, disabling the **SSPCLKOUT** pad, active-LOW enable

If the PrimeCell SSP is enabled, and there is valid data within the transmit FIFO, the start of transmission is signified by the **SSPFSSOUT** master signal being driven LOW, and this causes slave data to be immediately transferred onto the **SSPRXD** line of the master. The **nSSPOE** line is driven LOW, enabling the master **SSPTXD** output pad.

One half period later, valid master data is transferred to the **SSPTXD** line. Now that both the master and slave data have been set, the **SSPCLKOUT** master clock pin becomes LOW after one additional half **SSPCLKOUT** period. This means that data is captured on the falling edges and be propagated on the rising edges of the **SSPCLKOUT** signal.

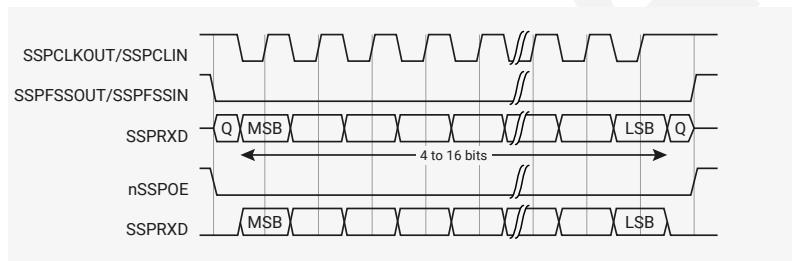
In the case of a single word transmission, after all bits of the data word are transferred, the **SSPFSSOUT** line is returned to its idle HIGH state one **SSPCLKOUT** period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the **SSPFSSOUT** signal must be pulsed HIGH between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not permit it to be altered if the **SPH** bit is logic zero. Therefore, the master device must raise the **SSPFSSIN** pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the **SSPFSSOUT** pin is returned to its idle state one **SSPCLKOUT** period after the last bit has been captured.

12.3.3.13. Motorola SPI Format with **SPO=1, SPH=1**

Figure 77 shows the transfer signal sequence for Motorola SPI format with **SPO=1, SPH=1**, and it covers both single and continuous transfers.

Figure 77. Motorola SPI frame format with **SPO=1** and **SPH=1**, single and continuous transfers



NOTE

In Figure 77, Q is an undefined signal.

In this configuration, during idle periods:

- the **SSPCLKOUT** signal is forced HIGH
- the **SSPFSSOUT** signal is forced HIGH
- the transmit data line **SSPTXD** is arbitrarily forced LOW
- the **nSSPOE** pad enable signal is forced HIGH (not connected to the pad in RP2350)
- when the PrimeCell SSP is configured as a master, the **nSSPCTLOE** line is driven LOW, enabling the **SSPCLKOUT** pad, active-LOW enable
- when the PrimeCell SSP is configured as a slave, the **nSSPCTLOE** line is driven HIGH, disabling the **SSPCLKOUT** pad, active-LOW enable.

If the PrimeCell SSP is enabled, and there is valid data within the transmit FIFO, the start of transmission is signified by the **SSPFSSOUT** master signal being driven LOW. The **nSSPOE** line is driven LOW, enabling the master **SSPTXD** output pad. After an additional one half **SSPCLKOUT** period, both master and slave data are enabled onto their respective transmission lines. At the same time, the **SSPCLKOUT** is enabled with a falling edge transition. Data is then captured on the rising edges and propagated on the falling edges of the **SSPCLKOUT** signal.

After all bits have been transferred, in the case of a single word transmission, the **SSPFSSOUT** line is returned to its idle HIGH state one **SSPCLKOUT** period after the last bit has been captured.

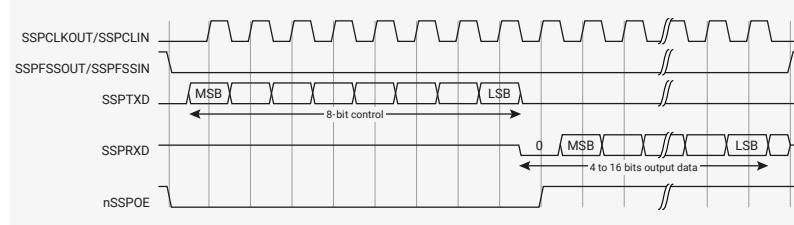
For continuous back-to-back transmissions, the **SSPFSSOUT** pin remains in its active-LOW state, until the final bit of the last word has been captured, and then returns to its idle state as the previous section describes.

For continuous back-to-back transfers, the **SSPFSSOUT** pin is held LOW between successive data words and termination is the same as that of the single word transfer.

12.3.3.14. National Semiconductor Microwire frame format

Figure 78 shows the National Semiconductor Microwire frame format for a single frame. Figure 79 shows the same format when back to back frames are transmitted.

Figure 78. Microwire frame format, single transfer



Microwire format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex, using a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the PrimeCell SSP to the off-chip slave device. During this transmission, the PrimeCell SSP receives no incoming data. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 bits in length, making the total frame length in the range 13-25 bits.

In this configuration, during idle periods:

- **SSPCLKOUT** is forced LOW
- **SSPFSSOUT** is forced HIGH
- the transmit data line, **SSPTXD**, is arbitrarily forced LOW
- the **nSSPOE** pad enable signal is forced HIGH (not connected to the pad in RP2350)

A transmission is triggered by writing a control byte to the transmit FIFO. The falling edge of **SSPFSSOUT** causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto the **SSPTXD** pin. **SSPFSSOUT** remains LOW for the duration of the frame transmission. The **SSPRXD** pin remains tristated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on the rising edge of each **SSPCLKOUT**. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the PrimeCell SSP. Each bit is driven onto **SSPRXD** line on the falling edge of **SSPCLKOUT**. The PrimeCell SSP in turn latches each bit on the rising edge of **SSPCLKOUT**. At the end of the frame, for single transfers, the **SSPFSSOUT** signal is pulled HIGH one clock period after the last bit has been latched in the receive serial shifter, that causes the data to be transferred to the receive FIFO.

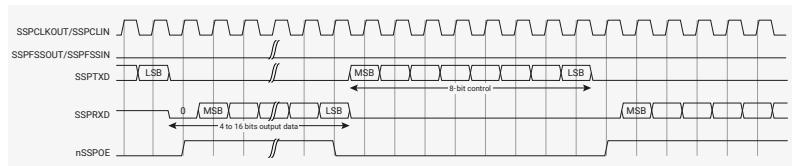
NOTE

The off-chip slave device can tristate the receive line either on the falling edge of **SSPCLKOUT** after the LSB has been latched by the receive shifter, or when the **SSPFSSOUT** pin goes HIGH.

For continuous transfers, data transmission begins and ends in the same manner as a single transfer. However, the **SSPFSSOUT** line is continuously asserted, held LOW, and transmission of data occurs back-to-back. The control byte of the next frame follows directly after the LSB of the received data from the current frame. Each of the received values is transferred from the receive shifter on the falling edge **SSPCLKOUT**, after the LSB of the frame has been latched into the PrimeCell SSP.

Figure 79 shows the National Semiconductor Microwire frame format when back-to-back frames are transmitted.

Figure 79. Microwire frame format, continuous transfers



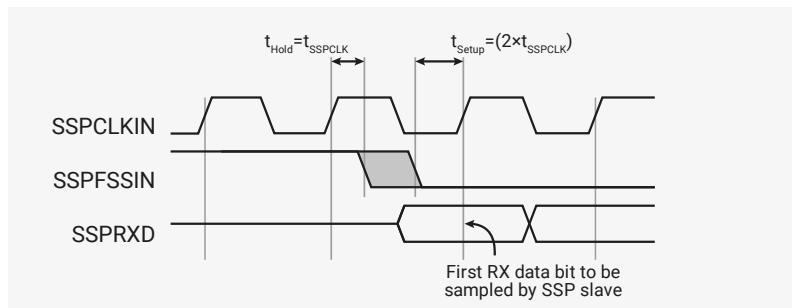
In Microwire mode, the PrimeCell SSP slave samples the first bit of receive data on the rising edge of **SSPCLKIN** after **SSPFSSIN** has gone LOW. Masters that drive a free-running **SSPCLKIN** must ensure that the **SSPFSSIN** signal has sufficient setup and hold margins with respect to the rising edge of **SSPCLKIN**.

Figure 80 shows these setup and hold time requirements.

With respect to the **SSPCLKIN** rising edge on which the first bit of receive data is to be sampled by the PrimeCell SSP slave, **SSPFSSIN** must have a setup of at least two times the period of **SSPCLK** on which the PrimeCell SSP operates.

With respect to the **SSPCLKIN** rising edge previous to this edge, **SSPFSSIN** must have a hold of at least one **SSPCLK** period.

Figure 80. Microwire frame format, SSPFSSIN input setup and hold requirements



12.3.3.15. Examples of master and slave configurations

Figure 81, Figure 82, and Figure 83 shows how you can connect the PrimeCell SSP (**PL022**) peripheral to other synchronous serial peripherals, when it is configured as a master or a slave.

NOTE

The SSP (**PL022**) does not support dynamic switching between master and slave in a system. Each instance is configured and connected either as a master or slave.

Figure 81 shows the PrimeCell SSP (**PL022**) instance twice, as a single master and one slave. The master can broadcast to the slave through the master **SSPTXD** line. In response, the slave drives its **nSSPOE** signal HIGH, enabling its **SSPTXD** data onto the **SSPRXD** line of the master.

Figure 81. PrimeCell SSP master coupled to a PL022 slave

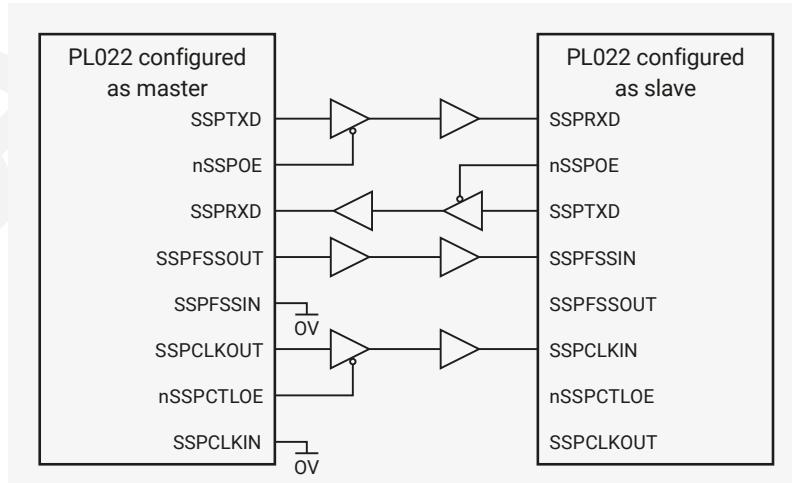


Figure 82 shows how an PrimeCell SSP (**PL022**), configured as master, interfaces to a Motorola SPI slave. The SPI Slave

Select (**SS**) signal is permanently tied LOW and configures it as a slave. Similar to the above operation, the master can broadcast to the slave through the master PrimeCell SSP **SSPTXD** line. In response, the slave drives its SPI **MISO** port onto the **SSPRXD** line of the master.

Figure 82. PrimeCell SSP master coupled to an SPI slave

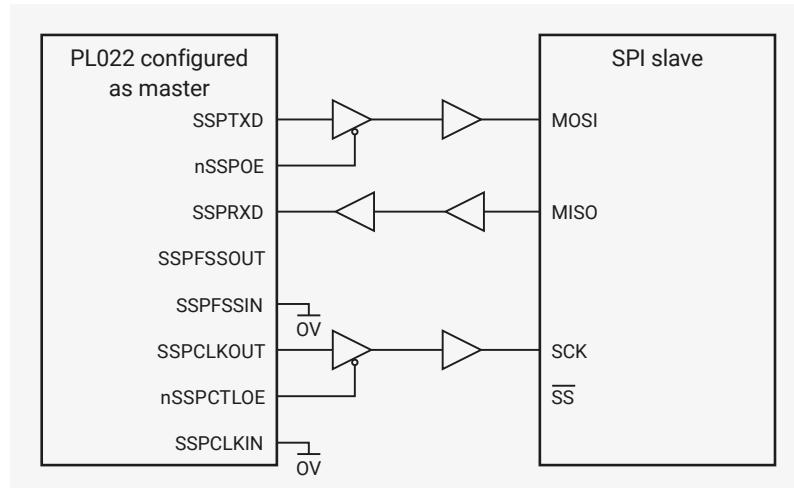
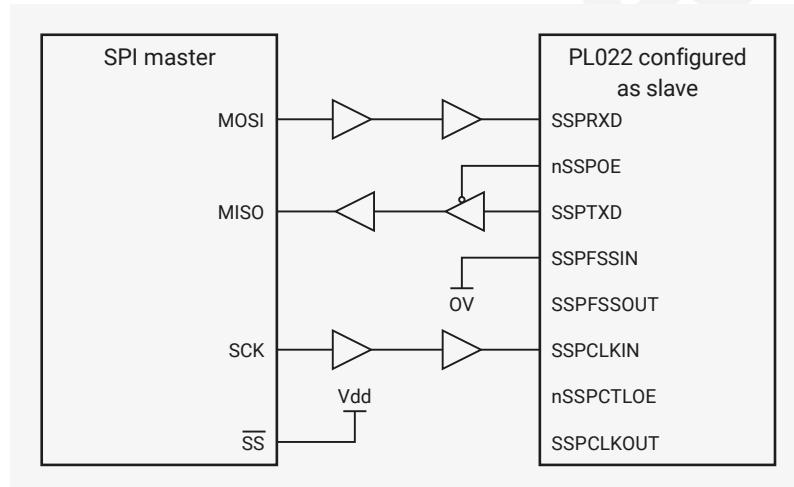


Figure 83 shows a Motorola SPI configured as a master and interfaced to an instance of a PrimeCell SSP (PL022) configured as a slave. In this case, the slave Select Signal (**SS**) is permanently tied HIGH to configure it as a master. The master can broadcast to the slave through the master SPI **MOSI** line and in response, the slave drives its **nSSPOE** signal LOW. This enables its **SSPTXD** data onto the **MISO** line of the master.

Figure 83. SPI master coupled to a PrimeCell SSP slave



12.3.3.16. PrimeCell DMA interface

The PrimeCell SSP provides an interface to connect to the DMA controller. The PrimeCell SSP DMA control register, **SSPDMACR** controls the DMA operation of the PrimeCell SSP.

The DMA interface includes the following signals, for receive:

SSPRXDMASREQ

Single-character DMA transfer request, asserted by the SSP. This signal is asserted when the receive FIFO contains at least one character.

SSPRXDMABREQ

Burst DMA transfer request, asserted by the SSP. This signal is asserted when the receive FIFO contains four or more characters.

SSPRXDMACLR

DMA request clear, asserted by the DMA controller to clear the receive request signals. If DMA burst transfer is requested, the clear signal is asserted during the transfer of the last data in the burst.

The DMA interface includes the following signals, for transmit:

SSPTXDMASREQ

Single-character DMA transfer request, asserted by the SSP. This signal is asserted when there is at least one empty location in the transmit FIFO.

SSPTXDMABREQ

Burst DMA transfer request, asserted by the SSP. This signal is asserted when the transmit FIFO contains four characters or fewer.

SSPTXDMACLR

DMA request clear, asserted by the DMA controller, to clear the transmit request signals. If a DMA burst transfer is requested, the clear signal is asserted during the transfer of the last data in the burst.

The burst transfer and single transfer request signals are not mutually exclusive. They can both be asserted at the same time. For example, when there is more data than the watermark level of four in the receive FIFO, the burst transfer request, and the single transfer request, are asserted. When the amount of data left in the receive FIFO is less than the watermark level, the single request only is asserted. This is useful for situations where the number of characters left to be received in the stream is less than a burst.

For example, if 19 characters must be received, the DMA controller then transfers four bursts of four characters, and three single transfers to complete the stream.

NOTE

For the remaining three characters, the PrimeCell SSP does not assert the burst request.

Each request signal remains asserted until the relevant DMA clear signal is asserted. After the request clear signal is deasserted, a request signal can become active again, depending on the conditions that previous sections describe. All request signals are deasserted if the PrimeCell SSP is disabled, or the DMA enable signal is cleared.

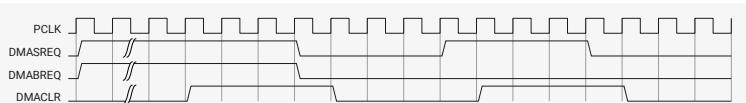
Table 1051 shows the trigger points for **DMABREQ**, for both the transmit and receive FIFOs.

Table 1051. DMA trigger points for the transmit and receive FIFOs

Burst length		
Watermark level	Transmit, number of empty locations	Receive, number of filled locations
1/2	4	4

Figure 84 shows the timing diagram for both a single transfer request, and a burst transfer request, with the appropriate DMA clear signal. The signals are all synchronous to PCLK.

Figure 84. DMA transfer waveforms



12.3.4. List of Registers

The **SPI0** and **SPI1** registers start at base addresses of **0x40080000** and **0x40088000** respectively (defined as **SPI0_BASE** and **SPI1_BASE** in SDK).

Table 1052. List of SPI registers

Offset	Name	Info
0x000	SSPCR0	Control register 0, SSPCR0 on page 3-4
0x004	SSPCR1	Control register 1, SSPCR1 on page 3-5

Offset	Name	Info
0x008	SSPDR	Data register, SSPDR on page 3-6
0x00c	SSPSR	Status register, SSPSR on page 3-7
0x010	SSPCPSR	Clock prescale register, SSPCPSR on page 3-8
0x014	SSPIMSC	Interrupt mask set or clear register, SSPIMSC on page 3-9
0x018	SSPRIS	Raw interrupt status register, SSPRIS on page 3-10
0x01c	SSPMIS	Masked interrupt status register, SSPMIS on page 3-11
0x020	SSPICR	Interrupt clear register, SSPICR on page 3-11
0x024	SSPDMACR	DMA control register, SSPDMACR on page 3-12
0xfe0	SSPPERIPHIDO	Peripheral identification registers, SSPPERIPHIDO-3 on page 3-13
0xfe4	SSPPERIPHID1	Peripheral identification registers, SSPPERIPHID0-3 on page 3-13
0xfe8	SSPPERIPHID2	Peripheral identification registers, SSPPERIPHID0-3 on page 3-13
0xfc	SSPPERIPHID3	Peripheral identification registers, SSPPERIPHID0-3 on page 3-13
0xff0	SSPCELLID0	PrimeCell identification registers, SSPPCELLID0-3 on page 3-16
0xff4	SSPPCELLID1	PrimeCell identification registers, SSPPCELLID0-3 on page 3-16
0xff8	SSPPCELLID2	PrimeCell identification registers, SSPPCELLID0-3 on page 3-16
0ffc	SSPPCELLID3	PrimeCell identification registers, SSPPCELLID0-3 on page 3-16

SPI: SSPCR0 Register

Offset: 0x000

Description

Control register 0, SSPCR0 on page 3-4

Table 1053. SSPCR0 Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:8	SCR	Serial clock rate. The value SCR is used to generate the transmit and receive bit rate of the PrimeCell SSP. The bit rate is: $F_{SSPCLK} \cdot CPSDVSR \cdot (1+SCR)$ where CPSDVSR is an even value from 2-254, programmed through the SSPCPSR register and SCR is a value from 0-255.	RW	0x00
7	SPH	SSPCLKOUT phase, applicable to Motorola SPI frame format only. See Motorola SPI frame format on page 2-10.	RW	0x0
6	SPO	SSPCLKOUT polarity, applicable to Motorola SPI frame format only. See Motorola SPI frame format on page 2-10.	RW	0x0
5:4	FRF	Frame format: 00 Motorola SPI frame format. 01 TI synchronous serial frame format. 10 National Microwire frame format. 11 Reserved, undefined operation.	RW	0x0

Bits	Name	Description	Type	Reset
3:0	DSS	Data Size Select: 0000 Reserved, undefined operation. 0001 Reserved, undefined operation. 0010 Reserved, undefined operation. 0011 4-bit data. 0100 5-bit data. 0101 6-bit data. 0110 7-bit data. 0111 8-bit data. 1000 9-bit data. 1001 10-bit data. 1010 11-bit data. 1011 12-bit data. 1100 13-bit data. 1101 14-bit data. 1110 15-bit data. 1111 16-bit data.	RW	0x0

SPI: SSPCR1 Register

Offset: 0x004

Description

Control register 1, SSPCR1 on page 3-5

Table 1054. SSPCR1 Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	SOD	Slave-mode output disable. This bit is relevant only in the slave mode, MS=1. In multiple-slave systems, it is possible for an PrimeCell SSP master to broadcast a message to all slaves in the system while ensuring that only one slave drives data onto its serial output line. In such systems the RXD lines from multiple slaves could be tied together. To operate in such systems, the SOD bit can be set if the PrimeCell SSP slave is not supposed to drive the SSPTXD line: 0 SSP can drive the SSPTXD output in slave mode. 1 SSP must not drive the SSPTXD output in slave mode.	RW	0x0
2	MS	Master or slave mode select. This bit can be modified only when the PrimeCell SSP is disabled, SSE=0: 0 Device configured as master, default. 1 Device configured as slave.	RW	0x0
1	SSE	Synchronous serial port enable: 0 SSP operation disabled. 1 SSP operation enabled.	RW	0x0
0	LBM	Loop back mode: 0 Normal serial port operation enabled. 1 Output of transmit serial shifter is connected to input of receive serial shifter internally.	RW	0x0

SPI: SSPDR Register

Offset: 0x008

Description

Data register, SSPDR on page 3-6

Table 1055. SSPDR Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:0	DATA	Transmit/Receive FIFO: Read Receive FIFO. Write Transmit FIFO. You must right-justify data when the PrimeCell SSP is programmed for a data size that is less than 16 bits. Unused bits at the top are ignored by transmit logic. The receive logic automatically right-justifies.	RWF	-

SPI: SSPSR Register

Offset: 0x00c

Description

Status register, SSPSR on page 3-7

Table 1056. SSPSR Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	BSY	PrimeCell SSP busy flag, RO: 0 SSP is idle. 1 SSP is currently transmitting and/or receiving a frame or the transmit FIFO is not empty.	RO	0x0
3	RFF	Receive FIFO full, RO: 0 Receive FIFO is not full. 1 Receive FIFO is full.	RO	0x0
2	RNE	Receive FIFO not empty, RO: 0 Receive FIFO is empty. 1 Receive FIFO is not empty.	RO	0x0
1	TNF	Transmit FIFO not full, RO: 0 Transmit FIFO is full. 1 Transmit FIFO is not full.	RO	0x1
0	TFE	Transmit FIFO empty, RO: 0 Transmit FIFO is not empty. 1 Transmit FIFO is empty.	RO	0x1

SPI: SSPCPSR Register

Offset: 0x010

Description

Clock prescale register, SSPCPSR on page 3-8

Table 1057. SSPCPSR Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	CPSDVS	Clock prescale divisor. Must be an even number from 2-254, depending on the frequency of SSPCLK. The least significant bit always returns zero on reads.	RW	0x00

SPI: SSPIMSC Register

Offset: 0x014

Description

Interrupt mask set or clear register, SSPIMSC on page 3-9

Table 1058. SSPIMSC Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3	TXIM	Transmit FIFO interrupt mask: 0 Transmit FIFO half empty or less condition interrupt is masked. 1 Transmit FIFO half empty or less condition interrupt is not masked.	RW	0x0
2	RXIM	Receive FIFO interrupt mask: 0 Receive FIFO half full or less condition interrupt is masked. 1 Receive FIFO half full or less condition interrupt is not masked.	RW	0x0
1	RTIM	Receive timeout interrupt mask: 0 Receive FIFO not empty and no read prior to timeout period interrupt is masked. 1 Receive FIFO not empty and no read prior to timeout period interrupt is not masked.	RW	0x0
0	RORIM	Receive overrun interrupt mask: 0 Receive FIFO written to while full condition interrupt is masked. 1 Receive FIFO written to while full condition interrupt is not masked.	RW	0x0

SPI: SSPRIS Register

Offset: 0x018

Description

Raw interrupt status register, SSPRIS on page 3-10

Table 1059. SSPRIS Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	TXRIS	Gives the raw interrupt state, prior to masking, of the SSPTXINTR interrupt	RO	0x1
2	RXRIS	Gives the raw interrupt state, prior to masking, of the SSPRXINTR interrupt	RO	0x0
1	RTRIS	Gives the raw interrupt state, prior to masking, of the SSPRTINTR interrupt	RO	0x0
0	RORRIS	Gives the raw interrupt state, prior to masking, of the SSPRORINTR interrupt	RO	0x0

SPI: SSPMIS Register

Offset: 0x01c

Description

Masked interrupt status register, SSPMIS on page 3-11

Table 1060. SSPMIS Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	TXMIS	Gives the transmit FIFO masked interrupt state, after masking, of the SSPTXINTR interrupt	RO	0x0
2	RXMIS	Gives the receive FIFO masked interrupt state, after masking, of the SSPRXINTR interrupt	RO	0x0
1	RTMIS	Gives the receive timeout masked interrupt state, after masking, of the SSPRTINTR interrupt	RO	0x0

Bits	Name	Description	Type	Reset
0	RORMIS	Gives the receive over run masked interrupt status, after masking, of the SSPRORINTR interrupt	RO	0x0

SPI: SSPICR Register

Offset: 0x020

Description

Interrupt clear register, SSPICR on page 3-11

Table 1061. SSPICR Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	RTIC	Clears the SSPRTINTR interrupt	WC	0x0
0	RORIC	Clears the SSPRORINTR interrupt	WC	0x0

SPI: SSPDMACR Register

Offset: 0x024

Description

DMA control register, SSPDMACR on page 3-12

Table 1062. SSPDMACR Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	TXDMAE	Transmit DMA Enable. If this bit is set to 1, DMA for the transmit FIFO is enabled.	RW	0x0
0	RXDMAE	Receive DMA Enable. If this bit is set to 1, DMA for the receive FIFO is enabled.	RW	0x0

SPI: SSPPERIPHID0 Register

Offset: 0xfe0

Description

Peripheral identification registers, SSPPeriphID0-3 on page 3-13

Table 1063. SSPPERIPHID0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	PARTNUMBER0	These bits read back as 0x22	RO	0x22

SPI: SSPPERIPHID1 Register

Offset: 0xfe4

Description

Peripheral identification registers, SSPPeriphID0-3 on page 3-13

Table 1064. SSPPERIPHID1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	DESIGNERO	These bits read back as 0x1	RO	0x1
3:0	PARTNUMBER1	These bits read back as 0x0	RO	0x0

SPI: SSPPERIPHID2 Register

Offset: 0xfe8

Description

Peripheral identification registers, SSPPERIPHID0-3 on page 3-13

Table 1065.
SSPPERIPHID2
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:4	REVISION	These bits return the peripheral revision	RO	0x3
3:0	DESIGNER1	These bits read back as 0x4	RO	0x4

SPI: SSPPERIPHID3 Register

Offset: 0xfc

Description

Peripheral identification registers, SSPPERIPHID0-3 on page 3-13

Table 1066.
SSPPERIPHID3
Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	CONFIGURATION	These bits read back as 0x00	RO	0x00

SPI: SSPPCELLID0 Register

Offset: 0xff0

Description

PrimeCell identification registers, SSPPCELLID0-3 on page 3-16

Table 1067.
SSPPCELLID0 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	SSPPCELLID0	These bits read back as 0x0D	RO	0x0d

SPI: SSPPCELLID1 Register

Offset: 0xff4

Description

PrimeCell identification registers, SSPPCELLID0-3 on page 3-16

Table 1068.
SSPPCELLID1 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	SSPPCELLID1	These bits read back as 0xF0	RO	0xf0

SPI: SSPPCELLID2 Register

Offset: 0xff8

Description

PrimeCell identification registers, SSPPCELLID0-3 on page 3-16

Table 1069.
SSPPCELLID2 Register

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
7:0	SSPPCELLID2	These bits read back as 0x05	RO	0x05

SPI: SSPPCELLID3 Register

Offset: 0xffc

Description

PrimeCell identification registers, SSPPCellID0-3 on page 3-16

Table 1070.
SSPPCELLID3 Register

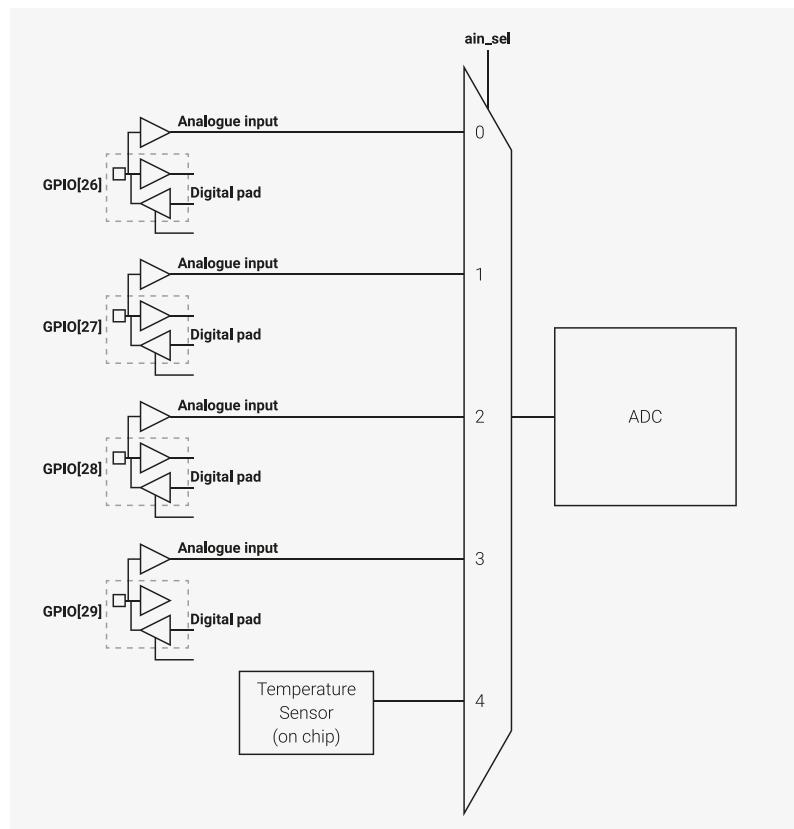
Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7:0	SSPPCELLID3	These bits read back as 0xB1	RO	0xb1

12.4. ADC and Temperature Sensor

RP2350 has an internal analogue-digital converter (ADC) with the following features:

- SAR ADC (see [Section 12.4.3](#))
- 500ks/s (using an independent 48MHz clock)
- 12-bit with 9.2 ENOB (see [Section 12.4.4](#))
- Five or nine input mux:
 - Four inputs available on QFN-60 package pins shared with [GPIO\[29:26\]](#)
 - Eight inputs available on QFN-80 package pins shared with [GPIO\[47:40\]](#)
 - One input dedicated to the internal temperature sensor (see [Section 12.4.6](#))
- Eight element receive sample FIFO
- Interrupt generation
- DMA interface (see [Section 12.4.3.5](#))

Figure 85. ADC Connection Diagram for QFN-60



NOTE

When using an ADC input shared with a GPIO pin, always disable the pin's digital functions by setting `IE` low and `OD` high in the pin's pad control register. See [Section 9.11.3, "Pad Control - User Bank"](#) for details. The maximum ADC input voltage is determined by the digital IO supply voltage (`IOVDD`), not the ADC supply voltage (`ADC_AVDD`). For example, if `IOVDD` is powered at 1.8V, the voltage on the ADC inputs should not exceed 1.8V even if `ADC_AVDD` is powered at 3.3V. Voltages greater than `IOVDD` will result in leakage currents through the ESD protection diodes. See [\[pin_specs\]](#) for details.

12.4.1. Changes from RP2040

- Removed spikes in differential nonlinearity at codes `0x200`, `0x600`, `0xa00` and `0xe00`, as documented by erratum RP2040-E11, improving the ADC's precision by around 0.5 ENOB.
- Increased the number of external ADC input channels from 4 to 8 channels, in the QFN-80 package only.

12.4.2. ADC controller

A digital controller manages the details of operating the RP2350 ADC, and provides additional functionality:

- One-shot or free-running capture mode
- Sample FIFO with DMA interface
- Pacing timer (16 integer bits, 8 fractional bits) for setting free-running sample rate
- Round-robin sampling of multiple channels in free-running capture mode
- Optional right-shift to 8 bits in free-running capture mode, so samples can be DMA'd to a byte buffer in system memory

12.4.2.1. Channel connections

The ADC channels are connected to the following GPIOs in QFN-60

Table 1071. ADC channel connections on QFN-60

Channel	Connection
0	GPIO[26]
1	GPIO[27]
2	GPIO[28]
3	GPIO[29]
4	Temperature Sensor

The ADC channels are connected to the following GPIOs in QFN-80

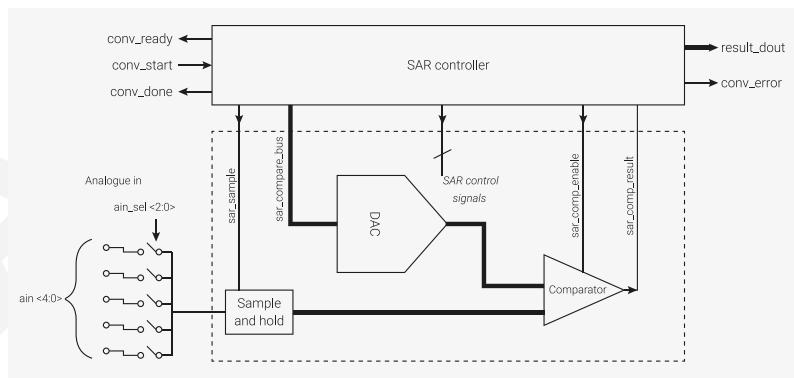
Table 1072. ADC channel connections on QFN-80

Channel	Connection
0	GPIO[40]
1	GPIO[41]
2	GPIO[42]
3	GPIO[43]
4	GPIO[44]
5	GPIO[45]
6	GPIO[46]
7	GPIO[47]
8	Temperature Sensor

12.4.3. SAR ADC

The Successive Approximation Register Analogue to Digital Converter (SAR ADC) is a combination of digital controller and analogue circuit as shown in [Figure 86](#).

Figure 86. SAR ADC Block diagram



The ADC requires a 48MHz clock (`clk_adc`), which could come from the USB PLL. Capturing a sample takes 96 clock cycles ($96 \times 1/48\text{MHz} = 2\mu\text{s}$ per sample (500ks/s)). The clock must be set up correctly before enabling the ADC.

Once the ADC block is provided with a clock, and its reset has been removed, writing a 1 to `CS.EN` will start a short internal power-up sequence for the ADC's analogue hardware. After a few clock cycles, `CS.READY` will go high, indicating the ADC is ready to start its first conversion.

To save power, you can disable the ADC at any time by clearing `CS.EN`. `CS.EN` does not enable the temperature sensor

bias source; it is controlled separately, see [Section 12.4.6](#) for details.

The ADC input is capacitive. When sampling, the ADC places about 1pF across the input. Packaging, PCB routing, and other external factors introduce additional capacitance. The effective impedance, even when sampling at 500ks/s, is over 100kΩ. DC measurements have no need to buffer.

12.4.3.1. One-shot Sample

To select an ADC input, write to to `CS.AINSEL`:

- On QFN-60, there are 4 external inputs, with an `AINSEL` value of 0 → 3 mapping to the ADC input on GPIO26 → GPIO29. Set `AINSEL` to 4 to select the internal temperature sensor.
- On QFN-80, there are 8 external inputs, with an `AINSEL` value of 0 → 7 mapping to the ADC input on GPIO40 → GPIO47. Set `AINSEL` to 8 to select the internal temperature sensor.

Switching `AINSEL` requires no settling time.

Write a 1 to `CS.START_ONCE` to immediately start a new conversion. `CS.READY` will go low to show that a conversion is currently in progress. After 96 cycles of `clk_adc`, `CS.READY` will go high. The 12-bit conversion result is available in `RESULT`.

12.4.3.2. Free-running Sampling

When `CS.START_MANY` is set, the ADC automatically starts new conversions at regular intervals. The most recent conversion result is always available in `RESULT`, but for IRQ or DMA-driven streaming of samples, you must enable the ADC FIFO ([Section 12.4.3.4](#)).

By default (`DIV` = 0), new conversions start immediately after the previous conversion finishes, producing a new sample every 96 cycles. At a clock frequency of 48MHz, this produces 500ks/s.

Set `DIV.INT` to a positive value n to trigger the ADC once per $n + 1$ cycles. The ADC ignores this if a conversion is currently in progress, so generally n will be ≥ 96 . For example, setting `DIV.INT` to 47999 runs the ADC at 1ks/s, if running from a 48MHz clock.

The pacing timer supports fractional-rate division (first order delta sigma). When setting `DIV.FRAC` to a non-zero value, the ADC starts a new conversion once per $1 + \text{INT} + \frac{\text{FRAC}}{256}$ cycles on average, by changing the sample interval between `INT + 1` and `INT + 2`.

12.4.3.3. Sampling Multiple Inputs

`CS.RROBIN` allows the ADC to sample multiple inputs in an interleaved fashion while performing free-running sampling. Each bit in `RROBIN` corresponds to one of the five possible values of `CS.AINSEL`. When the ADC completes a conversion, `CS.AINSEL` automatically cycles to the next input whose corresponding bit is set in `RROBIN`.

To disable the round-robin sampling feature, write all-zeroes to `CS.RROBIN`.

For example, if `AINSEL` is initially 0, and `RROBIN` is set to 0x06 (bits 1 and 2 are set), the ADC samples channels in the following order:

1. Channel 0
2. Channel 1
3. Channel 2
4. Channel 1
5. Channel 2

6. Channel 1

7. Channel 2

The ADC continues to sample channels 1 and 2 indefinitely.

NOTE

The initial value of AINSEL does not need to correspond with a set bit in RROBIN.

12.4.3.4. Sample FIFO

You can read ADC samples directly from the [RESULT](#) register or store them in a local 8-entry FIFO and read out from [FIFO](#). Use the [FCS](#) register to control FIFO operation.

When [FCS.EN](#) is set, the ADC writes each conversion result to the FIFO. A software interrupt handler or the RP2350 DMA can read this sample from the FIFO when notified by the ADC's [IRQ](#) or [DREQ](#) signals. Alternatively, software can poll the status bits in [FCS](#) to wait for each sample to become available.

If the FIFO is full when a conversion completes, the sticky error flag [FCS.OVER](#) is set. When the FIFO is full, the current FIFO contents do not change, so any conversions that complete during this time are lost.

Two flags control the data written to the FIFO by the ADC:

- [FCS.SHIFT](#) right-shifts the FIFO data to eight bits in size (i.e. FIFO bits 7:0 are conversion result bits 11:4). This is suitable for 8-bit DMA transfer to a byte buffer in memory, allowing deeper capture buffers, at the cost of some precision.
- [FCS.ERR](#) sets the [FIFO.ERR](#) flag of each FIFO value, showing that a conversion error took place, i.e. the SAR failed to converge.

Conversion errors indicate that the comparison of one or more bits failed to complete in the time allowed. Conversion errors are typically caused by comparator metastability: the closer to the comparator threshold the input signal is, the longer it takes to make a decision. The higher the comparator gain, the lower the probability of conversion errors.

CAUTION

Because conversion errors produce undefined results, you should always discard samples that contain conversion errors.

12.4.3.5. DMA

The RP2350 DMA ([Section 12.6](#)) can fetch ADC samples from the sample FIFO, by performing a normal memory-mapped read on the [FIFO](#) register, paced by the [ADC.DREQ](#) system data request signal. Before you can use the DMA to fetch ADC samples, you must:

- Enable the sample FIFO ([FCS.EN](#)) so that samples are written to it; the FIFO is disabled by default so that it does not inadvertently fill when the ADC is used for one-shot conversions. Configure the ADC sample rate ([Section 12.4.3.2](#)) before starting the ADC.
- Enable the ADC's data request handshake ([DREQ](#)) via [FCS.DREQ_EN](#).
- In the DMA channel used for the transfer, select the [DREQ_ADC](#) data request signal ([Section 12.6.4.1](#)).
- Set the threshold for [DREQ](#) assertion ([FCS.THRESH](#)) to 1, so that the DMA transfers as soon as a single sample is present in the FIFO. This is also the threshold used for [IRQ](#) assertion, so non-DMA use cases might prefer a higher value for less frequent interrupts.
- If the DMA transfer size is set to 8 bits (so that the DMA transfers to a byte array in memory), set [FCS.SHIFT](#) to pre-shift the FIFO samples to 8 bits of significance.

- To sample multiple input channels, write a mask of those channels to `CS.RROBIN`. Additionally, select the first channel to sample with `CS.AINSEL`.

Once the ADC is suitably configured, start the DMA channel first, then the ADC conversion via `CS.START_MANY`. Once the DMA completes, you can halt the ADC if you are finished sampling, or promptly start a new DMA transfer before the FIFO fills up. After clearing `CS.START_MANY` to halt the ADC, software should poll `CS.READY` to make sure the last conversion has finished, then drain any stray samples from the FIFO.

12.4.3.6. Interrupts

Use `INTE` to generate an interrupt when the FIFO level reaches a threshold defined in `FCS.THRESH`.

Use `INTS` to read the interrupt status. To clear the interrupt, drain the FIFO to a level lower than `FCS.THRESH`.

12.4.3.7. Supply

RP2350 separates the ADC supply out on its own pin to allow noise filtering.

12.4.4. ADC ENOB

12.4.5. INL and DNL

Details to follow.

12.4.6. Temperature Sensor

The temperature sensor measures the `Vbe` voltage of a biased bipolar diode, connected to the fifth ADC channel (`AINSEL=4`) on QFN-60 or the ninth ADC channel (`AINSEL=9`) or QFN-80. Typically, $V_{be} = 0.706V$ at $27^{\circ}C$, with a slope of $-1.721mV$ per degree. Therefore the temperature can be approximated as follows:

$$T = 27 - \frac{(ADC_voltage - 0.706)}{0.001721}$$

As the `Vbe` and the `Vbe` slope can vary over the temperature range, and from device to device, some user calibration may be required if accurate measurements are required.

The temperature sensor's bias source must be enabled before use, via `CS.TS_EN`. This increases current consumption on `ADC_AVDD` by approximately $40\mu A$.

NOTE

The on board temperature sensor is very sensitive to errors in reference voltage. At 3.3V, a value of 891 returned by the ADC corresponds to a temperature of $20.1^{\circ}C$. At a reference voltage 1% lower than 3.3V, the same reading of 891 correspond to a temperature of $24.3^{\circ}C$: a temperature change of over $4^{\circ}C$. To improve the accuracy of the internal temperature sensor, consider adding an external reference voltage.

NOTE

The INL errors, see [Section 12.4.5](#), aren't in the usable temperature range of the ADC.

12.4.7. List of Registers

The ADC registers start at a base address of [0x400a0000](#) (defined as [ADC_BASE](#) in SDK).

Table 1073. List of ADC registers

Offset	Name	Info
0x00	CS	ADC Control and Status
0x04	RESULT	Result of most recent ADC conversion
0x08	FCS	FIFO control and status
0x0c	FIFO	Conversion result FIFO
0x10	DIV	Clock divider. If non-zero, CS_START_MANY will start conversions at regular intervals rather than back-to-back. The divider is reset when either of these fields are written. Total period is $1 + \text{INT} + \text{FRAC} / 256$
0x14	INTR	Raw Interrupts
0x18	INTE	Interrupt Enable
0x1c	INTF	Interrupt Force
0x20	INTS	Interrupt status after masking & forcing

ADC: CS Register

Offset: 0x00

Description

ADC Control and Status

Table 1074. CS Register

Bits	Name	Description	Type	Reset
31:25	Reserved.	-	-	-
24:16	RROBIN	Round-robin sampling. 1 bit per channel. Set all bits to 0 to disable. Otherwise, the ADC will cycle through each enabled channel in a round-robin fashion. The first channel to be sampled will be the one currently indicated by AINSEL . AINSEL will be updated after each conversion with the newly-selected channel.	RW	0x000
15:12	AINSEL	Select analog mux input. Updated automatically in round-robin mode. This is corrected for the package option so only ADC channels which are bonded are available, and in the correct order	RW	0x0
11	Reserved.	-	-	-
10	ERR_STICKY	Some past ADC conversion encountered an error. Write 1 to clear.	WC	0x0

Bits	Name	Description	Type	Reset
9	ERR	The most recent ADC conversion encountered an error; result is undefined or noisy.	RO	0x0
8	READY	1 if the ADC is ready to start a new conversion. Implies any previous conversion has completed. 0 whilst conversion in progress.	RO	0x0
7:4	Reserved.	-	-	-
3	START_MANY	Continuously perform conversions whilst this bit is 1. A new conversion will start immediately after the previous finishes.	RW	0x0
2	START_ONCE	Start a single conversion. Self-clearing. Ignored if start_many is asserted.	SC	0x0
1	TS_EN	Power on temperature sensor. 1 - enabled. 0 - disabled.	RW	0x0
0	EN	Power on ADC and enable its clock. 1 - enabled. 0 - disabled.	RW	0x0

ADC: RESULT Register

Offset: 0x04

Table 1075. RESULT Register

Bits	Description	Type	Reset
31:12	Reserved.	-	-
11:0	Result of most recent ADC conversion	RO	0x000

ADC: FCS Register

Offset: 0x08

Description

FIFO control and status

Table 1076. FCS Register

Bits	Name	Description	Type	Reset
31:28	Reserved.	-	-	-
27:24	THRESH	DREQ/IRQ asserted when level \geq threshold	RW	0x0
23:20	Reserved.	-	-	-
19:16	LEVEL	The number of conversion results currently waiting in the FIFO	RO	0x0
15:12	Reserved.	-	-	-
11	OVER	1 if the FIFO has been overflowed. Write 1 to clear.	WC	0x0
10	UNDER	1 if the FIFO has been underflowed. Write 1 to clear.	WC	0x0
9	FULL		RO	0x0
8	EMPTY		RO	0x0
7:4	Reserved.	-	-	-
3	DREQ_EN	If 1: assert DMA requests when FIFO contains data	RW	0x0
2	ERR	If 1: conversion error bit appears in the FIFO alongside the result	RW	0x0

Bits	Name	Description	Type	Reset
1	SHIFT	If 1: FIFO results are right-shifted to be one byte in size. Enables DMA to byte buffers.	RW	0x0
0	EN	If 1: write result to the FIFO after each conversion.	RW	0x0

ADC: FIFO Register

Offset: 0x0c

Description

Conversion result FIFO

Table 1077. FIFO Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	ERR	1 if this particular sample experienced a conversion error. Remains in the same location if the sample is shifted.	RF	-
14:12	Reserved.	-	-	-
11:0	VAL		RF	-

ADC: DIV Register

Offset: 0x10

Description

Clock divider. If non-zero, CS_START_MANY will start conversions at regular intervals rather than back-to-back.

The divider is reset when either of these fields are written.

Total period is $1 + \text{INT} + \text{FRAC} / 256$

Table 1078. DIV Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:8	INT	Integer part of clock divisor.	RW	0x0000
7:0	FRAC	Fractional part of clock divisor. First-order delta-sigma.	RW	0x00

ADC: INTR Register

Offset: 0x14

Description

Raw Interrupts

Table 1079. INTR Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	FIFO	Triggered when the sample FIFO reaches a certain level. This level can be programmed via the FCS_THRESH field.	RO	0x0

ADC: INT Enable Register

Offset: 0x18

Description

Interrupt Enable

Table 1080. INTF Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	FIFO	Triggered when the sample FIFO reaches a certain level. This level can be programmed via the FCS_THRESH field.	RW	0x0

ADC: INTF Register

Offset: 0x1c

Description

Interrupt Force

Table 1081. INTF Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	FIFO	Triggered when the sample FIFO reaches a certain level. This level can be programmed via the FCS_THRESH field.	RW	0x0

ADC: INTS Register

Offset: 0x20

Description

Interrupt status after masking & forcing

Table 1082. INTS Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	FIFO	Triggered when the sample FIFO reaches a certain level. This level can be programmed via the FCS_THRESH field.	RO	0x0

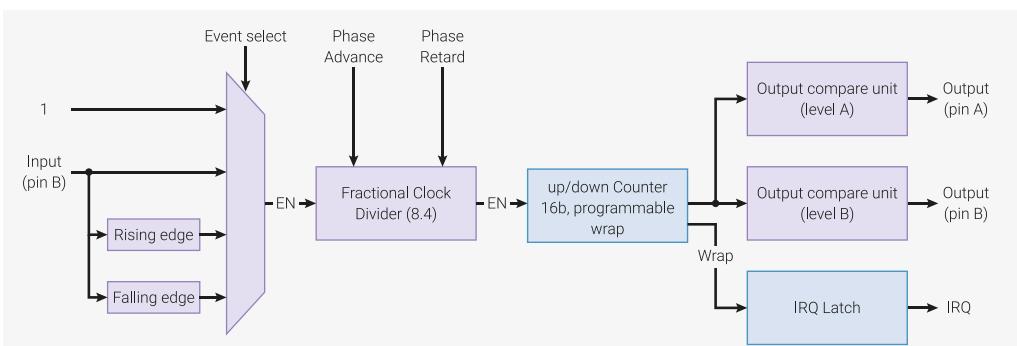
12.5. PWM

12.5.1. Overview

Pulse width modulation (PWM) smoothly varies the average voltage of a digital signal using controlled-width positive pulses at regular intervals. The fraction of time spent high is known as the duty cycle. This may be used to approximate an analogue output or control switchmode power electronics.

The RP2350 PWM block has 12 identical slices. Each slice can drive two PWM output signals, or measure the frequency or duty cycle of an input signal. The two outputs on each slice have the same period, but independently varying duty cycles, so this gives a total of 24 controllable PWM outputs in the QFN-80 package.

Figure 87. A single PWM slice. A 16-bit counter counts from 0 up to some programmed value, and then wraps to zero, or counts back down again, depending on PWM mode. The A and B outputs transition high and low based on the current count value and the preprogrammed A and B thresholds. The counter advances based on a number of events: it may be free-running, or gated by level or edge of an input signal on the B pin. A fractional divider slows the overall count rate for finer control of output frequency.



Each PWM slice is equipped with the following:

- 16-bit counter
- 8.4 fractional clock divider
- Two independent output channels, duty cycle from 0% to 100% **inclusive**
- Dual slope and trailing edge modulation
- Edge-sensitive input mode for frequency measurement
- Level-sensitive input mode for duty cycle measurement
- Configurable counter wrap value
 - Wrap and level registers are double buffered and can be changed race-free while PWM is running
- Interrupt request and DMA request on counter wrap
- Phase can be precisely advanced or retarded while running (increments of one count)

Slices can be enabled or disabled simultaneously via a single global control register. Slices then run in lockstep, so that more complex power circuitry can be switched by the outputs of multiple slices.

12.5.1.1. Changes from RP2040

- Increased the number of slices from 8 to 12, with the 4 additional slices available on GPIOs 32 through 47 in the QFN-80 package.
- Added a second shared interrupt line (controlled by [IRQ1_INTE](#)), to aid use of PWM slices as simple repeating timers.

12.5.2. Programmer's Model

All GPIO pins on RP2350 can be used for PWM:

Table 1083. Mapping of PWM channels to GPIO pins on RP2350. This is also shown in the main GPIO function table, [Table 646](#)

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
PWM Channel	8A	8B	9A	9B	10A	10B	11A	11B	8A	8B	9A	9B	10A	10B	11A	11B

- The first 16 PWM channels (8 × 2-channel slices) appear on GPIOs 0 through 15, in the order [PWM0_A](#), [PWM0_B](#), [PWM1_A](#), and so on.

- This pattern repeats for GPIOs 16 through 31. GPIO16 is **PWM0 A**, GPIO17 is **PWM0 B**, and so on up to **PWM7 B** on GPIO31. GPIO30 and above are available only in the QFN-80 package.
- The remaining 8 PWM channels (4×2 -channel slices) appear on GPIOs 32 through 39, and then repeat on GPIOs 40 through 47.
- If you select the same PWM output on two GPIO pins, the same signal appears on both.
- If you use **B** pin as an input and select it on multiple GPIO pins, the PWM slice sees the logical OR of those two GPIO inputs.

NOTE

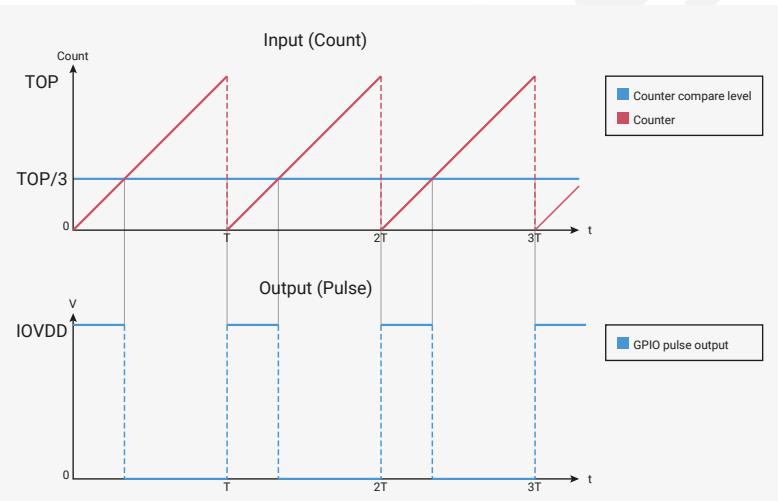
GPIOs 0 through 29 have the same channel assignment as RP2040 for pinout compatibility. This reduces the maximum number of independent PWM outputs in the QFN-60 package option of RP2350, but you can still use slices 8 through 11 for repeating timer interrupts in this package.

12.5.2.1. Pulse Width Modulation

The PWM hardware continuously compares an input value to a free-running counter. This produces a toggling output; the amount of time spent at the high output level corresponds to the input value. The fraction of time spent at the high signal level is known as the duty cycle of the signal.

The counting period is controlled by the **TOP** register, with a maximum possible period of 65536 cycles, as the counter and **TOP** are 16 bits in size. Use the **CC** register to configure input values.

Figure 88. The counter repeatedly counts from 0 to **TOP**, forming a sawtooth shape. The counter is continuously compared with some input value. When the input value is higher than the counter, the output is driven high. Otherwise, the output is low. The output period T is defined by the **TOP** value of the counter, and how fast the counter is configured to count. The average output voltage, as a fraction of the IO power supply, is the input value divided by the counter period ($TOP + 1$)



This example shows the counting period and the A and B counter compare levels being configured on one of RP2350's PWM slices.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/pwm/hello_pwm/hello_pwm.c Lines 15 - 29

```

15  // Tell GPIO 0 and 1 they are allocated to the PWM
16  gpio_set_function(0, GPIO_FUNC_PWM);
17  gpio_set_function(1, GPIO_FUNC_PWM);
18
19  // Find out which PWM slice is connected to GPIO 0 (it's slice 0)
20  uint slice_num = pwm_gpio_to_slice_num(0);
21
22  // Set period of 4 cycles (0 to 3 inclusive)
23  pwm_set_wrap(slice_num, 3);
24  // Set channel A output high for one cycle before dropping
25  pwm_set_chan_level(slice_num, PWM_CHAN_A, 1);
26  // Set initial B output high for three cycles before dropping

```

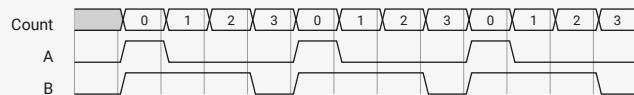
```

27     pwm_set_chan_level(slice_num, PWM_CHAN_B, 3);
28     // Set the PWM running
29     pwm_set_enabled(slice_num, true);

```

Figure 89 shows how the PWM hardware operates once it has been configured.

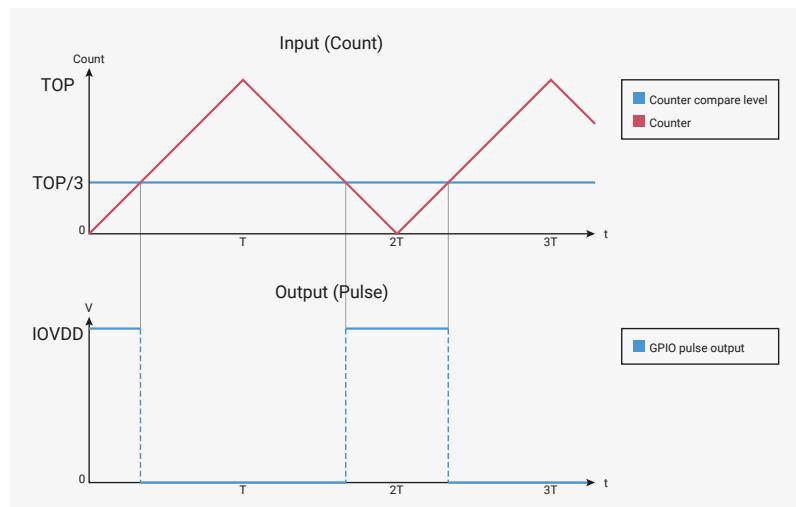
Figure 89. The slice counts repeatedly from 0 to 3, which is configured as the **TOP** value. The output waves therefore have a period of 4. Output A is high for 1 cycle in 4, so the average output voltage is 1/4 of the IO supply voltage. Output B is high for 3 cycles in every 4. Note the rising edges of A and B are always aligned.



By default, PWM slices count upward until they reach the value of the **TOP** register. After they reach the **TOP** value, they wrap to 0. Alternatively, set **CSR_PH_CORRECT** to 1 to enable **phase-correct mode**, where the counter counts downward after reaching **TOP**, until it reaches 0 again.

Phase-correct mode centres the pulse on the same point no matter the duty cycle; its phase is not a function of duty cycle. When phase-correct mode is enabled, the output frequency is halved. The slice spends two cycles at a count of **TOP** and two cycles at a count of 0 each PWM period.

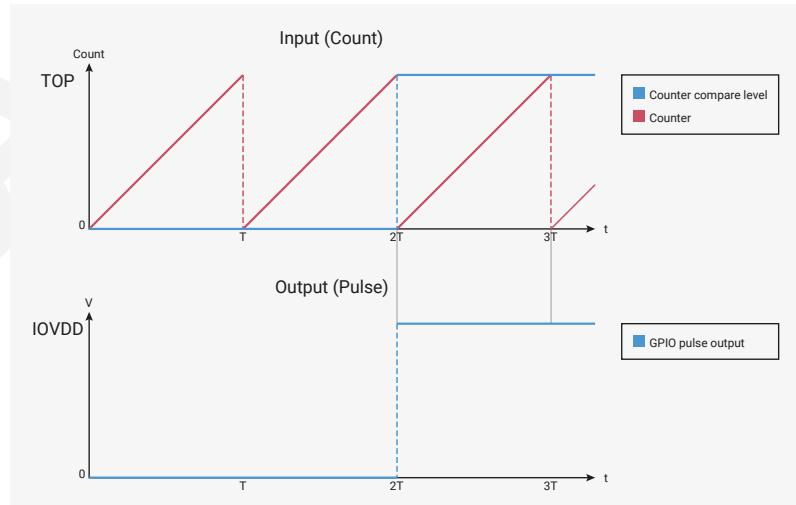
Figure 90. In phase-correct mode, the counter counts back down from **TOP** to 0 once it reaches **TOP**.



12.5.2.2. 0% and 100% Duty Cycle

The RP2350 PWM can produce toggle-free 0% and 100% duty cycle output.

Figure 91. Glitch-free 0% duty cycle output for **CC** = 0, and glitch-free 100% duty cycle output for **CC** = **TOP** + 1



A **CC** value of 0 produces a 0% output: the output signal is always low. A **CC** value of **TOP** + 1 (equal to the period when not

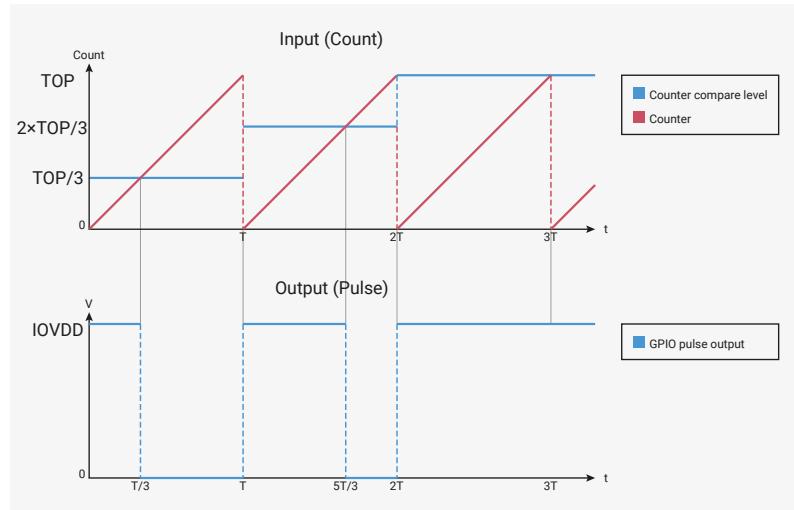
phase-corrected) produces a 100% output. If **TOP** is 254, the counter has a period of 255 cycles, and **CC** values in the range of 0 to 255 inclusive will produce duty cycles in the range 0% to 100% inclusive.

Glitch-free output at 0% and 100% helps avoid switching losses, for instance, when a MOSFET is controlled at its minimum and maximum current levels.

12.5.2.3. Double Buffering

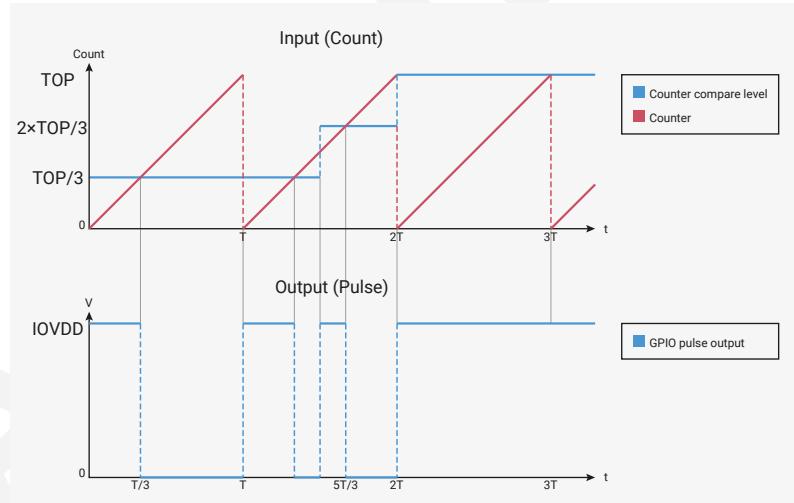
Figure 92 shows how a change in input value produces a change in output duty cycle. This can approximate analogue waveforms such as a sine wave.

Figure 92. The input value varies with each counter period: first $TOP / 3$, then $2 \times TOP / 3$, and finally $TOP + 1$ for 100% duty cycle. Each increase in the input value causes a corresponding increase in the output duty cycle.



In Figure 92, the input value only changes at the instant where the counter wraps through 0. Figure 93 shows what happens if the input value is allowed to change at any other time: an unwanted glitch is produced at the output.

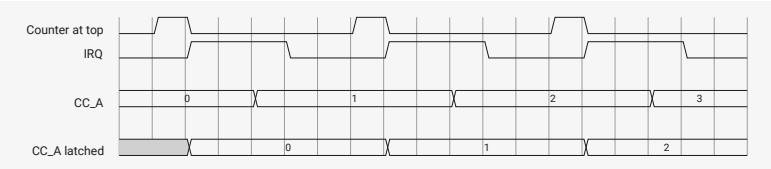
Figure 93. The input value changes whilst the counter is mid-ramp. This produces additional toggling at the output.



The behaviour becomes even more perplexing if the **TOP** register is also modified. It would be difficult for software to write to **CC** or **TOP** with the correct timing. To solve this, each slice has two copies of the **CC** and **TOP** registers: one copy which software can modify, and another, internal copy which is updated from the first register at the instant the counter wraps. Software can modify its copy of the register at will, but the changes are not captured by the PWM output until the next wrap.

Figure 94 shows the sequence of events where a software interrupt handler changes the value of **CC_A** each time the counter wraps.

Figure 94. Each counter wrap causes the interrupt request signal to assert. The processor enters its interrupt handler, writes to its copy of the CC register, and clears the interrupt. When the counter wraps again, the latched version of the CC register is instantaneously updated with the most recent value written by software, and this value controls the duty cycle for the next period. The IRQ is reasserted so that software can write another fresh value to its copy of the CC register.

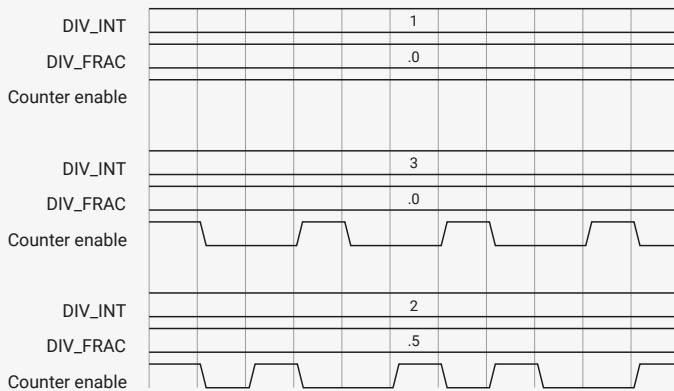


There is no limitation on what values can be written to CC or TOP, or when they are written. In normal PWM mode (CSR_PH_CORRECT is 0), the latched copies update when the counter wraps to 0, which occurs once every TOP + 1 cycles. In phase-correct mode (CSR_PH_CORRECT is 1), the latched copies update on the 0 to 0 count transition, when the counter stops counting downward and begins to count upward again.

12.5.2.4. Clock Divider

Each slice has a 8 integer bit, 4 fractional bit fractional clock divider configured by the DIV register. The clock divider allows you to slow the count rate by a factor of up to 256. To do this, the PWM generates an enable signal that gates counter operation. This allows you to achieve output frequencies significantly lower than the system clock. For instance, from a 125MHz system clock, the clock divider can slow the count rate to approximately 7.5Hz. Lower frequencies than this require a system timer interrupt (Section 12.8).

Figure 95. The clock divider generates an enable signal. The counter only counts on cycles where this signal is high. A clock divisor of 1 causes the enable to be asserted on every cycle, so the counter counts by one on every system clock cycle. Higher divisors cause the count enable to be asserted less frequently. Fractional division achieves an average fractional counting rate by spacing some enable pulses further apart than others.



The fractional divider is a first-order delta-sigma type.

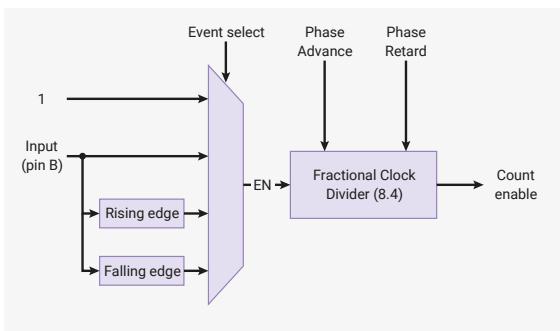
The clock divider also extends the effective count range when using level-sensitive or edge-sensitive modes to take duty cycle or frequency measurements.

12.5.2.5. Level-sensitive and Edge-sensitive Triggering

The PWM provides the following counter modes:

- Default free-running, counting continuously whenever the slice is enabled (free-running)
- Count continuously when a high level is detected on the B pin (level sensitive)
- Count once with each rising edge detected on the B pin (rising edge-sensitive)
- Count once with each falling edge detected on the B pin (falling edge-sensitive)

Figure 96. PWM slice event selection. The counter advances when its enable input is high. This enable is generated by two sequential stages. First, any one of four event types (always on, pin B high, pin B rise, pin B fall) can generate enable pulses for the fractional clock divider. The divider can reduce the rate of the enable pulses, before passing them on to the counter.



Use the `DIVMODE` field in each slice's `CSR` to select a mode. In free-running mode, the A and B pins are both outputs. In any other mode, the B pin becomes an input that controls counter operation. `CC_B` is ignored when not in free-running mode.

You can measure the duty cycle or frequency of an input signal by running the slice for a fixed amount of time in level-sensitive or edge-sensitive mode. Due to the type of edge-detect circuit used, the low period and high period of the measured signal must both be strictly greater than the system clock period when taking frequency measurements.

The clock divider still operates in level-sensitive and edge-sensitive modes. At maximum division (`DIV_INT` is 0), the counter only advances once per 256 high input cycles in level-sensitive modes, or once per 256 edges in edge-sensitive mode. This allows you to take longer-running measurements, although the resolution is still 16 bits.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/pwm/measure_duty_cycle/measure_duty_cycle.c Lines 19 - 37

```

19 float measure_duty_cycle(uint gpio) {
20     // Only the PWM B pins can be used as inputs.
21     assert(pwm_gpio_to_channel(gpio) == PWM_CHAN_B);
22     uint slice_num = pwm_gpio_to_slice_num(gpio);
23
24     // Count once for every 100 cycles the PWM B input is high
25     pwm_config cfg = pwm_get_default_config();
26     pwm_config_set_clkdiv_mode(&cfg, PWM_DIV_B_HIGH);
27     pwm_config_set_clkdiv(&cfg, 100);
28     pwm_init(slice_num, &cfg, false);
29     gpio_set_function(gpio, GPIO_FUNC_PWM);
30
31     pwm_set_enabled(slice_num, true);
32     sleep_ms(10);
33     pwm_set_enabled(slice_num, false);
34     float counting_rate = clock_get_hz(clk_sys) / 100;
35     float max_possible_count = counting_rate * 0.01;
36     return pwm_get_counter(slice_num) / max_possible_count;
37 }
```

12.5.2.6. Configuring PWM Period

When free-running, use the following three parameters to control the period of a PWM slice's output (measured in system clock cycles):

- The `TOP` register, which controls the maximum value of the counting period
- The `CSR_PH_CORRECT` bit, which enables phase-correct mode
- The `DIV` register, which controls the clock divider

The slice counts from 0 to `TOP`, then either wraps or begins counting backward, depending on the setting of `CSR_PH_CORRECT`. The clock divider slows the rate of counting, with a maximum speed of one count per cycle, and a minimum speed of one count per 256 cycles. Calculate the period in clock cycles with the following equation:

$$\text{period} = (\text{TOP} + 1) \times (\text{CSR_PH_CORRECT} + 1) \times \left(\text{DIV_INT} + \frac{\text{DIV_FRAC}}{16} \right)$$

To determine the output frequency based on the system clock frequency, use the following equation:

$$f_{\text{PWM}} = \frac{f_{\text{sys}}}{\text{period}} = \frac{f_{\text{sys}}}{(\text{TOP} + 1) \times (\text{CSR_PH_CORRECT} + 1) \times \left(\text{DIV_INT} + \frac{\text{DIV_FRAC}}{16} \right)}$$

Set `DIV_INT` to 0 to divide the count rate by the maximum possible value of 256. You must not set any `DIV_FRAC` bits when `DIV_INT` is 0.

12.5.2.7. Interrupt Request (IRQ) and DMA Data Request (DREQ)

The PWM block has two IRQ outputs. The interrupt status registers `INTR`, `INTS0`, `INTS1`, `INTE0` and `INTE1` allow software to:

- control which slices assert each of the two IRQs
- check which slices caused the assertion of an IRQ
- clear and acknowledge the interrupt

A slice generates an interrupt request each time its counter wraps (or, in phase-correct mode, each time the counter returns to 0). This sets the flag corresponding to this slice in the raw interrupt status register, `INTR`. If this slice's interrupt is enabled in `INTE`, this flag causes the PWM block's IRQ to be asserted, and the flag also appears in the masked interrupt status register `INTS`.

To clear flags, write a mask back to `INTR`. This is demonstrated in the LED fade SDK example below:

Pico Examples: https://asic-git.piowers.org/amethyst/pico-examples/-/blob/master/pwm/led_fade/pwm_led_fade.c

```

1 /**
2  * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
3 *
4  * SPDX-License-Identifier: BSD-3-Clause
5 */
6
7 // Fade an LED between low and high brightness. An interrupt handler updates
8 // the PWM slice's output level each time the counter wraps.
9
10 #include "pico/stdc.h"
11 #include <stdio.h>
12 #include "pico/time.h"
13 #include "hardware/irq.h"
14 #include "hardware/pwm.h"
15
16 #ifdef PICO_DEFAULT_LED_PIN
17 void on_pwm_wrap() {
18     static int fade = 0;
19     static bool going_up = true;
20     // Clear the interrupt flag that brought us here
21     pwm_clear_irq(pwm_gpio_to_slice_num(PICO_DEFAULT_LED_PIN));
22
23     if (going_up) {
24         ++fade;
25         if (fade > 255) {
26             fade = 255;
27             going_up = false;
28         }
29     } else {
30         --fade;
31         if (fade < 0) {
32             fade = 0;
33         }
34     }
35 }
36
37 int main() {
38     // Set up the PWM slice
39     // ...
40
41     // Set up the interrupt
42     // ...
43
44     // Start the fade
45     // ...
46
47     // Main loop
48     // ...
49
50     // Clean up
51     // ...
52
53     return 0;
54 }

```

```

33         going_up = true;
34     }
35 }
36 // Square the fade value to make the LED's brightness appear more linear
37 // Note this range matches with the wrap value
38 pwm_set_gpio_level(PICO_DEFAULT_LED_PIN, fade * fade);
39 }
40 #endif
41
42 // todo amy revisit
43 #if !PICO_RP2040
44 #define PWM_IRQ_WRAP PWM_IRQ_WRAP_0
45 #endif
46 int main() {
47 #ifndef PICO_DEFAULT_LED_PIN
48 #warning pwm/led_fade example requires a board with a regular LED
49 #else
50     // Tell the LED pin that the PWM is in charge of its value.
51     gpio_set_function(PICO_DEFAULT_LED_PIN, GPIO_FUNC_PWM);
52     // Figure out which slice we just connected to the LED pin
53     uint slice_num = pwm_gpio_to_slice_num(PICO_DEFAULT_LED_PIN);
54
55     // Mask our slice's IRQ output into the PWM block's single interrupt line,
56     // and register our interrupt handler
57     pwm_clear_irq(slice_num);
58     pwm_set_irq_enabled(slice_num, true);
59     irq_set_exclusive_handler(PWM_IRQ_WRAP, on_pwm_wrap);
60     irq_set_enabled(PWM_IRQ_WRAP, true);
61
62     // Get some sensible defaults for the slice configuration. By default, the
63     // counter is allowed to wrap over its maximum range (0 to 2**16-1)
64     pwm_config config = pwm_get_default_config();
65     // Set divider, reduces counter clock to sysclock/this value
66     pwm_config_set_clkdiv(&config, 4.f);
67     // Load the configuration into our PWM slice, and set it running.
68     pwm_init(slice_num, &config, true);
69
70     // Everything after this point happens in the PWM interrupt handler, so we
71     // can twiddle our thumbs
72     while (1)
73         tight_loop_contents();
74 #endif
75 }

```

This scheme allows multiple slices to generate interrupts concurrently. A system interrupt handler determines which slices caused the most recent interruption, and handles them appropriately. Normally, this means reloading those slices' **TOP** or **CC** registers, but the PWM block can also be used as a source of regular interrupt requests for non-PWM purposes.

The same pulse which sets the interrupt flag in **INTR** is also available as a one-cycle data request to the RP2350 system DMA. For each cycle the DMA sees a DREQ asserted, it makes one data transfer to its programmed location in as timely a manner as possible. Combined with the double-buffered behaviour of **CC** and **TOP**, the DMA can efficiently stream data to a PWM slice at a rate of one transfer per counter period. Alternatively, a PWM slice could serve as a pacing timer for DMA transfers to some other memory-mapped hardware.

12.5.2.8. On-the-fly Phase Adjustment

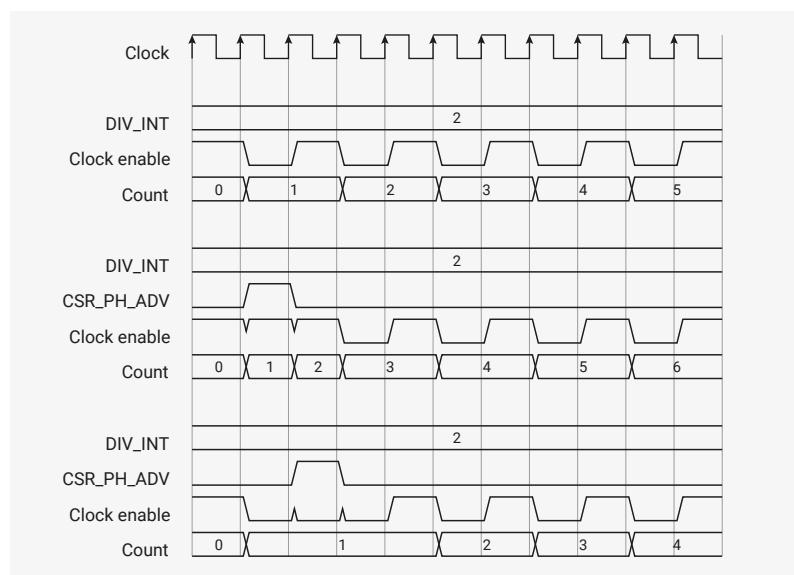
For some applications, it is necessary to control the phase relationship between two PWM outputs on different slices.

The global enable register **EN** contains an alias of the **CSR_EN** flag for each slice. Use this register to start and stop several slices simultaneously. If two slices with the same output frequency start at the same time, they run in perfect lockstep,

with a fixed phase relationship determined by the initial counter values.

The **CSR_PH_ADV** and **CSR_PH_RET** fields advance or retard a slice's output phase by one count whilst it is running. They do so by inserting or deleting pulses from the clock enable (the output of the clock divider), as shown in Figure 97.

Figure 97. The clock enable signal, output by the clock divider, controls the rate of counting. Phase advance forces the clock enable high on cycles where it is low, causing the counter to jump forward by one count. Phase retard forces the clock enable low when it would be high, holding the counter back by one count.



The counter cannot count faster than once per cycle, so **PH_ADV** requires **DIV_INT** > 1 or **DIV_FRAC** > 0. Likewise, the counter will not start to count backward if **PH_RET** is asserted when the clock enable is permanently low.

To advance or retard the phase by one count, software writes 1 to **PH_ADV** or **PH_RET**. Once an enable pulse has been inserted or deleted, the **PH_ADV** or **PH_RET** register bit returns to 0. Software can poll **CSR** until this happens. **PH_ADV** always inserts a pulse into the next available gap; **PH_RET** always deletes the next available pulse.

12.5.3. List of Registers

The PWM registers start at a base address of **0x400a8000** (defined as **PWM_BASE** in the SDK).

Table 1084. List of PWM registers

Offset	Name	Info
0x000	CH0_CSR	Control and status register
0x004	CH0_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x008	CH0_CTR	Direct access to the PWM counter
0x00c	CH0_CC	Counter compare values
0x010	CH0_TOP	Counter wrap value
0x014	CH1_CSR	Control and status register
0x018	CH1_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x01c	CH1_CTR	Direct access to the PWM counter
0x020	CH1_CC	Counter compare values
0x024	CH1_TOP	Counter wrap value
0x028	CH2_CSR	Control and status register

Offset	Name	Info
0x02c	CH2_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x030	CH2_CTR	Direct access to the PWM counter
0x034	CH2_CC	Counter compare values
0x038	CH2_TOP	Counter wrap value
0x03c	CH3_CSR	Control and status register
0x040	CH3_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x044	CH3_CTR	Direct access to the PWM counter
0x048	CH3_CC	Counter compare values
0x04c	CH3_TOP	Counter wrap value
0x050	CH4_CSR	Control and status register
0x054	CH4_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x058	CH4_CTR	Direct access to the PWM counter
0x05c	CH4_CC	Counter compare values
0x060	CH4_TOP	Counter wrap value
0x064	CH5_CSR	Control and status register
0x068	CH5_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x06c	CH5_CTR	Direct access to the PWM counter
0x070	CH5_CC	Counter compare values
0x074	CH5_TOP	Counter wrap value
0x078	CH6_CSR	Control and status register
0x07c	CH6_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x080	CH6_CTR	Direct access to the PWM counter
0x084	CH6_CC	Counter compare values
0x088	CH6_TOP	Counter wrap value
0x08c	CH7_CSR	Control and status register
0x090	CH7_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x094	CH7_CTR	Direct access to the PWM counter
0x098	CH7_CC	Counter compare values

Offset	Name	Info
0x09c	CH7_TOP	Counter wrap value
0x0a0	CH8_CSR	Control and status register
0x0a4	CH8_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x0a8	CH8_CTR	Direct access to the PWM counter
0x0ac	CH8_CC	Counter compare values
0x0b0	CH8_TOP	Counter wrap value
0x0b4	CH9_CSR	Control and status register
0x0b8	CH9_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x0bc	CH9_CTR	Direct access to the PWM counter
0x0c0	CH9_CC	Counter compare values
0x0c4	CH9_TOP	Counter wrap value
0x0c8	CH10_CSR	Control and status register
0x0cc	CH10_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x0d0	CH10_CTR	Direct access to the PWM counter
0x0d4	CH10_CC	Counter compare values
0x0d8	CH10_TOP	Counter wrap value
0x0dc	CH11_CSR	Control and status register
0x0e0	CH11_DIV	INT and FRAC form a fixed-point fractional number. Counting rate is system clock frequency divided by this number. Fractional division uses simple 1st-order sigma-delta.
0x0e4	CH11_CTR	Direct access to the PWM counter
0x0e8	CH11_CC	Counter compare values
0x0ec	CH11_TOP	Counter wrap value
0x0f0	EN	This register aliases the CSR_EN bits for all channels. Writing to this register allows multiple channels to be enabled or disabled simultaneously, so they can run in perfect sync. For each channel, there is only one physical EN register bit, which can be accessed through here or CHx_CSR.
0x0f4	INTR	Raw Interrupts
0x0f8	IRQ0_INTE	Interrupt Enable for irq0
0x0fc	IRQ0_INTF	Interrupt Force for irq0
0x100	IRQ0_INTS	Interrupt status after masking & forcing for irq0
0x104	IRQ1_INTE	Interrupt Enable for irq1
0x108	IRQ1_INTF	Interrupt Force for irq1

Offset	Name	Info
0x10c	IRQ1_INTS	Interrupt status after masking & forcing for irq1

PWM: CH0_CSR, CH1_CSR, ..., CH10_CSR, CH11_CSR Registers

Offsets: 0x000, 0x014, ..., 0x0c8, 0x0dc

Description

Control and status register

Table 1085. CH0_CSR, CH1_CSR, ..., CH10_CSR, CH11_CSR Registers

Bits	Name	Description	Type	Reset
31:8	Reserved.	-	-	-
7	PH_ADV	Advance the phase of the counter by 1 count, while it is running. Self-clearing. Write a 1, and poll until low. Counter must be running at less than full speed (div_int + div_frac / 16 > 1)	SC	0x0
6	PH_RET	Retard the phase of the counter by 1 count, while it is running. Self-clearing. Write a 1, and poll until low. Counter must be running.	SC	0x0
5:4	DIVMODE	0x0 → Free-running counting at rate dictated by fractional divider 0x1 → Fractional divider operation is gated by the PWM B pin. 0x2 → Counter advances with each rising edge of the PWM B pin. 0x3 → Counter advances with each falling edge of the PWM B pin.	RW	0x0
3	B_INV	Invert output B	RW	0x0
2	A_INV	Invert output A	RW	0x0
1	PH_CORRECT	1: Enable phase-correct modulation. 0: Trailing-edge	RW	0x0
0	EN	Enable the PWM channel.	RW	0x0

PWM: CH0_DIV, CH1_DIV, ..., CH10_DIV, CH11_DIV Registers

Offsets: 0x004, 0x018, ..., 0x0cc, 0x0e0

Description

INT and FRAC form a fixed-point fractional number.

Counting rate is system clock frequency divided by this number.

Fractional division uses simple 1st-order sigma-delta.

Table 1086. CH0_DIV, CH1_DIV, ..., CH10_DIV, CH11_DIV Registers

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:4	INT		RW	0x01
3:0	FRAC		RW	0x0

PWM: CH0_CTR, CH1_CTR, ..., CH10_CTR, CH11_CTR Registers

Offsets: 0x008, 0x01c, ..., 0x0d0, 0x0e4

Table 1087. CH0_CTR, ...
CH1_CTR, ..., CH10_CTR, CH11_CTR
Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Direct access to the PWM counter	RW	0x0000

PWM: CH0_CC, CH1_CC, ..., CH10_CC, CH11_CC Registers

Offsets: 0x00c, 0x020, ..., 0x0d4, 0x0e8

Description

Counter compare values

Table 1088. CH0_CC, ...
CH1_CC, ..., CH10_CC, CH11_CC Registers

Bits	Name	Description	Type	Reset
31:16	B		RW	0x0000
15:0	A		RW	0x0000

PWM: CH0_TOP, CH1_TOP, ..., CH10_TOP, CH11_TOP Registers

Offsets: 0x010, 0x024, ..., 0x0d8, 0x0ec

Table 1089. CH0_TOP, ...
CH1_TOP, ..., CH10_TOP, CH11_TOP Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Counter wrap value	RW	0xffff

PWM: EN Register

Offset: 0x0f0

Description

This register aliases the CSR_EN bits for all channels.

Writing to this register allows multiple channels to be enabled or disabled simultaneously, so they can run in perfect sync.

For each channel, there is only one physical EN register bit, which can be accessed through here or CHx_CSR.

Table 1090. EN Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		RW	0x0
10	CH10		RW	0x0
9	CH9		RW	0x0
8	CH8		RW	0x0
7	CH7		RW	0x0
6	CH6		RW	0x0
5	CH5		RW	0x0
4	CH4		RW	0x0
3	CH3		RW	0x0
2	CH2		RW	0x0
1	CH1		RW	0x0

Bits	Name	Description	Type	Reset
0	CH0		RW	0x0

PWM: INTR Register

Offset: 0x0f4

Description

Raw Interrupts

Table 1091. INTR Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		WC	0x0
10	CH10		WC	0x0
9	CH9		WC	0x0
8	CH8		WC	0x0
7	CH7		WC	0x0
6	CH6		WC	0x0
5	CH5		WC	0x0
4	CH4		WC	0x0
3	CH3		WC	0x0
2	CH2		WC	0x0
1	CH1		WC	0x0
0	CH0		WC	0x0

PWM: IRQ0_INTE Register

Offset: 0x0f8

Description

Interrupt Enable for irq0

Table 1092. IRQ0_INTE Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		RW	0x0
10	CH10		RW	0x0
9	CH9		RW	0x0
8	CH8		RW	0x0
7	CH7		RW	0x0
6	CH6		RW	0x0
5	CH5		RW	0x0
4	CH4		RW	0x0
3	CH3		RW	0x0
2	CH2		RW	0x0

Bits	Name	Description	Type	Reset
1	CH1		RW	0x0
0	CH0		RW	0x0

PWM: IRQ0_INTF Register

Offset: 0x0fc

Description

Interrupt Force for irq0

Table 1093.
IRQ0_INTF Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		RW	0x0
10	CH10		RW	0x0
9	CH9		RW	0x0
8	CH8		RW	0x0
7	CH7		RW	0x0
6	CH6		RW	0x0
5	CH5		RW	0x0
4	CH4		RW	0x0
3	CH3		RW	0x0
2	CH2		RW	0x0
1	CH1		RW	0x0
0	CH0		RW	0x0

PWM: IRQ0_INTS Register

Offset: 0x100

Description

Interrupt status after masking & forcing for irq0

Table 1094.
IRQ0_INTS Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		RO	0x0
10	CH10		RO	0x0
9	CH9		RO	0x0
8	CH8		RO	0x0
7	CH7		RO	0x0
6	CH6		RO	0x0
5	CH5		RO	0x0
4	CH4		RO	0x0
3	CH3		RO	0x0

Bits	Name	Description	Type	Reset
2	CH2		RO	0x0
1	CH1		RO	0x0
0	CHO		RO	0x0

PWM: IRQ1_INTE Register

Offset: 0x104

Description

Interrupt Enable for irq1

Table 1095.
IRQ1_INTE Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		RW	0x0
10	CH10		RW	0x0
9	CH9		RW	0x0
8	CH8		RW	0x0
7	CH7		RW	0x0
6	CH6		RW	0x0
5	CH5		RW	0x0
4	CH4		RW	0x0
3	CH3		RW	0x0
2	CH2		RW	0x0
1	CH1		RW	0x0
0	CHO		RW	0x0

PWM: IRQ1_INTF Register

Offset: 0x108

Description

Interrupt Force for irq1

Table 1096.
IRQ1_INTF Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		RW	0x0
10	CH10		RW	0x0
9	CH9		RW	0x0
8	CH8		RW	0x0
7	CH7		RW	0x0
6	CH6		RW	0x0
5	CH5		RW	0x0
4	CH4		RW	0x0

Bits	Name	Description	Type	Reset
3	CH3		RW	0x0
2	CH2		RW	0x0
1	CH1		RW	0x0
0	CH0		RW	0x0

PWM: IRQ1_INTS Register

Offset: 0x10c

Description

Interrupt status after masking & forcing for irq1

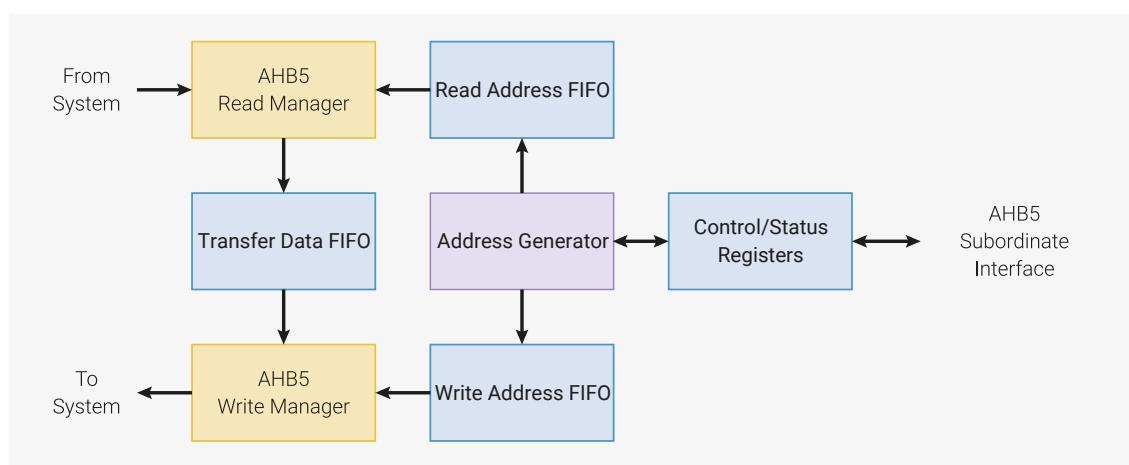
Table 1097.
IRQ1_INTS Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	CH11		RO	0x0
10	CH10		RO	0x0
9	CH9		RO	0x0
8	CH8		RO	0x0
7	CH7		RO	0x0
6	CH6		RO	0x0
5	CH5		RO	0x0
4	CH4		RO	0x0
3	CH3		RO	0x0
2	CH2		RO	0x0
1	CH1		RO	0x0
0	CH0		RO	0x0

12.6. DMA

The RP2350 Direct Memory Access (DMA) controller performs bulk data transfers on a processor's behalf. This leaves processors free to attend to other tasks or enter low-power sleep states. The DMA dual bus manager ports can issue one read and one write access per cycle. The data throughput is therefore far greater than one of RP2350's processors.

Figure 98. DMA Architecture Overview. The read manager can read data from some address every clock cycle. Likewise, the write manager can write to another address. The address generator produces matched pairs of read and write addresses, which the managers consume through the address FIFOs. The DMA can run up to 16 transfer sequences simultaneously, supervised by software via the control and status registers.



The DMA can perform one read access and one write access, up to 32 bits in size, every clock cycle. There are 16 independent channels, each of which supervises a sequence of bus transfers in one of the following scenarios:

Memory-to-peripheral

a peripheral signals the DMA when it needs more data to transmit. The DMA reads data from an array in RAM or flash, and writes to the peripheral's data FIFO.

Peripheral-to-memory

a peripheral signals the DMA when it has received data. The DMA reads this data from the peripheral's data FIFO, and writes it to an array in RAM.

Memory-to-memory

the DMA transfers data between two buffers in RAM, as fast as possible.

Each channel has its own control and status registers (CSRs) that software can use to program and monitor the channel's progress. When multiple channels are active at the same time, the DMA shares bandwidth evenly between the channels, with round-robin over all channels which are currently requesting data transfers.

The transfer size can be either 32, 16, or 8 bits. This is configured once per channel: source transfer size and destination transfer size are the same. The DMA performs byte lane replication on narrow writes, so byte data is available in all 4 bytes of the databus, and halfword data in both halfwords.

Channels can be combined in varied ways for more sophisticated behaviour and greater autonomy. For example, one channel can configure another, loading configuration data from a sequence of control blocks in memory, and the second can then call back to the first via the `CHAIN_TO` option when it needs to be reconfigured.

Making the DMA more autonomous means that much less processor supervision is required: overall this allows the system to do more at once, or to dissipate less power.

12.6.1. Changes from RP2040

The following new features have been added:

- Increased the number of DMA channels from 12 to 16.
- Increased the number of shared IRQ outputs from 2 to 4.
- Channels can be assigned to security domains using `SECCFG_CH0` through `SECCFG_CH15`.
- The DMA now filters bus accesses using the built-in memory protection unit (Section 12.6.3).
- Interrupts can be assigned to security domains using `SECCFG_IRQ0` through `SECCFG_IRQ3`.
- Pacing timers and the CRC sniffer can be assigned to security domains using the `SECCFG_MISC` register.
- The four most-significant bits of `TRANS_COUNT` (`CH0_TRANS_COUNT`) are redefined as the `MODE` field, which defines what happens when `TRANS_COUNT` reaches zero:

- This backward-incompatible change reduces the maximum transfers in one sequence from $2^{32}-1$ to $2^{28}-1$.
- Mode **0x0** has the same behaviour as RP2040, so there is no need to modify software that performs less than 256 million transfers at a time.
- Mode **0x1**, "trigger self", allows a channel to automatically restart itself after finishing a transfer sequence, in addition to the usual end-of-sequence actions like raising an interrupt or triggering other channels. This can be used for example to get periodic interrupts from streaming ring buffer transfers.
- Mode **0xf**, "endless", allows a channel to run forever: **TRANS_COUNT** does not decrement.
- New **CH0_CTRL_TRIG.INCR_READ_REV** and **CH0_CTRL_TRIG.INCR_WRITE_REV** fields allow addresses to decrement rather than increment, or to increment by two.

Some existing behaviour has been refined:

- The logic which adjusts values read from **WRITE_ADDR** and **READ_ADDR** according to the number of in-flight transfers is disabled for address-wrapping and non-incrementing transfers (erratum RP2040-E12).
- You can now poll the **ABORT** register to wait for completion of an aborted channel (erratum RP2040-E13).
- DMA completion actions such as **CHAIN_TO** are now strictly ordered against the last write completion, so a **CHAIN_TO** on a channel whose registers you write to is a well-defined operation.
 - This enables the use of control blocks which do not include one of the four trigger register aliases.
 - Previously, a channel was considered to complete on the *first* cycle of its last write's data phase. Now, a channel is considered to complete on the *last* cycle of its last write's data phase. This is usually the same cycle, but it can be later when the DMA encounters a write data-phase bus stall.
- Previously, the DMA's internal arbitration logic inserted an idle cycle after completing a round of active high-priority channels (**CH0_CTRL_TRIG.HIGH_PRIORITY**), even if there were no active low-priority requests. This reduced DMA throughput when lightly loaded. This idle cycle has been removed, eliminating lost throughput.
- IRQ assertion latency has been reduced by one cycle.

12.6.2. Configuring Channels

Each channel has four control/status registers:

- **READ_ADDR (CH0_READ_ADDR)** is the address of the next memory location to read.
- **WRITE_ADDR (CH0_WRITE_ADDR)** is the address of the next memory location to write.
- **TRANS_COUNT (CH0_TRANS_COUNT)** shows the number of transfers remaining in the current transfer sequence and programs the number of transfers in the next transfer sequence (see [Section 12.6.2.2](#)).
- **CTRL (CH0_CTRL_TRIG)** configures all other aspects of the channel's behaviour, enables/disables the channel, and provides completion status.

To directly instruct the DMA channel to perform a data transfer, software writes to these four registers, and then triggers the channel ([Section 12.6.3](#)). To make the DMA more autonomous, you can also program one DMA channel to write to another channel's configuration registers, queueing up many transfer sequences in advance.

All four are live registers; they update their status continuously as the channel progresses.

12.6.2.1. Read and Write Addresses

READ_ADDR and **WRITE_ADDR** contain the address the channel will next read from, and write to, respectively. These registers update automatically after each read/write access, incrementing to the next read/write address as required. The size of the increment varies according to:

- the transfer size: 1, 2 or 4 byte bus accesses as per **CH0_CTRL_TRIG.DATA_SIZE**

- the increment enable for each address register: `CH0_CTRL_TRIG.INCR_READ` and `CH0_CTRL_TRIG.INCR_WRITE`
- the increment direction: `CH0_CTRL_TRIG.INCR_READ_REV` and `CH0_CTRL_TRIG.INCR_WRITE_REV`

Software should generally program these registers with new start addresses each time a new transfer sequence starts. If `READ_ADDR` and `WRITE_ADDR` are not reprogrammed, the DMA will use the current values as start addresses for the next transfer. For example:

- If the address does not increment (e.g. it is the address of a peripheral FIFO), and the next transfer sequence is to/from that *same* address, there is no need to write to the register again.
- When transferring to/from a consecutive series of buffers in memory (e.g. scattering and gathering), an address register will already have incremented to the start of the next buffer at the completion of a transfer.

By not programming all four CSRs for each transfer sequence, software can use shorter interrupt handlers, and more compact control block formats when used with channel chaining (see register aliases in [Section 12.6.3.1](#), chaining in [Section 12.6.3.2](#)).

12.6.2.1.1. Address Alignment

`READ_ADDR` and `WRITE_ADDR` must be aligned to the transfer size, specified in `CH0_CTRL_TRIG.DATA_SIZE`. For 32-bit transfers, the address must be a multiple of four, and for 16-bit transfers, the address must be a multiple of two. Software is responsible for correctly aligning addresses written to `READ_ADDR` and `WRITE_ADDR`: the DMA does not enforce alignment.

If software initially writes a correctly aligned address, the address will remain correctly aligned throughout the transfer sequence, because the DMA always increments `READ_ADDR` and `WRITE_ADDR` by a multiple of the transfer size. Specifically, it increments by transfer size times -1, 0, 1 or 2, depending on the values of `CH0_CTRL_TRIG.INCR_READ`, `CH0_CTRL_TRIG.INCR_WRITE`, `CH0_CTRL_TRIG.INCR_READ_REV` and `CH0_CTRL_TRIG.INCR_WRITE_REV`.

The DMA MPU and system-level bus security filters perform protection checks on the lowest byte address of all bytes transferred on a given cycle (i.e. to the present value of `READ_ADDR/WRITE_ADDR`). RP2350 memory hardware ensures unaligned bus accesses do not cause data to be read/written from the other side of a protection boundary. This means that unaligned access can not be used to violate the memory protection model. Other than this, the result of an unaligned access is unspecified.

12.6.2.2. Transfer Count

Reading `TRANS_COUNT` (`CH0_TRANS_COUNT`) returns the number of transfers remaining in the current transfer sequence. This value updates continuously as the channel progresses. Writing to `TRANS_COUNT` sets the length of the *next* transfer sequence. Up to $2^{28}-1$ transfers can be performed in one sequence (`0x0fffffff`, approximately 256 million).

Each time the channel starts a new transfer sequence, the most recent value written to `TRANS_COUNT` is copied to the live transfer counter, which will then start to decrement again as the new transfer sequence makes progress. For debugging purposes, the `DBG_TCR` (`TRANS_COUNT` reload value) registers display the last value written to each channel's `TRANS_COUNT`.

If the channel is triggered multiple times without intervening writes to `TRANS_COUNT`, it performs the same number of transfers each time. For example, when chained to, one channel might load a fixed-size control block into another channel's CSRs. `TRANS_COUNT` would be programmed once by software, and then reload automatically every time.

Alternatively, `TRANS_COUNT` can be written with a new value before starting each transfer sequence. If `TRANS_COUNT` is the channel trigger (see [Section 12.6.3.1](#)), the channel will start immediately, and the value just written will be used, *not* the value currently in the reload register.

NOTE

The `TRANS_COUNT` is the number of *transfers* to be performed. The total number of bytes transferred is `TRANS_COUNT` times the size of each transfer in bytes, given by `CTRL.DATA_SIZE`.

12.6.2.2.1. Count Modes

The four most-significant bits of `TRANS_COUNT` contain the `MODE` field (`CHO_TRANS_COUNT.MODE`), which modifies the counting behaviour of `TRANS_COUNT`. Mode `0x0` is the default: `TRANS_COUNT` decrements once for every bus transfer, and the channel halts once `TRANS_COUNT` reaches zero and all in-flight transfers have finished. The value of `0x0` is chosen for backward-compatibility with RP2040 software, which expects the `TRANS_COUNT` register to contain a 32-bit count rather than a 4-bit mode and a 28-bit count. There are few use cases for a *finite* number of transfers greater than 2^{28} , which is why the four most-significant bits have been reallocated for use with endless transfers.

Mode `0x1`, `TRIGGER_SELF`, behaves the same as mode `0x0`, except that rather than halting upon completion, the channel immediately re-triggers itself. This is equivalent to a trigger performed by any other mechanism (Section 12.6.3): `TRANS_COUNT` is reloaded, and the channel resumes from the current `READ_ADDR` and `WRITE_ADDR` addresses. A completion interrupt is still raised (if `CTRL.IRQ QUIET` is not set) and the specified `CHAIN_TO` operation is still performed. The main use for this mode is streaming through SRAM ring buffers, where some action is required at regular intervals, for example requesting the processor to refill an audio buffer once it is half-empty.

Mode `0xf`, `ENDLESS`, disables the decrement of `TRANS_COUNT`. This means a channel will generally run indefinitely without pause, though triggering a channel with a mode of `0xf` and a count of `0x0` will result in the channel halting immediately.

All other values are reserved for future use and their effect is unspecified.

12.6.2.3. Control/Status

The `CTRL` register (`CHO_CTRL_TRIG`) has more, smaller fields than the other 3 registers. Among other things, `CTRL` is used to:

- Configure the size of this channel's data transfers, via the `DATA_SIZE` field. Reads are always the same size as writes.
- Configure if and how `READ_ADDR` and `WRITE_ADDR` increment after each read or write, via the `INCR_READ`, `INCR_READ_REV`, `INCR_WRITE`, `INCR_WRITE_REV`, `RING_SEL` and `RING_SIZE` fields. Ring transfers are available, where one of the address pointers wraps at some power-of-2 boundary.
- Select another channel (or none) to trigger when this channel completes, via the `CHAIN_TO` field.
- Select a peripheral data request (DREQ) signal to pace this channel's transfers, via the `TREQ_SEL` field.
- See when the channel is idle, using the `BUSY` flag.
- See if the channel has encountered a bus error the `READ_ERROR` and `WRITE_ERROR` flags, or the combined error status in the `AHB_ERROR` flag.

12.6.3. Triggering Channels

Once a channel has been correctly configured, you must trigger it. This instructs the channel to begin scheduling bus accesses, either paced by a peripheral data request signal (DREQ) or as fast as possible. The following events can trigger a channel:

- A write to a channel trigger register.
- Completion of another channel whose `CHAIN_TO` points to this channel.
- A write to the `MULTI_CHAN_TRIGGER` register (can trigger multiple channels at once).

Each trigger mechanism covers different use cases. For example, trigger registers are simple and efficient when

configuring and starting a channel in an interrupt service routine because the channel is triggered by the last configuration write. `CHAIN_TO` allows one channel to callback to another channel, which can then reconfigure the first channel. `MULTI_CHAN_TRIGGER` allows software to simply start a channel without touching any of its configuration registers.

Once triggered, the channel sets its `CTRL.BUSY` flag to indicate it is actively scheduling transfers. This remains set until the transfer count reaches zero, or the channel is aborted via the `CHAN_ABORT` register (Section 12.6.8.3).

When a channel is already running, indicated by `BUSY = 1`, it ignores additional triggers. A channel which is disabled (`CTRL.EN` is clear) also ignores triggers.

12.6.3.1. Aliases and Triggers

Table 1098. Control register aliases. Each channel has four control/status registers. Each register can be accessed at multiple different addresses. In each naturally-aligned group of four, all four registers appear, in different orders.

Offset	+0x0	+0x4	+0x8	+0xc (Trigger)
0x00 (Alias 0)	READ_ADDR	WRITE_ADDR	TRANS_COUNT	CTRL_TRIG
0x10 (Alias 1)	CTRL	READ_ADDR	WRITE_ADDR	TRANS_COUNT_TRIG
0x20 (Alias 2)	CTRL	TRANS_COUNT	READ_ADDR	WRITE_ADDR_TRIG
0x30 (Alias 3)	CTRL	WRITE_ADDR	TRANS_COUNT	READ_ADD_TRIG

The four CSRs are aliased multiple times in memory. Each of the four aliases exposes the same four physical registers, but in a different order. The final register in each alias (at offset +0xc, highlighted) is a trigger register. Writing to the trigger register starts the channel.

Often, only alias 0 is used, and aliases 1 through 3 can be ignored. To configure and start the channel, write `READ_ADDR`, `WRITE_ADDR`, `TRANS_COUNT`, and finally `CTRL`. Since `CTRL` is the trigger register in alias 0, this starts the channel.

The other aliases allow more compact control block lists when using one channel to configure another, and more efficient reconfiguration and launch in interrupt handlers:

- Each CSR is a trigger register in one of the aliases:
 - When gathering fixed-size buffers into a peripheral, the DMA channel can be configured and launched by writing only `READ_ADDR_TRIG`.
 - When scattering from a peripheral to fixed-size buffers, the channel can be configured and launched by writing only `WRITE_ADDR_TRIG`.
- Useful combinations of registers appear as naturally-aligned tuples which contain a trigger register. In conjunction with channel chaining and address wrapping, these implement compressed control block formats, e.g.:
 - (`WRITE_ADDR`, `TRANS_COUNT_TRIG`) for peripheral scatter operations
 - (`TRANS_COUNT`, `READ_ADDR_TRIG`) for peripheral gather operations, or calculating CRCs on a list of buffers
 - (`READ_ADDR`, `WRITE_ADDR_TRIG`) for manipulating fixed-size buffers in memory

Trigger registers do not start the channel if:

- The channel is disabled via `CTRL.EN` (if the trigger is `CTRL`, the just-written value of `EN` is used, not the value currently in the `CTRL` register)
- The channel is already running
- The value 0 is written to the trigger register (useful for ending control block chains, see null triggers (Section 12.6.3.3))
- The bus access has a security level lower than the channel's security level (Section 12.6.6.1)

12.6.3.2. Chaining

When a channel completes, it can name a different channel to immediately be triggered. This can be used as a callback for the second channel to reconfigure and restart the first.

This feature is configured through the `CHAIN_TO` field in the channel `CTRL` register. This 4-bit value selects a channel that will start when this one finishes. A channel cannot chain to itself. Setting `CHAIN_TO` to a channel's own index prevents chaining.

Chain triggers behave the same as triggers from other sources, such as trigger registers. For example, they cause `TRANS_COUNT` to reload, and they are ignored if the targeted channel is already running.

One application for `CHAIN_TO` is for a channel to request reconfiguration by another channel from a sequence of control blocks in memory. Channel A is configured to perform a wrapped transfer from memory to channel B's control registers (including a trigger register), and channel B is configured to chain back to channel A when it completes each transfer sequence. This is shown explicitly in the DMA control blocks example (Section 12.6.9.2).

Use of the register aliases (Section 12.6.3.1) enables compact formats for DMA control blocks: as little as one word, in some cases.

Another use of chaining is a *ping-pong* configuration, where two channels each trigger one another. The processor can respond to the channel completion interrupts and reconfigure each channel after it completes. However, the chained channel, which has already been configured, starts immediately. In other words, channel configuration and channel operation are pipelined. This can improve performance dramatically when a usage pattern requires many short transfer sequences.

The Section 12.6.9 goes into more detail on the possibilities of chain triggers in the real world.

12.6.3.3. Null Triggers and Chain Interrupts

As mentioned in Section 12.6.3.1, writing all-zeroes to a trigger register does not start the channel. This is called a null trigger, and it has two purposes:

- Cause a halt at the end of an array of control blocks, by appending an all-zeroes block.
- Reduce the number of interrupts generated when using control blocks.

By default, channels generate an interrupt each time they finish a transfer sequence, unless that channel's IRQ is masked in `INTE0` through `INTE3`. The rate of interrupts can be excessive, particularly as processor attention is generally not required while a sequence of control blocks are in progress. However, processor attention is required at the end of a chain.

The channel `CTRL` register has a field called `IRQ QUIET`. Its default value is 0. When this set to 1, channels generate an interrupt when they receive a null trigger, but not on normal completion of a transfer sequence. The interrupt is generated by the channel which receives the trigger.

12.6.4. Data Request (DREQ)

Peripherals produce or consume data at their own pace. If the DMA transferred data as fast as possible, loss or corruption of data would ensue. DREQs are a communication channel between peripherals and the DMA which enables the DMA to pace transfers according to the needs of the peripheral.

The `CTRL.TREQ_SEL` (transfer request) field selects an external DREQ. It can also be used to select one of the internal pacing timers, or select no TREQ at all (the transfer proceeds as fast as possible), e.g. for memory-to-memory transfers.

12.6.4.1. System DREQ Table

DREQ numbers use the following global assignment to peripheral DREQ channels:

Table 1099. DREQs

DREQ	DREQ Channel	DREQ	DREQ Channel	DREQ	DREQ Channel	DREQ	DREQ Channel
0	DREQ_PIO0_TX0	14	DREQ_PIO1_RX2	28	DREQ_UART0_TX	42	DREQ_PWM_WRAP10
1	DREQ_PIO0_TX1	15	DREQ_PIO1_RX3	29	DREQ_UART0_RX	43	DREQ_PWM_WRAP11
2	DREQ_PIO0_TX2	16	DREQ_PIO2_TX0	30	DREQ_UART1_TX	44	DREQ_I2C0_TX
3	DREQ_PIO0_TX3	17	DREQ_PIO2_TX1	31	DREQ_UART1_RX	45	DREQ_I2C0_RX
4	DREQ_PIO0_RX0	18	DREQ_PIO2_TX2	32	DREQ_PWM_WRAP0	46	DREQ_I2C1_TX
5	DREQ_PIO0_RX1	19	DREQ_PIO2_TX3	33	DREQ_PWM_WRAP1	47	DREQ_I2C1_RX
6	DREQ_PIO0_RX2	20	DREQ_PIO2_RX0	34	DREQ_PWM_WRAP2	48	DREQ_ADC
7	DREQ_PIO0_RX3	21	DREQ_PIO2_RX1	35	DREQ_PWM_WRAP3	49	DREQ_XIP_STREAM
8	DREQ_PIO1_TX0	22	DREQ_PIO2_RX2	36	DREQ_PWM_WRAP4	50	DREQ_XIP_QMITX
9	DREQ_PIO1_TX1	23	DREQ_PIO2_RX3	37	DREQ_PWM_WRAP5	51	DREQ_XIP_QMIRX
10	DREQ_PIO1_TX2	24	DREQ_SPI0_TX	38	DREQ_PWM_WRAP6	52	DREQ_HSTX
11	DREQ_PIO1_TX3	25	DREQ_SPI0_RX	39	DREQ_PWM_WRAP7	53	DREQ_CORESIGHT
12	DREQ_PIO1_RX0	26	DREQ_SPI1_TX	40	DREQ_PWM_WRAP8	54	DREQ_SHA256
13	DREQ_PIO1_RX1	27	DREQ_SPI1_RX	41	DREQ_PWM_WRAP9		

12.6.4.2. Credit-based DREQ Scheme

The RP2350 DMA is designed for systems where:

- The area and power cost of large peripheral data FIFOs is prohibitive.
- The bandwidth demands of individual peripherals may be high, e.g. >50% bus injection rate for short periods.
- Bus latency is low, but multiple managers may compete for bus access.

In addition, the DMA's transfer FIFOs and dual-manager-port structure permit multiple accesses to the same peripheral to be in-flight at once to improve throughput. Choice of DREQ mechanism is therefore critical:

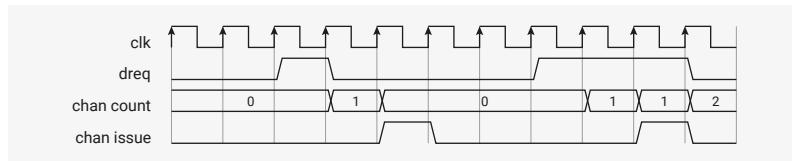
- The traditional "turn on the tap" method can cause overflow if multiple writes are backed up in the TDF. Some systems solve this by over-provisioning peripheral FIFOs and setting the DREQ threshold below the full level at the expense of precious area and power.
- The Arm-style single and burst handshake does not permit additional requests to be registered while the current request is being served. This limits performance when FIFOs are very shallow.

The RP2350 DMA uses a credit-based DREQ mechanism. For each peripheral, the DMA attempts to keep as many transfers in-flight as the peripheral has capacity for. This enables full bus throughput (1 word per clock) through an 8-deep peripheral FIFO with no possibility of overflow or underflow in the absence of fabric latency or contention.

For each channel, the DMA maintains a counter. Each 1-clock pulse on the `dreq` signal increments this counter. When non-zero, the channel requests a transfer from the DMA's internal arbiter. The counter decrements when the transfer is issued to the address FIFOs. At this point the transfer is in flight, but has not yet necessarily completed.

The counter is saturating, and six bits in size. The counter ignores increments at the maximum value or decrements at zero. The six-bit counter size supports counts up to the depth of any FIFO on RP2350.

Figure 99. DREQ counting



The effect is to upper bound the number of in-flight transfers based on the amount of room or data available in the peripheral FIFO. In the steady state, this gives maximum throughput, but can't underflow or overflow. This approach has the following caveats:

- The user *must not* access a FIFO currently being serviced by the DMA. This causes the channel and peripheral to become desynchronised, and can cause corruption or loss of data.
- Multiple channels *must not* be connected to the same DREQ.

12.6.5. Interrupts

Each channel can generate interrupts; these can be masked on a per-channel basis using one of the four identical interrupt enable registers, `INTE0` through `INTE3`. There are three circumstances where a channel raises an interrupt request:

- On the completion of each transfer sequence, if `CTRL.IRQ_QUIET` is disabled
- On receiving a null trigger, if `CTRL.IRQ_QUIET` is enabled
- On a read or write bus error

The masked interrupt status is visible in the `INTS` registers; there is one bit for each channel. Interrupts are cleared by writing a bit mask to `INTS`. One idiom for acknowledging interrupts is to read `INTS`, then write the same value back, so only enabled interrupts are cleared.

The RP2350 DMA provides four system IRQs, with independent masking and status registers (e.g. `INTE0`, `INTE1`). Any combination of channel interrupt requests can be routed to each system IRQ, though generally software only routes each channel interrupt to a single system IRQ. For example:

- Some channels can be given a higher priority in the system interrupt controller, if they have particularly tight timing requirements.
- In multiprocessor systems, different channel interrupts can be routed independently to different cores.
- When channels are assigned to a mixture of security domains, IRQs can also be assigned, so that software in each security domain can get interrupts from its own channels.

For debugging purposes, the `INTF` registers can force any channel interrupt to be asserted, which will cause assertion of any system IRQs which have that channel interrupt's enable bit set in their respective `INTE` registers.

12.6.6. Security

RP2350's processors support partitioning of memory and peripherals into multiple security domains. This partitioning is extended into the DMA, so that different security contexts can safely use their assigned channels without breaking any of the security invariants laid out by the processor security model. For example, an Arm processor in the Non-secure state must not be able to use the DMA to access memory or peripherals owned by Secure software.

The DMA defines four **security levels** which map onto Arm or RISC-V processor security states:

- 3: SP (secure and privileged)
 - Equivalent to Arm processors in the Secure, Privileged state
 - Equivalent to RISC-V processors in Machine mode
- 2: SU (secure and unprivileged)

- Equivalent to Arm processors in the Secure, Normal state
- 1: NSP (nonsecure and privileged)
 - Equivalent to Arm processors in the Non-secure, Privileged state
 - Equivalent to RISC-V processors in Supervisor mode
- 0: NSU (nonsecure and unprivileged)
 - Equivalent to Arm processors in the Non-secure, Normal state
 - Equivalent to RISC-V processors in User mode

So that the DMA can compare different security levels in a consistent way, they are considered ordered, with SP > SU > NSP > NSU. For example, when we say that a channel requires a *minimum* of SU to access its registers, this means that SP and SU are acceptable, and NSP and NSU are not. As a rule, every action has a reaction that is *at or below* the security level of the original action, and so the DMA can not be used to escalate accesses to a higher security level.

Software assigns internal DMA resources, like channels, interrupts, pacing timers and the CRC sniffer, to one of the four possible security levels. These resources are then accessible only at and above that level. Channel assignment in particular is discussed in [Section 12.6.6.1](#).

The DMA memory protection unit ([Section 12.6.6.3](#)) defines the minimum security level required to access up to eight programmable address ranges, so that channels of a given security level can not access memory beyond their means. This MPU is intended to mirror the SRAM and XIP memory protection boundaries configured in the processor SAU or PMP. In addition to the internal filtering performed by the DMA MPU, accesses are filtered by the system bus according to the [ACCESSCTRL](#) filter rules described in [Section 10.5.2](#).

The combination of these features allows the DMA to be safely shared by software running in different security domains. If this is not desired, the entire DMA block can instead be assigned wholesale to a single security domain using the [ACCESSCTRL DMA](#) register.

12.6.6.1. Channel Security Assignment

Channels are assigned to security domains using the channel [SECCFG](#) registers, [SECCFG_CH0](#) through [SECCFG_CH15](#). There is one register per channel. Each register contains a 2-bit security level, and a lock bit which prevents that [SECCFG](#) register from being changed once configured. At reset, all channels are assigned to the SP security level, which is the highest.

The security level of a channel defines:

- The security level of bus transfers performed by this channel, which is checked against both the DMA memory protection unit and the [ACCESSCTRL](#) bus-level filters described in [Section 10.5.2](#)
- The minimum security level required to read or write this channel's registers: access from a lower level returns a bus fault
- The minimum security level which must be defined on a shared IRQ line for that IRQ to be able to observe this channel's interrupts ([Section 12.6.6.2](#)), or for this channel's interrupt to be set/cleared through that IRQ's registers
- The minimum bus security level required to clear this channel's interrupts through the [INTR](#) register
- Which DREQs a channel can observe: channels assigned to the NSP or NSU security levels can not observe DREQs of Secure-only peripherals (as defined by the [ACCESSCTRL](#) peripheral configuration)
- Which pacing timer TREQs can be observed: pacing timer security levels are configured by [SECCFG_MISC](#) and must be no higher than the channel security level in order for the channel to observe the TREQ
- Whether the channel is visible to the CRC sniffer: the sniffer's security level is configured by [SECCFG_MISC](#) and must be no lower than the observed channel's security level
- Which channels this channel can trigger with a [CHAIN_TO](#): chaining from lower to higher security levels is not permitted

- The minimum bus security level required to trigger this channel with a write to [MULTI_CHAN_TRIGGER](#)

The channel [SECCFG](#) registers require privileged writes (SP/NSP), and will generate a bus fault on an attempted unprivileged write (SU/NSU). Additionally, the [S](#) bit (MSB of the security level) and the [LOCK](#) bit are writable only by SP, whilst the [P](#) bit (LSB of the security level) is also writable by NSP, if and only if the [S](#) bit is clear. Reads are always allowed: it is always possible to query which channels are assigned to you by reading the channel [SECCFG](#) registers.

Each channel [SECCFG](#) register can be locked manually by writing a one to the [LOCK](#) bit in that register, and will also lock automatically upon a successful write to one of the channel's control registers such as [CH0_CTRL_TRIG](#). This automatic locking avoids any race conditions that may arise from a channel's security level changing after it has already started making transfers, or from leaking secure pointers that have been written to its control registers. Once a channel [SECCFG](#) register has been locked, it becomes read-only. [LOCK](#) bits can be cleared only by a full reset of the DMA block.

Note that [SECCFG](#) registers can be written multiple times before being locked, so the full assignment does not have to be known up front: for example, Secure Arm software can set spare channels to NSP before launching the Non-secure software context, and Non-secure, Privileged software can then set the remaining channels it does not need to NSU before returning to the Non-secure, Normal context.

12.6.6.2. Interrupt Security Assignment

The RP2350 DMA has four system-level interrupt request lines (IRQs), each of which can be asserted on any combination of channel interrupts, as defined by the channel masks in the interrupt enable registers [INTE0](#) through [INTE3](#). Because the timing of interrupts may leak information, and because it is possible to cause software to malfunction by deliberately manipulating its interrupts, access to the channel interrupt flags must be controlled.

The interrupt security configuration registers, [SECCFG_IRQ0](#) through [SECCFG_IRQ3](#), define the security level for each interrupt. This is one of the four security levels laid out in [Section 12.6.6](#). The security level of an IRQ defines:

- Which channels are visible in this IRQ's status registers: channels of a level higher than the IRQ's will read back as zero
- Whether a bus access to this IRQ's control and status registers is permitted: bus accesses below this IRQ's security level will return bus faults and have no effect on the DMA
- Which channels will assert this IRQ: channels of a level higher than this IRQ's level will not cause the interrupt to assert, even if relevant INTE bit is set
- Whether a channel's interrupt can be cleared through this IRQ's INTS register, or set through this channel's INTF register: the interrupt flags of channels of higher security level than the IRQ can not be set or cleared

Note that the [INTR](#) register is shared between all IRQs, so it does not respect any of the IRQ security levels. Instead, it follows the security level of the bus access: reads of [INTR](#) will return the interrupt flags of all channels at or below the security level of the bus access (with higher-level channels reading back as zeroes), and writes to [INTR](#) have write-one-clear behaviour on channels which are at or below the security level of the bus access.

12.6.6.3. Memory Protection Unit

The DMA memory protection unit (MPU) monitors the addresses of all read/write transfers performed by the DMA, and notes the security level of the originating channel. The MPU is configured in advance with a user-defined security address map, which specifies the *minimum* security level required to access up to eight dynamically configured regions. This is one of the four security levels defined in [Section 12.6.6](#).

Transfers which fail to meet the minimum security level for their address are shot down before reaching the system bus, and a bus error is returned to the originating channel. This will be reported as either a read or write bus error in the channel's [CTRL](#) register, depending on whether it was a read or write address that failed the security check.

The intended use for the DMA MPU is to mirror the security definitions of SRAM and XIP memory from the processor SAU or PMP. The number of DMA MPU regions is not sufficient for assigning individual peripherals, so the [ACCESSCTRL](#) bus access registers ([Section 10.5.2](#)) are provided for this purpose.

Each of the eight MPU regions is configured with a base address, [MPU_BAR0](#) through [MPU_BAR7](#) for each region, and a limit address, [MPU_LAR0](#) through [MPU_LAR7](#).

MPU regions have a granularity of 32 bytes, so the base/limit addresses are configured by the 27 most-significant bits of each [BAR/LAR](#) register (bits 31:5). Addresses match MPU regions when the 27 most-significant bits of the address are greater than or equal to the [BAR](#) address bits, and less than or equal to the [LAR](#) address bits. For example, when [MPU_BAR0](#) and [MPU_LAR0](#) both have the value `0x1000000`, MPU region 0 matches on a 32-byte region extending from byte address `0x1000000` to `0x1000001f` (inclusive). Regions can be enabled or disabled using the [LAR.EN](#) bits – if a region is disabled, it matches no addresses.

The minimum security level required to access each region is defined by the [S](#) and [P](#) bits in the LSBs of that region's [LAR](#) register. When an address matches multiple regions, the lowest-numbered region applies. This matches the tie-break rules for the RISC-V PMP, but is different from the Arm SAU tie-break rules, so care must be taken when mirroring SAU mappings with overlapping regions. When none of the MPU regions are matched, the security level is defined by the global [MPU_CTRL.S](#) and [MPU_CTRL.P](#) bits.

The MPU configuration registers ([MPU_CTRL](#), [MPU_BAR0](#) through [MPU_BAR7](#) and [MPU_LAR0](#) through [MPU_LAR7](#)) do not permit unprivileged access. Bus accesses at the SU and NSU security levels will return a bus fault and have no other effect.

The MPU registers are also mostly read-only to NSP accesses, with the sole exception being the region [P](#) bits which are NSP-writable if and only if the corresponding region's [S](#) bit is clear. This delegates to Privileged, Non-secure software the decision of whether Non-secure regions are NSU-accessible.

12.6.7. Bus Error Handling

A bus error is an error condition flagged to one of the DMA's manager ports in response to an attempted read or write transfer, indicating the transfer was rejected for one of the following reasons:

- The DMA MPU forbids access to this address at the originating channel's security level ([Section 12.6.6.3](#)).
- The bus fabric failed to decode the address; the address did not match any known memory location (for example SIO is not visible from the DMA bus ports as it is tightly coupled to the processors).
- [ACCESSCTRL](#) forbids access to the addressed region at the originating channel's privilege level ([Section 10.5.2](#)).
- [ACCESSCTRL](#) forbids DMA access to the addressed region, irrespective of privilege.
- The APB bridge returned a timeout fault for a transfer exceeding 65535 cycles (e.g. accessed ADC whilst `clk_adc` was stopped).
- The downstream bus port returned an error response for any other device-specific reason, e.g. attempting to access configuration registers for a DMA channel with higher security level ([Section 12.6.6.1](#)).

12.6.7.1. Response to Bus Errors

Upon encountering a bus error, the DMA halts the offending channel and reports the error through the channel's [CH0_CTRL_TRIG.READ_ERROR](#) and [WRITE_ERROR](#) flags. The channel stops scheduling bus accesses.

Bus errors are exceptional events which usually indicate misconfiguration of the DMA or some other system hardware. Therefore the DMA refuses to restart the offending channel until its error status is cleared by writing 1 to the relevant error flag. Other channels are not affected, and continue their transfer sequences uninterrupted.

A channel which encounters a bus error does not [CHAIN_TO](#) other channels.

Bus errors always cause the channel's interrupt request to be asserted. Whether or not this causes a system-level IRQ depends on the channel masks configured in interrupt enable registers [INTE0](#) through [INTE3](#).

12.6.7.2. Recovery after Bus Errors

If an error is reported through `READ_ERR/WRITE_ERR` then, before restarting the channel, software must:

1. Poll for a low `BUSY` status to ensure that all in-flight transfers for this channel have been flushed from the DMA's bus pipeline.
2. Clear the error flags by writing 1 to each flag.

Generally the `BUSY` flag will already be low long before the processor enters its interrupt handler and checks the error status, but it is possible for these events to overlap when the DMA is accessing a slow device such as XIP with a high `SCK` divisor and processors are executing from SRAM.

`READ_ADDR` and `WRITE_ADDR` contain the approximate address where the bus error was encountered. This may be useful for the programmer to understand why the bus error occurred, and fix the software to avoid it in future.

Since the DMA performs reads and writes in parallel, it is possible for a channel to encounter both a read and write error simultaneously, and in this case the DMA sets both `READ_ERR` and `WRITE_ERR`. You must clear both.

12.6.7.3. Halt Timing

The DMA halts the channel as soon as possible following a bus error. This suppresses future reads and writes. Because the request to access the bus is masked, the bus access has no side effects on the system. The timing relationships are not straightforward due to the DMA's pipelining and buffering. The DMA provides the following ordering guarantees between transfers originating from one channel:

- Read error → read suppression:
 - Any reads scheduled to occur after a faulting read *will* be suppressed, but *may* still increment `READ_ADDR` up to two times total
- Write error → write suppression:
 - Any writes scheduled to occur after a faulting write *will* be suppressed, but *may* still increment `WRITE_ADDR` up to four times total
- Read error → write suppression:
 - Any write paired with a faulting read *will* be suppressed, but *will* increment `WRITE_ADDR`
 - Any write following the first write paired with a faulting read *will* be suppressed, but *may* increment `WRITE_ADDR` up to three times total
 - Up to three writes immediately preceding the first write paired with a faulting read *may* be suppressed, but *will* increment `WRITE_ADDR`
- Write error → read suppression:
 - Reads paired with writes before the first faulting write *will not* be suppressed, and *will* increment `READ_ADDR`.
 - Up to two read transfers paired with writes after the first faulting write *may* be suppressed, and *may* increment `READ_ADDR`

"Paired with" in the above paragraph refers to *the write access which writes data originating from a particular read transfer*, or vice versa. The DMA always schedules read and write accesses in matched pairs.

Slight variability in halt behaviour is due to the buffering of in-flight transfers, and the parallel operation of the read and write bus ports. The values of `READ_ADDR/WRITE_ADDR` following a bus error may be slightly beyond the address that experienced the first error, but the difference is bounded, and usually this is still sufficient to diagnose the reason for the fault. Additionally, `READ_ADDR` and `WRITE_ADDR` are guaranteed to over-increment by the same amount, since reads and writes are always scheduled in pairs.

Note in addition to the increments mentioned above, `READ_ADDR/WRITE_ADDR` always point to the *next* address to be written, so always point slightly past the faulting address if address increment is enabled.

12.6.8. Additional Features

12.6.8.1. Pacing Timers

These allow transfer of data roughly once every n `clk_sys` clocks instead of using external peripheral DREQ to trigger transfers. A fractional (X/Y) divider is used, and will generate a maximum of 1 request per `clk_sys` cycle.

There are 4 timers available in RP2350. Each DMA channel is able to select any of these in `CTRL.TREQ_SEL`. There is one register used to configure the pacing coefficients for each timer, `TIMER0` through `TIMER3`.

Each timer's security level is defined by a register field in `SECCFG_MISC`. This defines the minimum bus security level required to configure that timer (lower levels will get a bus fault), and the minimum channel security level required to observe that timer's TREQ.

12.6.8.2. CRC Calculation

The DMA can watch data from a given channel passing through the data FIFO, and calculate checksums based on this data. This is a purely passive affair: the data is not altered by this hardware, only observed.

The feature is controlled via the `SNIFF_CTRL` and `SNIFF_DATA` registers, and can be enabled/disabled per DMA transfer via the `CTRL.SNIFF_EN` field.

As this hardware cannot place back-pressure on the FIFO, it must keep up with the DMA's maximum transfer rate of 32 bits per clock.

The supported checksums are:

- CRC-32, MSB-first and LSB-first
- CRC-16-CCITT, MSB-first and LSB-first
- Simple summation (add to 32-bit accumulator)
- Even parity

The result register is both readable and writable, so that the initial seed value can be set.

Bit/byte manipulations are available on the result which may aid specific use cases:

- Bit inversion
- Bit reversal
- Byte swap

These manipulations do not affect the CRC calculation, just how the data is presented in the result register.

The sniffer's security level is configured by the `SECCFG_MISC.SNIFF_S` and `SECCFG_MISC.SNIFF_P` bits. This determines the minimum bus security level required to access the sniffer's control registers, as well as the maximum channel security level which the sniffer can observe.

12.6.8.3. Channel Abort

It is possible for a channel to get into an irrecoverable state. If commanded to transfer more data than a peripheral will ever request, the channel will never complete. Clearing the `CTRL.EN` bit pauses the channel, but does not solve the problem. This should not occur under normal circumstances, but it is important that there is a mechanism to recover without simply hard-resetting the entire DMA block.

In such a situation, use the `CHAN_ABORT` register to force the channel to complete early. There is one bit for each channel. Writing a 1 to the corresponding bit terminates the channel. This clears the transfer counter and forces the channel into an inactive state.

At the time an abort is triggered, a channel may have bus transfers currently in flight between the read and write manager. These transfers cannot be revoked. The `CTRL.BUSY` flag stays high until these transfers complete, and the channel reaches a safe state. This generally takes only a few cycles. The channel must not be restarted until its `CTRL.BUSY` flag deasserts. Starting a new sequence of transfers whilst transfers from an old sequence are still in flight will cause unpredictable behaviour.

The sequence to abort one or more channels in an unknown state (also accounting for the behaviour described in RP2350-E5) is:

1. Clear the `EN` bit and disable `CHAIN_TO` for all channels to be aborted.
2. Write the `CHAN_ABORT` register with a bitmap of those same channels.
3. Poll the `ABORT` register until all bits set by the previous write are clear.

When aborting a channel involved in a `CHAIN_TO`, it is recommended to simultaneously abort all other channels involved in the chain.

12.6.8.4. Debug

Debug registers are available for each DMA channel to show the dreq counter `DBG_CTDREQ` and next transfer count `DBG_TCR`. These can also be used to reset a DMA channel if required.

12.6.9. Example Use Cases

12.6.9.1. Using Interrupts to Reconfigure a Channel

When a channel finishes a block of transfers, it becomes available for making more transfers. Software detects that the channel is no longer busy, and reconfigures and restarts the channel. One approach is to poll the `CTRL_BUSY` bit until the channel is done, but this loses one of the key advantages of the DMA, namely that it does *not* have to operate in lockstep with a processor. By setting the correct bit in `INTE0` through `INTE3`, you can instruct the DMA to raise one of its four interrupt request lines when a given channel completes. Rather than repeatedly asking if a channel is done, you are told.

NOTE

Having four system interrupt lines allows different channel completion interrupts to be routed to different cores, or to pre-empt one another on the same core if one channel is more time-critical. It also allows channel interrupts to target different security domains.

When the interrupt is asserted, the processor can be configured to drop whatever it is doing and call a user-specified handler function. The handler can reconfigure and restart the channel. When the handler exits, the processor returns to the interrupted code running in the foreground.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/dma/channel_irq/channel_irq.c Lines 35 - 52

```

35 void dma_handler() {
36     static int pwm_level = 0;
37     static uint32_t wavetable[N_PWM_LEVELS];
38     static bool first_run = true;
39     // Entry number `i` has `i` one bits and `(32 - i)` zero bits.
40     if (first_run) {
41         first_run = false;
42         for (int i = 0; i < N_PWM_LEVELS; ++i)
43             wavetable[i] = ~(~0u << i);
44     }
45 }
```

```

46  // Clear the interrupt request.
47  dma_hw->ints0 = 1u << dma_chan;
48  // Give the channel a new wave table entry to read from, and re-trigger it
49  dma_channel_set_read_addr(dma_chan, &wavetable[pwm_level], true);
50
51  pwm_level = (pwm_level + 1) % N_PWM_LEVELS;
52 }

```

In many cases, most of the configuration can be done the first time the channel starts. This way, only addresses and transfer lengths need reprogramming in the interrupt handler.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/dma/channel_irq/channel_irq.c Lines 54 - 94

```

54 int main() {
55 #ifndef PICO_DEFAULT_LED_PIN
56 #warning dma/channel_irq example requires a board with a regular LED
57 #else
58  // Set up a PIO state machine to serialise our bits
59  uint offset = pio_add_program(pio0, &pio_serialiser_program);
60  pio_serialiser_program_init(pio0, 0, offset, PICO_DEFAULT_LED_PIN, PIO_SERIAL_CLKDIV);
61
62  // Configure a channel to write the same word (32 bits) repeatedly to PIO0
63  // SM0's TX FIFO, paced by the data request signal from that peripheral.
64  dma_chan = dma_claim_unused_channel(true);
65  dma_channel_config c = dma_channel_get_default_config(dma_chan);
66  channel_config_set_transfer_data_size(&c, DMA_SIZE_32);
67  channel_config_set_read_increment(&c, false);
68  channel_config_set_dreq(&c, DREQ_PI00_TX0);
69
70  dma_channel_configure(
71      dma_chan,
72      &c,
73      &pio0_hw->txf[0], // Write address (only need to set this once)
74      NULL,             // Don't provide a read address yet
75      PWM_REPEAT_COUNT, // Write the same value many times, then halt and interrupt
76      false             // Don't start yet
77  );
78
79  // Tell the DMA to raise IRQ line 0 when the channel finishes a block
80  dma_channel_set_irq0_enabled(dma_chan, true);
81
82  // Configure the processor to run dma_handler() when DMA IRQ 0 is asserted
83  irq_set_exclusive_handler(DMA_IRQ_0, dma_handler);
84  irq_set_enabled(DMA_IRQ_0, true);
85
86  // Manually call the handler once, to trigger the first transfer
87  dma_handler();
88
89  // Everything else from this point is interrupt-driven. The processor has
90  // time to sit and think about its early retirement -- maybe open a bakery?
91  while (true)
92      tight_loop_contents();
93 #endif
94 }

```

One disadvantage of this technique is that you don't start to reconfigure the channel until some time after the channel makes its last transfer. If there is heavy interrupt activity on the processor, this may be quite a long time, and quite a large gap in transfers. This makes it difficult to sustain a high data throughput.

This is solved by using two channels, with their **CHAIN_TO** fields crossed over, so that channel A triggers channel B when it completes, and vice versa. At any point in time, one of the channels is transferring data. The other is either already

configured to start the next transfer immediately when the current one finishes, or it is in the process of being reconfigured. When channel A completes, it immediately starts the cued-up transfer on channel B. At the same time, the interrupt is fired, and the handler reconfigures channel A so that it is ready when channel B completes.

12.6.9.2. DMA Control Blocks

Frequently, multiple smaller buffers must be gathered together and sent to the same peripheral. To address this use case, the RP2350 DMA can execute a long and complex sequence of transfers without processor control. One channel repeatedly reconfigures a second channel, and the second channel restarts the first each time it completes block of transfers.

Because the first DMA channel transfers data directly from memory to the second channel's control registers, the format of the control blocks in memory must match those registers. Each time, the last register written to will be one of the trigger registers (Section 12.6.3.1), which will start the second channel on its programmed block of transfers. The register aliases (Section 12.6.3.1) give some flexibility for the block layout, and more importantly allow some registers to be omitted from the blocks, so they occupy less memory and can be loaded more quickly.

This example shows how multiple buffers can be gathered and transferred to the UART, by reprogramming TRANS_COUNT and READ_ADDR_TRIG:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/dma/control_blocks/control_blocks.c

```

1 /**
2  * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
3  *
4  * SPDX-License-Identifier: BSD-3-Clause
5  */
6
7 // Use two DMA channels to make a programmed sequence of data transfers to the
8 // UART (a data gather operation). One channel is responsible for transferring
9 // the actual data, the other repeatedly reprograms that channel.
10
11 #include <stdio.h>
12 #include "pico/stl.h"
13 #include "hardware/dma.h"
14 #include "hardware/structs/uart.h"
15
16 // These buffers will be DMA'd to the UART, one after the other.
17
18 const char word0[] = "Transferring ";
19 const char word1[] = "one ";
20 const char word2[] = "word ";
21 const char word3[] = "at ";
22 const char word4[] = "a ";
23 const char word5[] = "time.\n";
24
25 // Note the order of the fields here: it's important that the length is before
26 // the read address, because the control channel is going to write to the last
27 // two registers in alias 3 on the data channel:
28 //      +0x0      +0x4      +0x8      +0xC (Trigger)
29 // Alias 0: READ_ADDR  WRITE_ADDR  TRANS_COUNT  CTRL
30 // Alias 1: CTRL      READ_ADDR  WRITE_ADDR  TRANS_COUNT
31 // Alias 2: CTRL      TRANS_COUNT  READ_ADDR  WRITE_ADDR
32 // Alias 3: CTRL      WRITE_ADDR  TRANS_COUNT  READ_ADDR
33 //
34 // This will program the transfer count and read address of the data channel,
35 // and trigger it. Once the data channel completes, it will restart the
36 // control channel (via CHAIN_TO) to load the next two words into its control
37 // registers.
38
39 const struct {uint32_t len; const char *data;} control_blocks[] = {

```

```

40     {count_of(word0) - 1, word0}, // Skip null terminator
41     {count_of(word1) - 1, word1},
42     {count_of(word2) - 1, word2},
43     {count_of(word3) - 1, word3},
44     {count_of(word4) - 1, word4},
45     {count_of(word5) - 1, word5},
46     {0, NULL}                      // Null trigger to end chain.
47 };
48
49 int main() {
50 #ifndef uart_default
51 #warning dma/control_blocks example requires a UART
52 #else
53     stdio_init_all();
54     puts("DMA control block example:");
55
56     // ctrl_chan loads control blocks into data_chan, which executes them.
57     int ctrl_chan = dma_claim_unused_channel(true);
58     int data_chan = dma_claim_unused_channel(true);
59
60     // The control channel transfers two words into the data channel's control
61     // registers, then halts. The write address wraps on a two-word
62     // (eight-byte) boundary, so that the control channel writes the same two
63     // registers when it is next triggered.
64
65     dma_channel_config c = dma_channel_get_default_config(ctrl_chan);
66     channel_config_set_transfer_data_size(&c, DMA_SIZE_32);
67     channel_config_set_read_increment(&c, true);
68     channel_config_set_write_increment(&c, true);
69     channel_config_set_ring(&c, true, 3); // 1 << 3 byte boundary on write ptr
70
71     dma_channel_configure(
72         ctrl_chan,
73         &c,
74         &dma_hw->ch[data_chan].a13_transfer_count, // Initial write address
75         &control_blocks[0],                         // Initial read address
76         2,                                         // Halt after each control block
77         false                                      // Don't start yet
78     );
79
80     // The data channel is set up to write to the UART FIFO (paced by the
81     // UART's TX data request signal) and then chain to the control channel
82     // once it completes. The control channel programs a new read address and
83     // data length, and retriggers the data channel.
84
85     c = dma_channel_get_default_config(data_chan);
86     channel_config_set_transfer_data_size(&c, DMA_SIZE_8);
87     channel_config_set_dreq(&c, uart_get_dreq(uart_default, true));
88     // Trigger ctrl_chan when data_chan completes
89     channel_config_set_chain_to(&c, ctrl_chan);
90     // Raise the IRQ flag when 0 is written to a trigger register (end of chain):
91     channel_config_set_irq_quiet(&c, true);
92
93     dma_channel_configure(
94         data_chan,
95         &c,
96         &uart_get_hw(uart_default)->dr,
97         NULL,                                // Initial read address and transfer count are unimportant;
98         0,                                   // the control channel will reprogram them each time.
99         false                                 // Don't start yet.
100    );
101
102    // Everything is ready to go. Tell the control channel to load the first
103    // control block. Everything is automatic from here.

```

```

104     dma_start_channel_mask(1u << ctrl_chan);
105
106     // The data channel will assert its IRQ flag when it gets a null trigger,
107     // indicating the end of the control block list. We're just going to wait
108     // for the IRQ flag instead of setting up an interrupt handler.
109     while (!(dma_hw->intr & 1u << data_chan))
110         tight_loop_contents();
111     dma_hw->ints0 = 1u << data_chan;
112
113     puts("DMA finished.");
114 #endif
115 }

```

12.6.10. List of Registers

The DMA registers start at a base address of `0x50000000` (defined as `DMA_BASE` in SDK).

Table 1100. List of DMA registers

Offset	Name	Info
0x000	<code>CH0_READ_ADDR</code>	DMA Channel 0 Read Address pointer
0x004	<code>CH0_WRITE_ADDR</code>	DMA Channel 0 Write Address pointer
0x008	<code>CH0_TRANS_COUNT</code>	DMA Channel 0 Transfer Count
0x00c	<code>CH0_CTRL_TRIG</code>	DMA Channel 0 Control and Status
0x010	<code>CH0_AL1_CTRL</code>	Alias for channel 0 CTRL register
0x014	<code>CH0_AL1_READ_ADDR</code>	Alias for channel 0 READ_ADDR register
0x018	<code>CH0_AL1_WRITE_ADDR</code>	Alias for channel 0 WRITE_ADDR register
0x01c	<code>CH0_AL1_TRANS_COUNT_TRIG</code>	Alias for channel 0 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x020	<code>CH0_AL2_CTRL</code>	Alias for channel 0 CTRL register
0x024	<code>CH0_AL2_TRANS_COUNT</code>	Alias for channel 0 TRANS_COUNT register
0x028	<code>CH0_AL2_READ_ADDR</code>	Alias for channel 0 READ_ADDR register
0x02c	<code>CH0_AL2_WRITE_ADDR_TRIG</code>	Alias for channel 0 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x030	<code>CH0_AL3_CTRL</code>	Alias for channel 0 CTRL register
0x034	<code>CH0_AL3_WRITE_ADDR</code>	Alias for channel 0 WRITE_ADDR register
0x038	<code>CH0_AL3_TRANS_COUNT</code>	Alias for channel 0 TRANS_COUNT register
0x03c	<code>CH0_AL3_READ_ADDR_TRIG</code>	Alias for channel 0 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x040	<code>CH1_READ_ADDR</code>	DMA Channel 1 Read Address pointer
0x044	<code>CH1_WRITE_ADDR</code>	DMA Channel 1 Write Address pointer
0x048	<code>CH1_TRANS_COUNT</code>	DMA Channel 1 Transfer Count
0x04c	<code>CH1_CTRL_TRIG</code>	DMA Channel 1 Control and Status

Offset	Name	Info
0x050	CH1_AL1_CTRL	Alias for channel 1 CTRL register
0x054	CH1_AL1_READ_ADDR	Alias for channel 1 READ_ADDR register
0x058	CH1_AL1_WRITE_ADDR	Alias for channel 1 WRITE_ADDR register
0x05c	CH1_AL1_TRANS_COUNT_TRIG	Alias for channel 1 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x060	CH1_AL2_CTRL	Alias for channel 1 CTRL register
0x064	CH1_AL2_TRANS_COUNT	Alias for channel 1 TRANS_COUNT register
0x068	CH1_AL2_READ_ADDR	Alias for channel 1 READ_ADDR register
0x06c	CH1_AL2_WRITE_ADDR_TRIG	Alias for channel 1 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x070	CH1_AL3_CTRL	Alias for channel 1 CTRL register
0x074	CH1_AL3_WRITE_ADDR	Alias for channel 1 WRITE_ADDR register
0x078	CH1_AL3_TRANS_COUNT	Alias for channel 1 TRANS_COUNT register
0x07c	CH1_AL3_READ_ADDR_TRIG	Alias for channel 1 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x080	CH2_READ_ADDR	DMA Channel 2 Read Address pointer
0x084	CH2_WRITE_ADDR	DMA Channel 2 Write Address pointer
0x088	CH2_TRANS_COUNT	DMA Channel 2 Transfer Count
0x08c	CH2_CTRL_TRIG	DMA Channel 2 Control and Status
0x090	CH2_AL1_CTRL	Alias for channel 2 CTRL register
0x094	CH2_AL1_READ_ADDR	Alias for channel 2 READ_ADDR register
0x098	CH2_AL1_WRITE_ADDR	Alias for channel 2 WRITE_ADDR register
0x09c	CH2_AL1_TRANS_COUNT_TRIG	Alias for channel 2 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x0a0	CH2_AL2_CTRL	Alias for channel 2 CTRL register
0x0a4	CH2_AL2_TRANS_COUNT	Alias for channel 2 TRANS_COUNT register
0x0a8	CH2_AL2_READ_ADDR	Alias for channel 2 READ_ADDR register
0x0ac	CH2_AL2_WRITE_ADDR_TRIG	Alias for channel 2 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x0b0	CH2_AL3_CTRL	Alias for channel 2 CTRL register
0x0b4	CH2_AL3_WRITE_ADDR	Alias for channel 2 WRITE_ADDR register
0x0b8	CH2_AL3_TRANS_COUNT	Alias for channel 2 TRANS_COUNT register
0x0bc	CH2_AL3_READ_ADDR_TRIG	Alias for channel 2 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.

Offset	Name	Info
0x0c0	CH3_READ_ADDR	DMA Channel 3 Read Address pointer
0x0c4	CH3_WRITE_ADDR	DMA Channel 3 Write Address pointer
0x0c8	CH3_TRANS_COUNT	DMA Channel 3 Transfer Count
0x0cc	CH3_CTRL_TRIG	DMA Channel 3 Control and Status
0x0d0	CH3_AL1_CTRL	Alias for channel 3 CTRL register
0x0d4	CH3_AL1_READ_ADDR	Alias for channel 3 READ_ADDR register
0x0d8	CH3_AL1_WRITE_ADDR	Alias for channel 3 WRITE_ADDR register
0x0dc	CH3_AL1_TRANS_COUNT_TRIG	Alias for channel 3 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x0e0	CH3_AL2_CTRL	Alias for channel 3 CTRL register
0x0e4	CH3_AL2_TRANS_COUNT	Alias for channel 3 TRANS_COUNT register
0x0e8	CH3_AL2_READ_ADDR	Alias for channel 3 READ_ADDR register
0x0ec	CH3_AL2_WRITE_ADDR_TRIG	Alias for channel 3 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x0f0	CH3_AL3_CTRL	Alias for channel 3 CTRL register
0x0f4	CH3_AL3_WRITE_ADDR	Alias for channel 3 WRITE_ADDR register
0x0f8	CH3_AL3_TRANS_COUNT	Alias for channel 3 TRANS_COUNT register
0x0fc	CH3_AL3_READ_ADDR_TRIG	Alias for channel 3 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x100	CH4_READ_ADDR	DMA Channel 4 Read Address pointer
0x104	CH4_WRITE_ADDR	DMA Channel 4 Write Address pointer
0x108	CH4_TRANS_COUNT	DMA Channel 4 Transfer Count
0x10c	CH4_CTRL_TRIG	DMA Channel 4 Control and Status
0x110	CH4_AL1_CTRL	Alias for channel 4 CTRL register
0x114	CH4_AL1_READ_ADDR	Alias for channel 4 READ_ADDR register
0x118	CH4_AL1_WRITE_ADDR	Alias for channel 4 WRITE_ADDR register
0x11c	CH4_AL1_TRANS_COUNT_TRIG	Alias for channel 4 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x120	CH4_AL2_CTRL	Alias for channel 4 CTRL register
0x124	CH4_AL2_TRANS_COUNT	Alias for channel 4 TRANS_COUNT register
0x128	CH4_AL2_READ_ADDR	Alias for channel 4 READ_ADDR register
0x12c	CH4_AL2_WRITE_ADDR_TRIG	Alias for channel 4 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x130	CH4_AL3_CTRL	Alias for channel 4 CTRL register

Offset	Name	Info
0x134	CH4_AL3_WRITE_ADDR	Alias for channel 4 WRITE_ADDR register
0x138	CH4_AL3_TRANS_COUNT	Alias for channel 4 TRANS_COUNT register
0x13c	CH4_AL3_READ_ADDR_TRIG	Alias for channel 4 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x140	CH5_READ_ADDR	DMA Channel 5 Read Address pointer
0x144	CH5_WRITE_ADDR	DMA Channel 5 Write Address pointer
0x148	CH5_TRANS_COUNT	DMA Channel 5 Transfer Count
0x14c	CH5_CTRL_TRIG	DMA Channel 5 Control and Status
0x150	CH5_AL1_CTRL	Alias for channel 5 CTRL register
0x154	CH5_AL1_READ_ADDR	Alias for channel 5 READ_ADDR register
0x158	CH5_AL1_WRITE_ADDR	Alias for channel 5 WRITE_ADDR register
0x15c	CH5_AL1_TRANS_COUNT_TRIG	Alias for channel 5 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x160	CH5_AL2_CTRL	Alias for channel 5 CTRL register
0x164	CH5_AL2_TRANS_COUNT	Alias for channel 5 TRANS_COUNT register
0x168	CH5_AL2_READ_ADDR	Alias for channel 5 READ_ADDR register
0x16c	CH5_AL2_WRITE_ADDR_TRIG	Alias for channel 5 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x170	CH5_AL3_CTRL	Alias for channel 5 CTRL register
0x174	CH5_AL3_WRITE_ADDR	Alias for channel 5 WRITE_ADDR register
0x178	CH5_AL3_TRANS_COUNT	Alias for channel 5 TRANS_COUNT register
0x17c	CH5_AL3_READ_ADDR_TRIG	Alias for channel 5 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x180	CH6_READ_ADDR	DMA Channel 6 Read Address pointer
0x184	CH6_WRITE_ADDR	DMA Channel 6 Write Address pointer
0x188	CH6_TRANS_COUNT	DMA Channel 6 Transfer Count
0x18c	CH6_CTRL_TRIG	DMA Channel 6 Control and Status
0x190	CH6_AL1_CTRL	Alias for channel 6 CTRL register
0x194	CH6_AL1_READ_ADDR	Alias for channel 6 READ_ADDR register
0x198	CH6_AL1_WRITE_ADDR	Alias for channel 6 WRITE_ADDR register
0x19c	CH6_AL1_TRANS_COUNT_TRIG	Alias for channel 6 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x1a0	CH6_AL2_CTRL	Alias for channel 6 CTRL register
0x1a4	CH6_AL2_TRANS_COUNT	Alias for channel 6 TRANS_COUNT register

Offset	Name	Info
0x1a8	CH6_AL2_READ_ADDR	Alias for channel 6 READ_ADDR register
0x1ac	CH6_AL2_WRITE_ADDR_TRIG	Alias for channel 6 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x1b0	CH6_AL3_CTRL	Alias for channel 6 CTRL register
0x1b4	CH6_AL3_WRITE_ADDR	Alias for channel 6 WRITE_ADDR register
0x1b8	CH6_AL3_TRANS_COUNT	Alias for channel 6 TRANS_COUNT register
0x1bc	CH6_AL3_READ_ADDR_TRIG	Alias for channel 6 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x1c0	CH7_READ_ADDR	DMA Channel 7 Read Address pointer
0x1c4	CH7_WRITE_ADDR	DMA Channel 7 Write Address pointer
0x1c8	CH7_TRANS_COUNT	DMA Channel 7 Transfer Count
0x1cc	CH7_CTRL_TRIG	DMA Channel 7 Control and Status
0x1d0	CH7_AL1_CTRL	Alias for channel 7 CTRL register
0x1d4	CH7_AL1_READ_ADDR	Alias for channel 7 READ_ADDR register
0x1d8	CH7_AL1_WRITE_ADDR	Alias for channel 7 WRITE_ADDR register
0x1dc	CH7_AL1_TRANS_COUNT_TRIG	Alias for channel 7 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x1e0	CH7_AL2_CTRL	Alias for channel 7 CTRL register
0x1e4	CH7_AL2_TRANS_COUNT	Alias for channel 7 TRANS_COUNT register
0x1e8	CH7_AL2_READ_ADDR	Alias for channel 7 READ_ADDR register
0x1ec	CH7_AL2_WRITE_ADDR_TRIG	Alias for channel 7 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x1f0	CH7_AL3_CTRL	Alias for channel 7 CTRL register
0x1f4	CH7_AL3_WRITE_ADDR	Alias for channel 7 WRITE_ADDR register
0x1f8	CH7_AL3_TRANS_COUNT	Alias for channel 7 TRANS_COUNT register
0x1fc	CH7_AL3_READ_ADDR_TRIG	Alias for channel 7 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x200	CH8_READ_ADDR	DMA Channel 8 Read Address pointer
0x204	CH8_WRITE_ADDR	DMA Channel 8 Write Address pointer
0x208	CH8_TRANS_COUNT	DMA Channel 8 Transfer Count
0x20c	CH8_CTRL_TRIG	DMA Channel 8 Control and Status
0x210	CH8_AL1_CTRL	Alias for channel 8 CTRL register
0x214	CH8_AL1_READ_ADDR	Alias for channel 8 READ_ADDR register
0x218	CH8_AL1_WRITE_ADDR	Alias for channel 8 WRITE_ADDR register

Offset	Name	Info
0x21c	CH8_AL1_TRANS_COUNT_TRIG	Alias for channel 8 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x220	CH8_AL2_CTRL	Alias for channel 8 CTRL register
0x224	CH8_AL2_TRANS_COUNT	Alias for channel 8 TRANS_COUNT register
0x228	CH8_AL2_READ_ADDR	Alias for channel 8 READ_ADDR register
0x22c	CH8_AL2_WRITE_ADDR_TRIG	Alias for channel 8 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x230	CH8_AL3_CTRL	Alias for channel 8 CTRL register
0x234	CH8_AL3_WRITE_ADDR	Alias for channel 8 WRITE_ADDR register
0x238	CH8_AL3_TRANS_COUNT	Alias for channel 8 TRANS_COUNT register
0x23c	CH8_AL3_READ_ADDR_TRIG	Alias for channel 8 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x240	CH9_READ_ADDR	DMA Channel 9 Read Address pointer
0x244	CH9_WRITE_ADDR	DMA Channel 9 Write Address pointer
0x248	CH9_TRANS_COUNT	DMA Channel 9 Transfer Count
0x24c	CH9_CTRL_TRIG	DMA Channel 9 Control and Status
0x250	CH9_AL1_CTRL	Alias for channel 9 CTRL register
0x254	CH9_AL1_READ_ADDR	Alias for channel 9 READ_ADDR register
0x258	CH9_AL1_WRITE_ADDR	Alias for channel 9 WRITE_ADDR register
0x25c	CH9_AL1_TRANS_COUNT_TRIG	Alias for channel 9 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x260	CH9_AL2_CTRL	Alias for channel 9 CTRL register
0x264	CH9_AL2_TRANS_COUNT	Alias for channel 9 TRANS_COUNT register
0x268	CH9_AL2_READ_ADDR	Alias for channel 9 READ_ADDR register
0x26c	CH9_AL2_WRITE_ADDR_TRIG	Alias for channel 9 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x270	CH9_AL3_CTRL	Alias for channel 9 CTRL register
0x274	CH9_AL3_WRITE_ADDR	Alias for channel 9 WRITE_ADDR register
0x278	CH9_AL3_TRANS_COUNT	Alias for channel 9 TRANS_COUNT register
0x27c	CH9_AL3_READ_ADDR_TRIG	Alias for channel 9 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x280	CH10_READ_ADDR	DMA Channel 10 Read Address pointer
0x284	CH10_WRITE_ADDR	DMA Channel 10 Write Address pointer
0x288	CH10_TRANS_COUNT	DMA Channel 10 Transfer Count

Offset	Name	Info
0x28c	CH10_CTRL_TRIG	DMA Channel 10 Control and Status
0x290	CH10_AL1_CTRL	Alias for channel 10 CTRL register
0x294	CH10_AL1_READ_ADDR	Alias for channel 10 READ_ADDR register
0x298	CH10_AL1_WRITE_ADDR	Alias for channel 10 WRITE_ADDR register
0x29c	CH10_AL1_TRANS_COUNT_TRIG	Alias for channel 10 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x2a0	CH10_AL2_CTRL	Alias for channel 10 CTRL register
0x2a4	CH10_AL2_TRANS_COUNT	Alias for channel 10 TRANS_COUNT register
0x2a8	CH10_AL2_READ_ADDR	Alias for channel 10 READ_ADDR register
0x2ac	CH10_AL2_WRITE_ADDR_TRIG	Alias for channel 10 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x2b0	CH10_AL3_CTRL	Alias for channel 10 CTRL register
0x2b4	CH10_AL3_WRITE_ADDR	Alias for channel 10 WRITE_ADDR register
0x2b8	CH10_AL3_TRANS_COUNT	Alias for channel 10 TRANS_COUNT register
0x2bc	CH10_AL3_READ_ADDR_TRIG	Alias for channel 10 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x2c0	CH11_READ_ADDR	DMA Channel 11 Read Address pointer
0x2c4	CH11_WRITE_ADDR	DMA Channel 11 Write Address pointer
0x2c8	CH11_TRANS_COUNT	DMA Channel 11 Transfer Count
0x2cc	CH11_CTRL_TRIG	DMA Channel 11 Control and Status
0x2d0	CH11_AL1_CTRL	Alias for channel 11 CTRL register
0x2d4	CH11_AL1_READ_ADDR	Alias for channel 11 READ_ADDR register
0x2d8	CH11_AL1_WRITE_ADDR	Alias for channel 11 WRITE_ADDR register
0x2dc	CH11_AL1_TRANS_COUNT_TRIG	Alias for channel 11 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x2e0	CH11_AL2_CTRL	Alias for channel 11 CTRL register
0x2e4	CH11_AL2_TRANS_COUNT	Alias for channel 11 TRANS_COUNT register
0x2e8	CH11_AL2_READ_ADDR	Alias for channel 11 READ_ADDR register
0x2ec	CH11_AL2_WRITE_ADDR_TRIG	Alias for channel 11 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x2f0	CH11_AL3_CTRL	Alias for channel 11 CTRL register
0x2f4	CH11_AL3_WRITE_ADDR	Alias for channel 11 WRITE_ADDR register
0x2f8	CH11_AL3_TRANS_COUNT	Alias for channel 11 TRANS_COUNT register

Offset	Name	Info
0x2fc	CH11_AL3_READ_ADDR_TRIG	Alias for channel 11 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x300	CH12_READ_ADDR	DMA Channel 12 Read Address pointer
0x304	CH12_WRITE_ADDR	DMA Channel 12 Write Address pointer
0x308	CH12_TRANS_COUNT	DMA Channel 12 Transfer Count
0x30c	CH12_CTRL_TRIG	DMA Channel 12 Control and Status
0x310	CH12_AL1_CTRL	Alias for channel 12 CTRL register
0x314	CH12_AL1_READ_ADDR	Alias for channel 12 READ_ADDR register
0x318	CH12_AL1_WRITE_ADDR	Alias for channel 12 WRITE_ADDR register
0x31c	CH12_AL1_TRANS_COUNT_TRIG	Alias for channel 12 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x320	CH12_AL2_CTRL	Alias for channel 12 CTRL register
0x324	CH12_AL2_TRANS_COUNT	Alias for channel 12 TRANS_COUNT register
0x328	CH12_AL2_READ_ADDR	Alias for channel 12 READ_ADDR register
0x32c	CH12_AL2_WRITE_ADDR_TRIG	Alias for channel 12 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x330	CH12_AL3_CTRL	Alias for channel 12 CTRL register
0x334	CH12_AL3_WRITE_ADDR	Alias for channel 12 WRITE_ADDR register
0x338	CH12_AL3_TRANS_COUNT	Alias for channel 12 TRANS_COUNT register
0x33c	CH12_AL3_READ_ADDR_TRIG	Alias for channel 12 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x340	CH13_READ_ADDR	DMA Channel 13 Read Address pointer
0x344	CH13_WRITE_ADDR	DMA Channel 13 Write Address pointer
0x348	CH13_TRANS_COUNT	DMA Channel 13 Transfer Count
0x34c	CH13_CTRL_TRIG	DMA Channel 13 Control and Status
0x350	CH13_AL1_CTRL	Alias for channel 13 CTRL register
0x354	CH13_AL1_READ_ADDR	Alias for channel 13 READ_ADDR register
0x358	CH13_AL1_WRITE_ADDR	Alias for channel 13 WRITE_ADDR register
0x35c	CH13_AL1_TRANS_COUNT_TRIG	Alias for channel 13 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x360	CH13_AL2_CTRL	Alias for channel 13 CTRL register
0x364	CH13_AL2_TRANS_COUNT	Alias for channel 13 TRANS_COUNT register
0x368	CH13_AL2_READ_ADDR	Alias for channel 13 READ_ADDR register

Offset	Name	Info
0x36c	CH13_AL2_WRITE_ADDR_TRIG	Alias for channel 13 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x370	CH13_AL3_CTRL	Alias for channel 13 CTRL register
0x374	CH13_AL3_WRITE_ADDR	Alias for channel 13 WRITE_ADDR register
0x378	CH13_AL3_TRANS_COUNT	Alias for channel 13 TRANS_COUNT register
0x37c	CH13_AL3_READ_ADDR_TRIG	Alias for channel 13 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x380	CH14_READ_ADDR	DMA Channel 14 Read Address pointer
0x384	CH14_WRITE_ADDR	DMA Channel 14 Write Address pointer
0x388	CH14_TRANS_COUNT	DMA Channel 14 Transfer Count
0x38c	CH14_CTRL_TRIG	DMA Channel 14 Control and Status
0x390	CH14_AL1_CTRL	Alias for channel 14 CTRL register
0x394	CH14_AL1_READ_ADDR	Alias for channel 14 READ_ADDR register
0x398	CH14_AL1_WRITE_ADDR	Alias for channel 14 WRITE_ADDR register
0x39c	CH14_AL1_TRANS_COUNT_TRIG	Alias for channel 14 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x3a0	CH14_AL2_CTRL	Alias for channel 14 CTRL register
0x3a4	CH14_AL2_TRANS_COUNT	Alias for channel 14 TRANS_COUNT register
0x3a8	CH14_AL2_READ_ADDR	Alias for channel 14 READ_ADDR register
0x3ac	CH14_AL2_WRITE_ADDR_TRIG	Alias for channel 14 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x3b0	CH14_AL3_CTRL	Alias for channel 14 CTRL register
0x3b4	CH14_AL3_WRITE_ADDR	Alias for channel 14 WRITE_ADDR register
0x3b8	CH14_AL3_TRANS_COUNT	Alias for channel 14 TRANS_COUNT register
0x3bc	CH14_AL3_READ_ADDR_TRIG	Alias for channel 14 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x3c0	CH15_READ_ADDR	DMA Channel 15 Read Address pointer
0x3c4	CH15_WRITE_ADDR	DMA Channel 15 Write Address pointer
0x3c8	CH15_TRANS_COUNT	DMA Channel 15 Transfer Count
0x3cc	CH15_CTRL_TRIG	DMA Channel 15 Control and Status
0x3d0	CH15_AL1_CTRL	Alias for channel 15 CTRL register
0x3d4	CH15_AL1_READ_ADDR	Alias for channel 15 READ_ADDR register
0x3d8	CH15_AL1_WRITE_ADDR	Alias for channel 15 WRITE_ADDR register

Offset	Name	Info
0x3dc	CH15_AL1_TRANS_COUNT_TRIG	Alias for channel 15 TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x3e0	CH15_AL2_CTRL	Alias for channel 15 CTRL register
0x3e4	CH15_AL2_TRANS_COUNT	Alias for channel 15 TRANS_COUNT register
0x3e8	CH15_AL2_READ_ADDR	Alias for channel 15 READ_ADDR register
0x3ec	CH15_AL2_WRITE_ADDR_TRIG	Alias for channel 15 WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x3f0	CH15_AL3_CTRL	Alias for channel 15 CTRL register
0x3f4	CH15_AL3_WRITE_ADDR	Alias for channel 15 WRITE_ADDR register
0x3f8	CH15_AL3_TRANS_COUNT	Alias for channel 15 TRANS_COUNT register
0x3fc	CH15_AL3_READ_ADDR_TRIG	Alias for channel 15 READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.
0x400	INTR	Interrupt Status (raw)
0x404	INTE0	Interrupt Enables for IRQ 0
0x408	INTF0	Force Interrupts
0x40c	INTS0	Interrupt Status for IRQ 0
0x414	INTE1	Interrupt Enables for IRQ 1
0x418	INTF1	Force Interrupts
0x41c	INTS1	Interrupt Status for IRQ 1
0x424	INTE2	Interrupt Enables for IRQ 2
0x428	INTF2	Force Interrupts
0x42c	INTS2	Interrupt Status for IRQ 2
0x434	INTE3	Interrupt Enables for IRQ 3
0x438	INTF3	Force Interrupts
0x43c	INTS3	Interrupt Status for IRQ 3
0x440	TIMER0	Pacing timer (generate periodic TREQs)
0x444	TIMER1	Pacing timer (generate periodic TREQs)
0x448	TIMER2	Pacing timer (generate periodic TREQs)
0x44c	TIMER3	Pacing timer (generate periodic TREQs)
0x450	MULTI_CHAN_TRIGGER	Trigger one or more channels simultaneously
0x454	SNIFF_CTRL	Sniffer Control
0x458	SNIFF_DATA	Data accumulator for sniff hardware
0x460	FIFO_LEVELS	Debug RAF, WAF, TDF levels
0x464	CHAN_ABORT	Abort an in-progress transfer sequence on one or more channels

Offset	Name	Info
0x468	N_CHANNELS	The number of channels this DMA instance is equipped with. This DMA supports up to 16 hardware channels, but can be configured with as few as one, to minimise silicon area.
0x480	SECCFG_CH0	Security level configuration for channel 0.
0x484	SECCFG_CH1	Security level configuration for channel 1.
0x488	SECCFG_CH2	Security level configuration for channel 2.
0x48c	SECCFG_CH3	Security level configuration for channel 3.
0x490	SECCFG_CH4	Security level configuration for channel 4.
0x494	SECCFG_CH5	Security level configuration for channel 5.
0x498	SECCFG_CH6	Security level configuration for channel 6.
0x49c	SECCFG_CH7	Security level configuration for channel 7.
0x4a0	SECCFG_CH8	Security level configuration for channel 8.
0x4a4	SECCFG_CH9	Security level configuration for channel 9.
0x4a8	SECCFG_CH10	Security level configuration for channel 10.
0x4ac	SECCFG_CH11	Security level configuration for channel 11.
0x4b0	SECCFG_CH12	Security level configuration for channel 12.
0x4b4	SECCFG_CH13	Security level configuration for channel 13.
0x4b8	SECCFG_CH14	Security level configuration for channel 14.
0x4bc	SECCFG_CH15	Security level configuration for channel 15.
0x4c0	SECCFG_IRQ0	Security configuration for IRQ 0. Control whether the IRQ permits configuration by Non-secure/Unprivileged contexts, and whether it can observe Secure/Privileged channel interrupt flags.
0x4c4	SECCFG_IRQ1	Security configuration for IRQ 1. Control whether the IRQ permits configuration by Non-secure/Unprivileged contexts, and whether it can observe Secure/Privileged channel interrupt flags.
0x4c8	SECCFG_IRQ2	Security configuration for IRQ 2. Control whether the IRQ permits configuration by Non-secure/Unprivileged contexts, and whether it can observe Secure/Privileged channel interrupt flags.
0x4cc	SECCFG_IRQ3	Security configuration for IRQ 3. Control whether the IRQ permits configuration by Non-secure/Unprivileged contexts, and whether it can observe Secure/Privileged channel interrupt flags.
0x4d0	SECCFG_MISC	Miscellaneous security configuration
0x500	MPU_CTRL	Control register for DMA MPU. Accessible only from a Privileged context.
0x504	MPU_BAR0	Base address register for MPU region 0. Writable only from a Secure, Privileged context.
0x508	MPU_LAR0	Limit address register for MPU region 0. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x50c	MPU_BAR1	Base address register for MPU region 1. Writable only from a Secure, Privileged context.

Offset	Name	Info
0x510	MPU_LAR1	Limit address register for MPU region 1. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x514	MPU_BAR2	Base address register for MPU region 2. Writable only from a Secure, Privileged context.
0x518	MPU_LAR2	Limit address register for MPU region 2. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x51c	MPU_BAR3	Base address register for MPU region 3. Writable only from a Secure, Privileged context.
0x520	MPU_LAR3	Limit address register for MPU region 3. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x524	MPU_BAR4	Base address register for MPU region 4. Writable only from a Secure, Privileged context.
0x528	MPU_LAR4	Limit address register for MPU region 4. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x52c	MPU_BAR5	Base address register for MPU region 5. Writable only from a Secure, Privileged context.
0x530	MPU_LAR5	Limit address register for MPU region 5. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x534	MPU_BAR6	Base address register for MPU region 6. Writable only from a Secure, Privileged context.
0x538	MPU_LAR6	Limit address register for MPU region 6. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x53c	MPU_BAR7	Base address register for MPU region 7. Writable only from a Secure, Privileged context.
0x540	MPU_LAR7	Limit address register for MPU region 7. Writable only from a Secure, Privileged context, with the exception of the P bit.
0x800	CH0_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x804	CH0_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0x840	CH1_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x844	CH1_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0x880	CH2_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x884	CH2_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer

Offset	Name	Info
0x8c0	CH3_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x8c4	CH3_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0x900	CH4_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x904	CH4_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0x940	CH5_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x944	CH5_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0x980	CH6_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x984	CH6_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0x9c0	CH7_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0x9c4	CH7_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xa00	CH8_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0xa04	CH8_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xa40	CH9_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0xa44	CH9_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xa80	CH10_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.

Offset	Name	Info
0xa84	CH10_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xac0	CH11_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0xac4	CH11_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xb00	CH12_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0xb04	CH12_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xb40	CH13_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0xb44	CH13_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xb80	CH14_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0xb84	CH14_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer
0xbc0	CH15_DBG_CTDREQ	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.
0xbc4	CH15_DBG_TCR	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer

DMA: CH0_READ_ADDR, CH1_READ_ADDR, ..., CH14_READ_ADDR, CH15_READ_ADDR Registers

Offsets: 0x000, 0x040, ..., 0x380, 0x3c0

Description

DMA Channel N Read Address pointer

Table 1101.
CH0_READ_ADDR,
CH1_READ_ADDR, ...,
CH14_READ_ADDR,
CH15_READ_ADDR
Registers

Bits	Description	Type	Reset
31:0	This register updates automatically each time a read completes. The current value is the next address to be read by this channel.	RW	0x00000000

DMA: CH0_WRITE_ADDR, CH1_WRITE_ADDR, ..., CH14_WRITE_ADDR, CH15_WRITE_ADDR Registers

Offsets: 0x004, 0x044, ..., 0x384, 0x3c4

Description

DMA Channel N Write Address pointer

Table 1102.
 CH0_WRITE_ADDR,
 CH1_WRITE_ADDR, ...
 CH14_WRITE_ADDR,
 CH15_WRITE_ADDR
 Registers

Bits	Description	Type	Reset
31:0	This register updates automatically each time a write completes. The current value is the next address to be written by this channel.	RW	0x00000000

DMA: CH0_TRANS_COUNT, CH1_TRANS_COUNT, ..., CH14_TRANS_COUNT, CH15_TRANS_COUNT Registers

Offsets: 0x008, 0x048, ..., 0x388, 0x3c8

Description

DMA Channel N Transfer Count

Table 1103.
 CH0_TRANS_COUNT,
 CH1_TRANS_COUNT,
 ...
 CH14_TRANS_COUNT,
 CH15_TRANS_COUNT
 Registers

Bits	Name	Description	Type	Reset
31:28	MODE	<p>When MODE is 0x0, the transfer count decrements with each transfer until 0, and then the channel triggers the next channel indicated by CTRL_CHAIN_TO.</p> <p>When MODE is 0x1, the transfer count decrements with each transfer until 0, and then the channel re-triggers itself, in addition to the trigger indicated by CTRL_CHAIN_TO. This is useful for e.g. an endless ring-buffer DMA with periodic interrupts.</p> <p>When MODE is 0xf, the transfer count does not decrement. The DMA channel performs an endless sequence of transfers, never triggering other channels or raising interrupts, until an ABORT is raised.</p> <p>All other values are reserved. 0x0 → NORMAL 0x1 → TRIGGER_SELF 0xf → ENDLESS</p>	RW	0x0

Bits	Name	Description	Type	Reset
27:0	COUNT	<p>28-bit transfer count (256 million transfers maximum).</p> <p>Program the number of bus transfers a channel will perform before halting. Note that, if transfers are larger than one byte in size, this is not equal to the number of bytes transferred (see CTRL_DATA_SIZE).</p> <p>When the channel is active, reading this register shows the number of transfers remaining, updating automatically each time a write transfer completes.</p> <p>Writing this register sets the RELOAD value for the transfer counter. Each time this channel is triggered, the RELOAD value is copied into the live transfer counter. The channel can be started multiple times, and will perform the same number of transfers each time, as programmed by most recent write.</p> <p>The RELOAD value can be observed at CHx_DBG_TCR. If TRANS_COUNT is used as a trigger, the written value is used immediately as the length of the new transfer sequence, as well as being written to RELOAD.</p>	RW	0x0000000

DMA: CH0_CTRL_TRIG, CH1_CTRL_TRIG, ..., CH14_CTRL_TRIG, CH15_CTRL_TRIG Registers

Offsets: 0x00c, 0x04c, ..., 0x38c, 0x3cc

Description

DMA Channel N Control and Status

Table 1104.
CH0_CTRL_TRIG,
CH1_CTRL_TRIG, ...,
CH14_CTRL_TRIG,
CH15_CTRL_TRIG
Registers

Bits	Name	Description	Type	Reset
31	AHB_ERROR	Logical OR of the READ_ERROR and WRITE_ERROR flags. The channel halts when it encounters any bus error, and always raises its channel IRQ flag.	RO	0x0
30	READ_ERROR	If 1, the channel received a read bus error. Write one to clear. READ_ADDR shows the approximate address where the bus error was encountered (will not be earlier, or more than 3 transfers later)	WC	0x0
29	WRITE_ERROR	If 1, the channel received a write bus error. Write one to clear. WRITE_ADDR shows the approximate address where the bus error was encountered (will not be earlier, or more than 5 transfers later)	WC	0x0
28:27	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
26	BUSY	<p>This flag goes high when the channel starts a new transfer sequence, and low when the last transfer of that sequence completes. Clearing EN while BUSY is high pauses the channel, and BUSY will stay high while paused.</p> <p>To terminate a sequence early (and clear the BUSY flag), see CHAN_ABORT.</p>	RO	0x0
25	SNIFF_EN	<p>If 1, this channel's data transfers are visible to the sniff hardware, and each transfer will advance the state of the checksum. This only applies if the sniff hardware is enabled, and has this channel selected.</p> <p>This allows checksum to be enabled or disabled on a per-control-block basis.</p>	RW	0x0
24	BSWAP	Apply byte-swap transformation to DMA data. For byte data, this has no effect. For halfword data, the two bytes of each halfword are swapped. For word data, the four bytes of each word are swapped to reverse order.	RW	0x0
23	IRQ QUIET	<p>In QUIET mode, the channel does not generate IRQs at the end of every transfer block. Instead, an IRQ is raised when NULL is written to a trigger register, indicating the end of a control block chain.</p> <p>This reduces the number of interrupts to be serviced by the CPU when transferring a DMA chain of many small control blocks.</p>	RW	0x0
22:17	TREQ_SEL	<p>Select a Transfer Request signal. The channel uses the transfer request signal to pace its data transfer rate. Sources for TREQ signals are internal (TIMERS) or external (DREQ, a Data Request from the system).</p> <p>0x0 to 0x3a → select DREQ n as TREQ 0x3b → Select Timer 0 as TREQ 0x3c → Select Timer 1 as TREQ 0x3d → Select Timer 2 as TREQ (Optional) 0x3e → Select Timer 3 as TREQ (Optional) 0x3f → Permanent request, for unpaced transfers.</p>	RW	0x00
16:13	CHAIN_TO	<p>When this channel completes, it will trigger the channel indicated by CHAIN_TO. Disable by setting CHAIN_TO = <i>(this channel)</i>.</p> <p>Note this field resets to 0, so channels 1 and above will chain to channel 0 by default. Set this field to avoid this behaviour.</p>	RW	0x0
12	RING_SEL	<p>Select whether RING_SIZE applies to read or write addresses. If 0, read addresses are wrapped on a $(1 \ll \text{RING_SIZE})$ boundary. If 1, write addresses are wrapped.</p>	RW	0x0

Bits	Name	Description	Type	Reset
11:8	RING_SIZE	<p>Size of address wrap region. If 0, don't wrap. For values $n > 0$, only the lower n bits of the address will change. This wraps the address on a $(1 \ll n)$ byte boundary, facilitating access to naturally-aligned ring buffers.</p> <p>Ring sizes between 2 and 32768 bytes are possible. This can apply to either read or write addresses, based on value of RING_SEL.</p> <p>0x0 → RING_NONE</p>	RW	0x0
7	INCR_WRITE_REV	<p>If 1, and INCR_WRITE is 1, the write address is decremented rather than incremented with each transfer.</p> <p>If 1, and INCR_WRITE is 0, this otherwise-unused combination causes the write address to be incremented by twice the transfer size, i.e. skipping over alternate addresses.</p>	RW	0x0
6	INCR_WRITE	<p>If 1, the write address increments with each transfer. If 0, each write is directed to the same, initial address.</p> <p>Generally this should be disabled for memory-to-peripheral transfers.</p>	RW	0x0
5	INCR_READ_REV	<p>If 1, and INCR_READ is 1, the read address is decremented rather than incremented with each transfer.</p> <p>If 1, and INCR_READ is 0, this otherwise-unused combination causes the read address to be incremented by twice the transfer size, i.e. skipping over alternate addresses.</p>	RW	0x0
4	INCR_READ	<p>If 1, the read address increments with each transfer. If 0, each read is directed to the same, initial address.</p> <p>Generally this should be disabled for peripheral-to-memory transfers.</p>	RW	0x0
3:2	DATA_SIZE	<p>Set the size of each bus transfer (byte/halfword/word). READ_ADDR and WRITE_ADDR advance by this amount (1/2/4 bytes) with each transfer.</p> <p>0x0 → SIZE_BYTE</p> <p>0x1 → SIZE_HALFWORD</p> <p>0x2 → SIZE_WORD</p>	RW	0x0
1	HIGH_PRIORITY	<p>HIGH_PRIORITY gives a channel preferential treatment in issue scheduling: in each scheduling round, all high priority channels are considered first, and then only a single low priority channel, before returning to the high priority channels.</p> <p>This only affects the order in which the DMA schedules channels. The DMA's bus priority is not changed. If the DMA is not saturated then a low priority channel will see no loss of throughput.</p>	RW	0x0

Bits	Name	Description	Type	Reset
0	EN	DMA Channel Enable. When 1, the channel will respond to triggering events, which will cause it to become BUSY and start transferring data. When 0, the channel will ignore triggers, stop issuing transfers, and pause the current transfer sequence (i.e. BUSY will remain high if already high)	RW	0x0

DMA: CH0_AL1_CTRL, CH1_AL1_CTRL, ..., CH14_AL1_CTRL, CH15_AL1_CTRL Registers

Offsets: 0x010, 0x050, ..., 0x390, 0x3d0

Table 1105.
CH0_AL1_CTRL,
CH1_AL1_CTRL, ...,
CH14_AL1_CTRL,
CH15_AL1_CTRL
Registers

Bits	Description	Type	Reset
31:0	Alias for channel N CTRL register	RW	-

DMA: CH0_AL1_READ_ADDR, CH1_AL1_READ_ADDR, ..., CH14_AL1_READ_ADDR, CH15_AL1_READ_ADDR Registers

Offsets: 0x014, 0x054, ..., 0x394, 0x3d4

Table 1106.
CH0_AL1_READ_ADDR
'
CH1_AL1_READ_ADDR
,...
CH14_AL1_READ_ADDR
R,
CH15_AL1_READ_ADDR
R Registers

Bits	Description	Type	Reset
31:0	Alias for channel N READ_ADDR register	RW	-

DMA: CH0_AL1_WRITE_ADDR, CH1_AL1_WRITE_ADDR, ..., CH14_AL1_WRITE_ADDR, CH15_AL1_WRITE_ADDR Registers

Offsets: 0x018, 0x058, ..., 0x398, 0x3d8

Table 1107.
CH0_AL1_WRITE_ADDR
R,
CH1_AL1_WRITE_ADDR
R, ...
CH14_AL1_WRITE_ADDR
DR,
CH15_AL1_WRITE_ADDR
DR Registers

Bits	Description	Type	Reset
31:0	Alias for channel N WRITE_ADDR register	RW	-

DMA: CH0_AL1_TRANS_COUNT_TRIG, CH1_AL1_TRANS_COUNT_TRIG, ..., CH14_AL1_TRANS_COUNT_TRIG, CH15_AL1_TRANS_COUNT_TRIG Registers

Offsets: 0x01c, 0x05c, ..., 0x39c, 0x3dc

Table 1108.
CH0_AL1_TRANS_CO
NT_TRIG,
CH1_AL1_TRANS_CO
NT_TRIG, ...
CH14_AL1_TRANS_CO
UNT_TRIG,
CH15_AL1_TRANS_CO
UNT_TRIG Registers

Bits	Description	Type	Reset
31:0	Alias for channel N TRANS_COUNT register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.	RW	-

DMA: CH0_AL2_CTRL, CH1_AL2_CTRL, ..., CH14_AL2_CTRL, CH15_AL2_CTRL Registers

Offsets: 0x020, 0x060, ..., 0x3a0, 0x3e0

Table 1109.
 CH0_AL2_CTRL,
 CH1_AL2_CTRL, ...
 CH14_AL2_CTRL,
 CH15_AL2_CTRL
 Registers

Bits	Description	Type	Reset
31:0	Alias for channel N CTRL register	RW	-

**DMA: CH0_AL2_TRANS_COUNT, CH1_AL2_TRANS_COUNT, ...,
 CH14_AL2_TRANS_COUNT, CH15_AL2_TRANS_COUNT Registers**

Offsets: 0x024, 0x064, ..., 0x3a4, 0x3e4

Table 1110.
 CH0_AL2_TRANS_CO
 NT,
 CH1_AL2_TRANS_CO
 NT, ...
 CH14_AL2_TRANS_CO
 NT,
 CH15_AL2_TRANS_CO
 NT Registers

Bits	Description	Type	Reset
31:0	Alias for channel N TRANS_COUNT register	RW	-

**DMA: CH0_AL2_READ_ADDR, CH1_AL2_READ_ADDR, ...,
 CH14_AL2_READ_ADDR, CH15_AL2_READ_ADDR Registers**

Offsets: 0x028, 0x068, ..., 0x3a8, 0x3e8

Table 1111.
 CH0_AL2_READ_ADDR
 ,
 CH1_AL2_READ_ADDR
 , ...
 CH14_AL2_READ_ADD
 R,
 CH15_AL2_READ_ADD
 R Registers

Bits	Description	Type	Reset
31:0	Alias for channel N READ_ADDR register	RW	-

**DMA: CH0_AL2_WRITE_ADDR_TRIG, CH1_AL2_WRITE_ADDR_TRIG, ...,
 CH14_AL2_WRITE_ADDR_TRIG, CH15_AL2_WRITE_ADDR_TRIG Registers**

Offsets: 0x02c, 0x06c, ..., 0x3ac, 0x3ec

Table 1112.
 CH0_AL2_WRITE_ADD
 R_TRIG,
 CH1_AL2_WRITE_ADD
 R_TRIG, ...
 CH14_AL2_WRITE_ADD
 DR_TRIG,
 CH15_AL2_WRITE_ADD
 DR_TRIG Registers

Bits	Description	Type	Reset
31:0	Alias for channel N WRITE_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.	RW	-

DMA: CH0_AL3_CTRL, CH1_AL3_CTRL, ..., CH14_AL3_CTRL, CH15_AL3_CTRL Registers

Offsets: 0x030, 0x070, ..., 0x3b0, 0x3f0

Table 1113.
 CH0_AL3_CTRL,
 CH1_AL3_CTRL, ...
 CH14_AL3_CTRL,
 CH15_AL3_CTRL
 Registers

Bits	Description	Type	Reset
31:0	Alias for channel N CTRL register	RW	-

**DMA: CH0_AL3_WRITE_ADDR, CH1_AL3_WRITE_ADDR, ...,
 CH14_AL3_WRITE_ADDR, CH15_AL3_WRITE_ADDR Registers**

Offsets: 0x034, 0x074, ..., 0x3b4, 0x3f4

Table 1114.
 CH0_AL3_WRITE_ADD
 R,
 CH1_AL3_WRITE_ADD
 R, ...
 CH14_AL3_WRITE_ADD
 DR,
 CH15_AL3_WRITE_ADD
 DR Registers

Bits	Description	Type	Reset
31:0	Alias for channel N WRITE_ADDR register	RW	-

**DMA: CH0_AL3_TRANS_COUNT, CH1_AL3_TRANS_COUNT, ...,
 CH14_AL3_TRANS_COUNT, CH15_AL3_TRANS_COUNT Registers**

Offsets: 0x038, 0x078, ..., 0x3b8, 0x3f8

Table 1115.
~~CH0_AL3_TRANS_CO
~~NT,~~
~~CH1_AL3_TRANS_CO
~~NT, ...~~
~~CH14_AL3_TRANS_CO
~~NT,~~
~~CH15_AL3_TRANS_CO
~~NT Registers~~~~~~~~~~

Bits	Description	Type	Reset
31:0	Alias for channel N TRANS_COUNT register	RW	-

DMA: CH0_AL3_READ_ADDR_TRIG, CH1_AL3_READ_ADDR_TRIG, ..., CH14_AL3_READ_ADDR_TRIG, CH15_AL3_READ_ADDR_TRIG Registers

Offsets: 0x03c, 0x07c, ..., 0x3bc, 0x3fc

Table 1116.
~~CH0_AL3_READ_ADDR
~~_TRIG,~~
~~CH1_AL3_READ_ADDR
~~_TRIG, ...~~
~~CH14_AL3_READ_ADD
~~R_TRIG,~~
~~CH15_AL3_READ_ADD
~~R_TRIG Registers~~~~~~~~~~

Bits	Description	Type	Reset
31:0	Alias for channel N READ_ADDR register This is a trigger register (0xc). Writing a nonzero value will reload the channel counter and start the channel.	RW	-

DMA: INTR Register

Offset: 0x400

Description

Interrupt Status (raw)

Table 1117. INTR Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Raw interrupt status for DMA Channels 0..15. Bit n corresponds to channel n. Ignores any masking or forcing. Channel interrupts can be cleared by writing a bit mask to INTR or INTS0/1/2/3. Channel interrupts can be routed to either of four system-level IRQs based on INTE0, INTE1, INTE2 and INTE3. The multiple system-level interrupts might be used to allow NVIC IRQ preemption for more time-critical channels, to spread IRQ load across different cores, or to target IRQs to different security domains. It is also valid to ignore the multiple IRQs, and just use INTE0/INTS0/IRQ 0. If this register is accessed at a security/privilege level less than that of a given channel (as defined by that channel's SECCFG_CHx register), then that channel's interrupt status will read as 0, ignore writes.	WC	0x0000

DMA: INTE0 Register

Offset: 0x404

Description

Interrupt Enables for IRQ 0

Table 1118. INTFO Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Set bit n to pass interrupts from channel n to DMA IRQ 0.</p> <p>Note this bit has no effect if the channel security/privilege level, defined by SECCFG_CHx, is greater than the IRQ security/privilege defined by SECCFG_IRQ0.</p>	RW	0x0000

DMA: INTFO Register

Offset: 0x408

Description

Force Interrupts

Table 1119. INTFO Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Write 1s to force the corresponding bits in INTS0. The interrupt remains asserted until INTFO is cleared.	RW	0x0000

DMA: INTS0 Register

Offset: 0x40c

Description

Interrupt Status for IRQ 0

Table 1120. INTS0 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Indicates active channel interrupt requests which are currently causing IRQ 0 to be asserted.</p> <p>Channel interrupts can be cleared by writing a bit mask here.</p> <p>Channels with a security/privilege (SECCFG_CHx) greater SECCFG_IRQ0) read as 0 in this register, and ignore writes.</p>	WC	0x0000

DMA: INTE1 Register

Offset: 0x414

Description

Interrupt Enables for IRQ 1

Table 1121. INTF1 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Set bit n to pass interrupts from channel n to DMA IRQ 1.</p> <p>Note this bit has no effect if the channel security/privilege level, defined by SECCFG_CHx, is greater than the IRQ security/privilege defined by SECCFG_IRQ1.</p>	RW	0x0000

DMA: INTF1 Register

Offset: 0x418

Description

Force Interrupts

Table 1122. INTF1 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Write 1s to force the corresponding bits in INTS1. The interrupt remains asserted until INTF1 is cleared.	RW	0x0000

DMA: INTS1 Register

Offset: 0x41c

Description

Interrupt Status for IRQ 1

Table 1123. INTS1 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Indicates active channel interrupt requests which are currently causing IRQ 1 to be asserted.</p> <p>Channel interrupts can be cleared by writing a bit mask here.</p> <p>Channels with a security/privilege (SECCFG_CHx) greater SECCFG_IRQ1) read as 0 in this register, and ignore writes.</p>	WC	0x0000

DMA: INTE2 Register

Offset: 0x424

Description

Interrupt Enables for IRQ 2

Table 1124. INTF2 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Set bit n to pass interrupts from channel n to DMA IRQ 2.</p> <p>Note this bit has no effect if the channel security/privilege level, defined by SECCFG_CHx, is greater than the IRQ security/privilege defined by SECCFG_IRQ2.</p>	RW	0x0000

DMA: INTF2 Register

Offset: 0x428

Description

Force Interrupts

Table 1125. INTF2 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Write 1s to force the corresponding bits in INTS2. The interrupt remains asserted until INTF2 is cleared.	RW	0x0000

DMA: INTS2 Register

Offset: 0x42c

Description

Interrupt Status for IRQ 2

Table 1126. INTS2 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Indicates active channel interrupt requests which are currently causing IRQ 2 to be asserted.</p> <p>Channel interrupts can be cleared by writing a bit mask here.</p> <p>Channels with a security/privilege (SECCFG_CHx) greater SECCFG_IRQ2) read as 0 in this register, and ignore writes.</p>	WC	0x0000

DMA: INTE3 Register

Offset: 0x434

Description

Interrupt Enables for IRQ 3

Table 1127. INTF3 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Set bit n to pass interrupts from channel n to DMA IRQ 3.</p> <p>Note this bit has no effect if the channel security/privilege level, defined by SECCFG_CHx, is greater than the IRQ security/privilege defined by SECCFG_IRQ3.</p>	RW	0x0000

DMA: INTF3 Register

Offset: 0x438

Description

Force Interrupts

Table 1128. INTS3 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Write 1s to force the corresponding bits in INTS3. The interrupt remains asserted until INTF3 is cleared.	RW	0x0000

DMA: INTS3 Register

Offset: 0x43c

Description

Interrupt Status for IRQ 3

Table 1129. INTS3 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	<p>Indicates active channel interrupt requests which are currently causing IRQ 3 to be asserted.</p> <p>Channel interrupts can be cleared by writing a bit mask here.</p> <p>Channels with a security/privilege (SECCFG_CHx) greater SECCFG_IRQ3) read as 0 in this register, and ignore writes.</p>	WC	0x0000

DMA: TIMER0, TIMER1, TIMER2, TIMER3 Registers

Offsets: 0x440, 0x444, 0x448, 0x44c

Description

Pacing (X/Y) fractional timer

The pacing timer produces TREQ assertions at a rate set by $((X/Y) * \text{sys_clk})$. This equation is evaluated every sys_clk cycles and therefore can only generate TREQs at a rate of 1 per sys_clk (i.e. permanent TREQ) or less.

Table 1130. TIMER0,
TIMER1, TIMER2,
TIMER3 Registers

Bits	Name	Description	Type	Reset
31:16	X	Pacing Timer Dividend. Specifies the X value for the (X/Y) fractional timer.	RW	0x0000
15:0	Y	Pacing Timer Divisor. Specifies the Y value for the (X/Y) fractional timer.	RW	0x0000

DMA: MULTI_CHAN_TRIGGER Register

Offset: 0x450

Description

Trigger one or more channels simultaneously

Table 1131.
MULTI_CHAN_TRIGGER Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Each bit in this register corresponds to a DMA channel. Writing a 1 to the relevant bit is the same as writing to that channel's trigger register; the channel will start if it is currently enabled and not already busy.	SC	0x0000

DMA: SNIFF_CTRL Register

Offset: 0x454

Description

Sniffer Control

Table 1132.
SNIFF_CTRL Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11	OUT_INV	If set, the result appears inverted (bitwise complement) when read. This does not affect the way the checksum is calculated; the result is transformed on-the-fly between the result register and the bus.	RW	0x0
10	OUT_REV	If set, the result appears bit-reversed when read. This does not affect the way the checksum is calculated; the result is transformed on-the-fly between the result register and the bus.	RW	0x0
9	BSWAP	Locally perform a byte reverse on the sniffed data, before feeding into checksum. Note that the sniff hardware is downstream of the DMA channel byteswap performed in the read master: if channel CTRL_BSWAP and SNIFF_CTRL_BSWAP are both enabled, their effects cancel from the sniffer's point of view.	RW	0x0

Bits	Name	Description	Type	Reset
8:5	CALC	0x0 → Calculate a CRC-32 (IEEE802.3 polynomial) 0x1 → Calculate a CRC-32 (IEEE802.3 polynomial) with bit reversed data 0x2 → Calculate a CRC-16-CCITT 0x3 → Calculate a CRC-16-CCITT with bit reversed data 0xe → XOR reduction over all data. == 1 if the total 1 population count is odd. 0xf → Calculate a simple 32-bit checksum (addition with a 32 bit accumulator)	RW	0x0
4:1	DMACH	DMA channel for Sniffer to observe	RW	0x0
0	EN	Enable sniffer	RW	0x0

DMA: SNIFF_DATA Register

Offset: 0x458

Description

Data accumulator for sniff hardware

Table 1133.
SNIFF_DATA Register

Bits	Description	Type	Reset
31:0	Write an initial seed value here before starting a DMA transfer on the channel indicated by SNIFF_CTRL_DMACH. The hardware will update this register each time it observes a read from the indicated channel. Once the channel completes, the final result can be read from this register.	RW	0x00000000

DMA: FIFO_LEVELS Register

Offset: 0x460

Description

Debug RAF, WAF, TDF levels

Table 1134.
FIFO_LEVELS Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	RAF_LVL	Current Read-Address-FIFO fill level	RO	0x00
15:8	WAF_LVL	Current Write-Address-FIFO fill level	RO	0x00
7:0	TDF_LVL	Current Transfer-Data-FIFO fill level	RO	0x00

DMA: CHAN_ABORT Register

Offset: 0x464

Description

Abort an in-progress transfer sequence on one or more channels

Table 1135.
CHAN_ABORT Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-

Bits	Description	Type	Reset
15:0	Each bit corresponds to a channel. Writing a 1 aborts whatever transfer sequence is in progress on that channel. The bit will remain high until any in-flight transfers have been flushed through the address and data FIFOs. After writing, this register must be polled until it returns all-zero. Until this point, it is unsafe to restart the channel.	SC	0x0000

DMA: N_CHANNELS Register

Offset: 0x468

Table 1136.
N_CHANNELS Register

Bits	Description	Type	Reset
31:5	Reserved.	-	-
4:0	The number of channels this DMA instance is equipped with. This DMA supports up to 16 hardware channels, but can be configured with as few as one, to minimise silicon area.	RO	-

DMA: SECCFG_CH0, SECCFG_CH1, ..., SECCFG_CH14, SECCFG_CH15 Registers

Offsets: 0x480, 0x484, ..., 0x4b8, 0x4bc

Description

Security configuration for channel N . Control whether this channel performs Secure/Non-secure and Privileged/Unprivileged bus accesses.

If this channel generates bus accesses of some security level, an access of at least that level (in the order S+P > S+U > NS+P > NS+U) is required to program, trigger, abort, check the status of, interrupt on or acknowledge the interrupt of this channel.

This register automatically locks down (becomes read-only) once software starts to configure the channel.

This register is world-readable, but is writable only from a Secure, Privileged context.

Table 1137.
SECCFG_CH0,
SECCFG_CH1, ...,
SECCFG_CH14,
SECCFG_CH15
Registers

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	LOCK	LOCK is 0 at reset, and is set to 1 automatically upon a successful write to this channel's control registers. That is, a write to CTRL, READ_ADDR, WRITE_ADDR, TRANS_COUNT and their aliases. Once its LOCK bit is set, this register becomes read-only. A failed write, for example due to the write's privilege being lower than that specified in the channel's SECCFG register, will not set the LOCK bit.	RW	0x0
1	S	Secure channel. If 1, this channel performs Secure bus accesses. If 0, it performs Non-secure bus accesses. If 1, this channel is controllable only from a Secure context.	RW	0x1

Bits	Name	Description	Type	Reset
0	P	Privileged channel. If 1, this channel performs Privileged bus accesses. If 0, it performs Unprivileged bus accesses. If 1, this channel is controllable only from a Privileged context of the same Secure/Non-secure level, or any context of a higher Secure/Non-secure level.	RW	0x1

DMA: SECCFG_IRQ0, SECCFG_IRQ1, SECCFG_IRQ2, SECCFG_IRQ3 Registers

Offsets: 0x4c0, 0x4c4, 0x4c8, 0x4cc

Description

Security configuration for IRQ N. Control whether the IRQ permits configuration by Non-secure/Unprivileged contexts, and whether it can observe Secure/Privileged channel interrupt flags.

Table 1138.
SECCFG_IRQ0,
SECCFG_IRQ1,
SECCFG_IRQ2,
SECCFG_IRQ3
Registers

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	S	Secure IRQ. If 1, this IRQ's control registers can only be accessed from a Secure context. If 0, this IRQ's control registers can be accessed from a Non-secure context, but Secure channels (as per SECCFG_CHx) are masked from the IRQ status, and this IRQ's registers can not be used to acknowledge the channel interrupts of Secure channels.	RW	0x1
0	P	Privileged IRQ. If 1, this IRQ's control registers can only be accessed from a Privileged context. If 0, this IRQ's control registers can be accessed from an Unprivileged context, but Privileged channels (as per SECCFG_CHx) are masked from the IRQ status, and this IRQ's registers can not be used to acknowledge the channel interrupts of Privileged channels.	RW	0x1

DMA: SECCFG_MISC Register

Offset: 0x4d0

Description

Miscellaneous security configuration

Table 1139.
SECCFG_MISC
Register

Bits	Name	Description	Type	Reset
31:10	Reserved.	-	-	-
9	TIMER3_S	If 1, the TIMER3 register is only accessible from a Secure context, and timer DREQ 3 is only visible to Secure channels.	RW	0x1
8	TIMER3_P	If 1, the TIMER3 register is only accessible from a Privileged (or more Secure) context, and timer DREQ 3 is only visible to Privileged (or more Secure) channels.	RW	0x1
7	TIMER2_S	If 1, the TIMER2 register is only accessible from a Secure context, and timer DREQ 2 is only visible to Secure channels.	RW	0x1

Bits	Name	Description	Type	Reset
6	TIMER2_P	If 1, the TIMER2 register is only accessible from a Privileged (or more Secure) context, and timer DREQ 2 is only visible to Privileged (or more Secure) channels.	RW	0x1
5	TIMER1_S	If 1, the TIMER1 register is only accessible from a Secure context, and timer DREQ 1 is only visible to Secure channels.	RW	0x1
4	TIMER1_P	If 1, the TIMER1 register is only accessible from a Privileged (or more Secure) context, and timer DREQ 1 is only visible to Privileged (or more Secure) channels.	RW	0x1
3	TIMER0_S	If 1, the TIMER0 register is only accessible from a Secure context, and timer DREQ 0 is only visible to Secure channels.	RW	0x1
2	TIMER0_P	If 1, the TIMER0 register is only accessible from a Privileged (or more Secure) context, and timer DREQ 0 is only visible to Privileged (or more Secure) channels.	RW	0x1
1	SNIFF_S	If 1, the sniffer can see data transfers from Secure channels, and can itself only be accessed from a Secure context. If 0, the sniffer can be accessed from either a Secure or Non-secure context, but can not see data transfers of Secure channels.	RW	0x1
0	SNIFF_P	If 1, the sniffer can see data transfers from Privileged channels, and can itself only be accessed from a privileged context, or from a Secure context when SNIFF_S is 0. If 0, the sniffer can be accessed from either a Privileged or Unprivileged context (with sufficient security level) but can not see transfers from Privileged channels.	RW	0x1

DMA: MPU_CTRL Register

Offset: 0x500

Description

Control register for DMA MPU. Accessible only from a Privileged context.

Table 1140.
MPU_CTRL Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	NS_HIDE_ADDR	By default, when a region's S bit is clear, Non-secure-Privileged reads can see the region's base address and limit address. Set this bit to make the addresses appear as 0 to Non-secure reads, even when the region is Non-secure, to avoid leaking information about the processor SAU map.	RW	0x0
2	S	Determine whether an address not covered by an active MPU region is Secure (1) or Non-secure (0)	RW	0x0

Bits	Name	Description	Type	Reset
1	P	Determine whether an address not covered by an active MPU region is Privileged (1) or Unprivileged (0)	RW	0x0
0	Reserved.	-	-	-

DMA: MPU_BAR0, MPU_BAR1, ..., MPU_BAR6, MPU_BAR7 Registers

Offsets: 0x504, 0x50c, ..., 0x534, 0x53c

Description

Base address register for MPU region N . Writable only from a Secure, Privileged context.

Table 1141.
MPU_BAR0,
MPU_BAR1, ...,
MPU_BAR6,
MPU_BAR7 Registers

Bits	Name	Description	Type	Reset
31:5	ADDR	This MPU region matches addresses where $addr[31:5]$ (the 27 most significant bits) are greater than or equal to BAR_ADDR, and less than or equal to LAR_ADDR. Readable from any Privileged context, if and only if this region's S bit is clear, and MPU_CTRL_NS_HIDE_ADDR is clear. Otherwise readable only from a Secure, Privileged context.	RW	0x0000000
4:0	Reserved.	-	-	-

DMA: MPU_LAR0, MPU_LAR1, ..., MPU_LAR6, MPU_LAR7 Registers

Offsets: 0x508, 0x510, ..., 0x538, 0x540

Description

Limit address register for MPU region N . Writable only from a Secure, Privileged context, with the exception of the P bit.

Table 1142.
MPU_LAR0,
MPU_LAR1, ...,
MPU_LAR6,
MPU_LAR7 Registers

Bits	Name	Description	Type	Reset
31:5	ADDR	Limit address bits 31:5. Readable from any Privileged context, if and only if this region's S bit is clear, and MPU_CTRL_NS_HIDE_ADDR is clear. Otherwise readable only from a Secure, Privileged context.	RW	0x0000000
4:3	Reserved.	-	-	-
2	S	Determines the Secure/Non-secure (=1/0) status of addresses matching this region, if this region is enabled.	RW	0x0
1	P	Determines the Privileged/Unprivileged (=1/0) status of addresses matching this region, if this region is enabled. Writable from any Privileged context, if and only if the S bit is clear. Otherwise, writable only from a Secure, Privileged context.	RW	0x0
0	EN	Region enable. If 1, any address within range specified by the base address (BAR_ADDR) and limit address (LAR_ADDR) has the attributes specified by S and P.	RW	0x0

DMA: CH0_DBG_CTDREQ, CH1_DBG_CTDREQ, ..., CH14_DBG_CTDREQ, CH15_DBG_CTDREQ Registers

Offsets: 0x800, 0x840, ..., 0xb80, 0xbc0

Table 1143.
CH0_DBG_CTDREQ,
CH1_DBG_CTDREQ, ...,
CH14_DBG_CTDREQ,
CH15_DBG_CTDREQ
Registers

Bits	Description	Type	Reset
31:6	Reserved.	-	-
5:0	Read: get channel DREQ counter (i.e. how many accesses the DMA expects it can perform on the peripheral without overflow/underflow. Write any value: clears the counter, and cause channel to re-initiate DREQ handshake.	WC	0x00

DMA: CH0_DBG_TCR, CH1_DBG_TCR, ..., CH14_DBG_TCR, CH15_DBG_TCR Registers

Offsets: 0x804, 0x844, ..., 0xb84, 0xbc4

Table 1144.
CH0_DBG_TCR,
CH1_DBG_TCR, ...,
CH14_DBG_TCR,
CH15_DBG_TCR
Registers

Bits	Description	Type	Reset
31:0	Read to get channel TRANS_COUNT reload value, i.e. the length of the next transfer	RO	0x00000000

12.7. USB

12.7.1. Overview

NOTE

Prerequisite knowledge required

This section requires knowledge of the USB protocol. If you aren't yet familiar with the USB protocol, we recommend [USB Made Simple](#). For formal definitions of the terminology used in this section, see the [USB 2.0 Specification](#).

RP2350 contains a USB 2.0 controller that can operate as either:

- a Full Speed (FS) device (12Mbps)
- a host that can communicate with both Low Speed (LS) (1.5Mbps) and Full Speed devices, including multiple downstream devices connected to a USB hub

There is an integrated USB 1.1 PHY which interfaces the USB controller with the **DP** and **DM** pins of the chip. You may use this as 3.3 V GPIO when the USB controller is not in use.

12.7.1.1. Features

The USB controller hardware handles the low level USB protocol. The programmer must configure the controller, provide data buffers, and consume or provide data buffers in response to events on the bus. The controller interrupts the processor when it needs attention. The USB controller has 4 kB of dual-port SRAM (DPSRAM) used for configuration and data buffers.

12.7.1.1.1. Device Mode

In Device Mode, the USB controller has the following characteristics:

- USB 2.0-compatible Full Speed device (12Mbps)
- Supports up to 32 endpoints (Endpoints 0 → 15 in both in and out directions)

- Supports Control, Isochronous (ISO), Bulk, and Interrupt endpoint types
- Supports double buffering
- 3840 bytes of usable buffer space in DPSRAM. This is equivalent to 60×64 -byte buffers

12.7.1.1.2. Host Mode

In Host Mode, the USB controller can:

- communicate with Full Speed (12Mbps) devices and Low Speed devices (1.5Mbps)
- communicate with multiple devices via a USB hub, including Low Speed devices connected to a Full Speed hub
- poll up to 15 interrupt endpoints in hardware (used by hubs to notify the host of connect/disconnect events, used by mice to notify the host of movement, etc.)

12.7.1.1.3. USB DPRAM

The USB controller uses 4 kiB of dual-port SRAM (DPSRAM) to exchange data and control information with the controller. This is also referred to as dual-port RAM (DPRAM). One port is accessible from the system bus, clocked by `clk_sys`. The other port is accessible from the controller, clocked by `clk_usb`. The DPRAM is mapped in the system address space starting from `0x50100000`, `USBCTRL_DPRAM_BASE`.

The USB DPRAM supports 32-bit, 16-bit and 8-bit reads and writes. Writes complete in one cycle. Reads complete in two cycles.

You can store general user data in USB DPRAM space not required for USB controller operation. When the controller is disabled, all 4 kiB of DPRAM is available. Before accessing the DPRAM, you must take the USB controller out of reset.

Since the USB controller is in the peripheral address space, it is not accessible for processor instruction fetch. Attempting to fetch instructions from USB DPRAM unconditionally returns a bus error response, no matter the configuration of the processor SAU/MPU/PMP or the system `ACCESSCTRL` registers.

As peripheral addresses are marked Exempt in the IDAU (Section 10.2.2), the SAU configuration for this address range is ignored. Accesses to USB DPRAM are controlled only by the processor MPU/PMP and the `ACCESSCTRL` `USBCTRL` register.

12.7.2. Changes from RP2040

All changes from RP2040 are a superset of the RP2040 features. Existing software for the RP2040 USB controller will continue to work with one exception: you must clear the `MAIN_CTRL.PHY_ISO` bit at startup and after power down events. We recommend leaving the `LINESTATE_TUNING` register at its reset value. Software should not clear this register.

12.7.2.1. Errata Fixes

RP2350 fixes all RP2040 USB errata. This includes fixes for the following RP2040B0 and B1 errata which are also fixed by RP2040B2:

- RP2040-E2: USB device endpoint abort is not cleared
- RP2040-E5: USB device fails to exit RESET state on busy USB bus

For more information about RP2040B2, see the RP2040 datasheet.

RP2350 fixes the following RP2040B2 errata, which require software workarounds on RP2040B2:

- RP2040-E3: USB host: interrupt endpoint buffer done flag can be set with incorrect buffer select

- RP2040-E4: USB host writes to upper half of buffer status in single buffered mode
- RP2040-E15: USB Device controller will hang if certain bus errors occur during an IN transfer (see [Section 12.7.2.2.4](#))

12.7.2.2. New Features

12.7.2.2.1. General

- The USB PHY **DP** and **DM** can be used as regular GPIO pins. See the GPIO muxing [Table 646](#) in [Section 9.4](#).
- A **MAIN_CTRL.PHY_ISO** control isolates the PHY from the switched core power domain while the switched core domain is powered down. The isolation control resets to 1, meaning the **MAIN_CTRL.PHY_ISO** bit needs to be cleared before the PHY can be used. For more information on isolation, see [Chapter 9](#).
- **SIE_CTRL.PULLDOWN_EN** defaults to a 1 to match the reset state of isolation latches in the USB PHY. Pulling the **DP** and **DM** pins down by default saves power by preventing them from floating when unused.
- The **USB_MUXING.TO_PHY** bit defaults to a 1 to match the reset state of isolation latches.
- Added **SM_STATE**, which exposes the internal state of the controller's modules.

12.7.2.2.2. Host

- You can now optionally stop a transaction if a **NAK** is received. This allows the USB host to stop a bulk transaction if the device is not able to transfer data. Some devices using bulk endpoints, such as a UART, will return **NAKs** until a character is received. Stopping the transaction in hardware rather than using software means the host can get a **NAK** and guarantee no data has been dropped. RP2350 adds two register bits and an interrupt to support this:
 - The **NAK_POLL.STOP_EPX_ON_NAK** control, which enables and disables the feature.
 - The **NAK_POLL.EPX_STOPPED_ON_NAK** status bit, which also has an associated interrupt **INTS.EPX_STOPPED_ON_NAK**.
- RP2350 increases inter-packet and turnaround timeouts to accommodate worst-case hub delays. This issue, only seen with long chains of USB hubs, was never seen in practice. Timings in the host state machine have been corrected to match USB spec. This fix is enabled by **LINESTATE_TUNING.MULTI_HUB_FIX**.

12.7.2.2.3. Device

- Added wake from suspend fix: Any bus activity (defined as **K** or **SE0**) should cause a wake from suspend, not just a qualified period of resume signalling. This fix is enabled by default and can be disabled with **LINESTATE_TUNING.DEV_LS_WAKE_FIX** (**LS** means line state in this instance, not low speed).
- Added DPSRAM double read feature to ensure data consistency. This avoids the need to set the **AVAILABLE** bit in the buffer control register separate to the rest of the buffer information. This feature is enabled by default and controlled by **LINESTATE_TUNING.DEV_BUFF_CONTROL_DOUBLE_READ_FIX**.
- Added ability to stop **DEVICE OUT FROM HOST** when a short packet is received. For **EP0** this is controlled by **SIE_CTRL.EP0_STOP_ON_SHORT_PACKET**. This is done by stopping the transaction and then not toggling the buffer if in double buffered mode. Also added **short_packet** interrupt to notify software that a short packet has been received (**INTS.RX_SHORT_PACKET**)

12.7.2.2.4. Device error handling

- Added **DEV_RX_ERR QUIESCE** feature: the device endpoint error count replicates the host's internal Cerr count so software can detect if the host has probably halted the endpoint after three consecutive errors. The various stages of Rx decode generate their own error signals that propagate to the top level. These error signals arrive at different times, so two error interrupts generate for every failed transfer. Added an optional override for this behaviour by

forcing the device Rx controller to idle after the first instance of an error during a transfer. This fix is enabled with [LINESTATE_TUNING.DEV_RX_ERR QUIESCE](#).

- Added [SIE_RX_CHATTER_SE0_FIX](#): the existing error recovery implementation waits for 8 FS idle bit-times before signalling a framing error and returning to idle. This works OK for random bus errors, but when a hub terminates a downstream packet, the hub forces a bit-stuff error followed by EOP. A valid token from the host may immediately follow this, but the device controller may ignore it due to the enforced delay. Optionally waits for either a valid EOP or 8 idle bit times before signalling a framing error. To enable the fix, use [LINESTATE_TUNING.SIE_RX_CHATTER_SE0_FIX](#).
- Fix RP2040-E15: the receive state machine doesn't always handle cases where the bitstream deserialiser can abort a transfer. If decoding terminates due to bitstuff errors during the middle phases of a packet, the device controller can lock up. Unconditionally disables Rx if the deserialiser has flagged a bitstuff error and subsequently signalled framing error after linestate returns to idle. To enable this fix, use [LINESTATE_TUNING.SIE_RX_BITSTUFF_FIX](#).
- Device state machine watchdog: added a watchdog so that if the device state machine gets stuck for a certain amount of time it can be forced to idle. This is to handle any other error cases not anticipated by the above fixes. To enable the watchdog, use [DEV_SM_WATCHDOG](#).

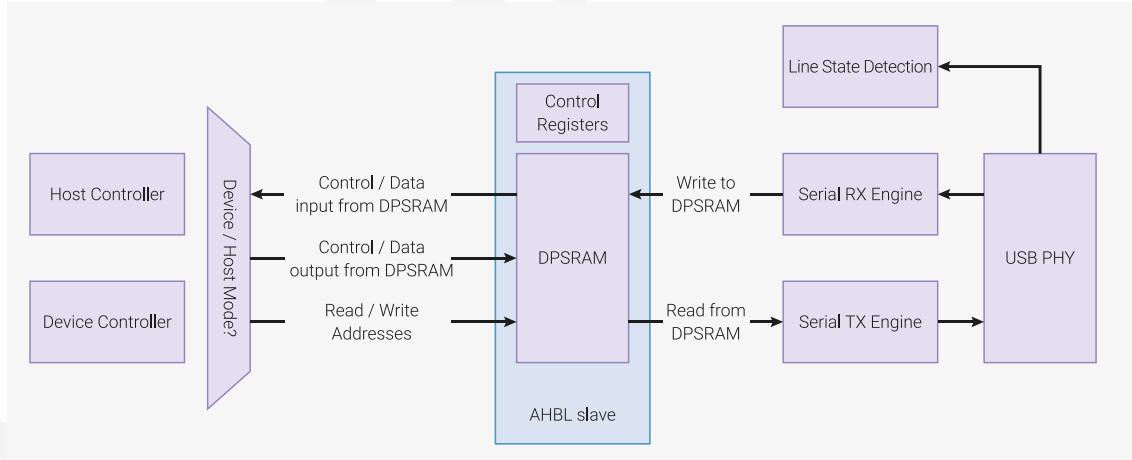
12.7.3. Architecture

12.7.3.1. Clock speed

This controller requires `clk_usb` to be running at 48MHz. `clk_sys` must also be running at $\geq 48\text{MHz}$.

12.7.3.2. Overview

Figure 100. A simplified overview of the USB controller architecture.



The USB controller is an area-efficient design that muxes a device controller or host controller onto a common set of components. Each component is detailed below.

12.7.3.3. USB PHY

The USB PHY provides the electrical interface between the USB `DP` and `DM` pins and the digital logic of the controller. The `DP` and `DM` pins are a differential pair, meaning the values are always the inverse of each other, except to encode a specific line state (e.g. [SE0](#)). The USB PHY drives the `DP` and `DM` pins to transmit data and performs a differential receive of any incoming data. The USB PHY provides both single-ended and differential receive data to the line state detection module.

The USB PHY has built in pull-up and pull-down resistors. When the controller acts as a Full Speed device, the `DP` pin is

pulled up to indicate to the host that a Full Speed device has been connected. In host mode, a weak pull-down is applied to **DP** and **DM** so that the lines are pulled to a logical zero until the device pulls up **DP** for Full Speed or **DM** for Low Speed.

12.7.3.4. Line State Detection

The [USB 2.0 Specification](#) defines several line states (Bus Reset, Connected, Suspend, Resume, Data 1, Data 0, etc.) that need to be detected. The line state detection module has several state machines to detect these states and signal events to the other hardware components. There is no shared clock signal in USB, so the Rx data must be sampled by an internal clock. The maximum data rate of USB Full Speed is 12Mbps. The Rx data is sampled at 48MHz, giving 4 clock cycles to capture and filter the bus state. The line state detection module distributes the filtered Rx data to the Serial Rx Engine.

12.7.3.5. Serial Rx Engine

The serial receive (Rx) engine decodes receive data captured by the line state detection module. It produces the following information:

- The **PID** of the incoming data packet
- The device address for the incoming data
- The device endpoint for the incoming data
- Data bytes

The serial receive engine also detects errors in Rx data by performing a CRC check on the incoming data. Any errors are signalled to the other hardware blocks and can raise an interrupt.

NOTE

If you disconnect the USB cable during packet transfer in either host or device mode, the hardware will raise errors. Software must account for this scenario if you enable error interrupts.

12.7.3.6. Serial Tx Engine

The serial transmit (Tx) engine is a mirror of the serial receive engine. It is connected to the currently active controller (either device or host). It creates **TOKEN** and **DATA** packets, calculates the CRC, and transmits them on the bus.

12.7.3.7. DPSRAM

The USB controller uses 4 kiB (4096 bytes) of Dual Port SRAM (DPSRAM) to store control registers and data buffers. The DPSRAM is accessible as a 32-bit wide memory at address 0 of the USB controller ([0x50100000](#)).

The DPSRAM has the following characteristics, which differ from most registers on RP2350:

- Supports 8-bit, 16-bit, and 32-bit accesses (typically, RP2350 registers only support 32-bit accesses)
- Does *not* support set/clear aliases. (typically, RP2350 registers support these)

Data Buffers are typically 64 bytes long, as this is the maximum normal packet size for most Full Speed packets. Isochronous endpoints support a maximum buffer size of 1023 bytes. For other packet types, the maximum size is 64 bytes per buffer.

12.7.3.7.1. Concurrent access

The DPSRAM in the USB controller is asynchronous. The **dual port** part of the name indicates that *both* the processor and the USB controller have ports to read and write, and these two ports are in different clock domains. As a result, the processor and USB controller can access the same memory address at the same time. One could write and one could read simultaneously. This could result in inconsistent data reads. You can avoid this scenario by following the rules outlined in this section.

The **AVAILABLE** bit in the buffer control register indicates who has ownership of a buffer. Set this bit to 1 from the processor to give the controller ownership of the buffer. When it has finished using the buffer, the controller sets the bit back to 0. Set the **AVAILABLE** bit separately from the rest of the data in the buffer control register so that the rest of the data in the buffer control register is accurate when the **AVAILABLE** bit is set.

This is necessary because the processor clock **clk_sys** can run several times faster than the **clk_usb** clock. Therefore **clk_sys** can update the data *during* a USB controller read on a slower clock. The correct process is:

1. Write buffer information (length, etc.) to the buffer control register.
2. **nop** for some **clk_sys** cycles to ensure that at least one **clk_usb** cycle passes. Consider a scenario where **clk_sys** runs at 125MHz and **clk_usb** runs at 48MHz. Because $\lceil \frac{125}{48} \rceil = 3$, you should issue 3 **nop** instructions between the writes to guarantee that at least one **clk_usb** cycle has passed.
3. Set the **AVAILABLE** bit.

If **clk_sys** and **clk_usb** run at the same frequency, then it is not necessary to set the **AVAILABLE** bit separately.

i NOTE

When the USB controller writes the status back to the DPSRAM, it does a 16-bit write to the lower 2 bytes for buffer 0 and the upper 2 bytes for buffer 1. When using double-buffered mode, always treat the buffer control register as two 16-bit registers when updating it in software.

12.7.3.7.2. Layout

Addresses **0x0** → **0xff** are used for control registers containing configuration data. The remaining space, addresses **0x100** → **0xffff** (3840 bytes) can be used for data buffers. The controller has control registers that start at address **0x10000**.

The memory layout depends on the USB controller mode:

- In Device mode, the host can access multiple endpoints, so each endpoint must have endpoint control and buffer control registers.
- In Host mode, the host software running on the processor decides which endpoints and devices to access. This only requires one set of endpoint control and buffer control registers. As well as software-driven transfers, the host controller can poll up to 15 interrupt endpoints and has a register for each of these interrupt endpoints.

Table 1145. DPSRAM layout

Offset	Device Function	Host Function
0x0	Setup packet (8 bytes)	
0x8	EP1 in control	Interrupt endpoint control 1
0xc	EP1 out control	Spare
0x10	EP2 in control	Interrupt endpoint control 2
0x14	EP2 out control	Spare
0x18	EP3 in control	Interrupt endpoint control 3
0x1c	EP3 out control	Spare
0x20	EP4 in control	Interrupt endpoint control 4

Offset	Device Function	Host Function
0x24	EP4 out control	Spare
0x28	EP5 in control	Interrupt endpoint control 5
0x2c	EP5 out control	Spare
0x30	EP6 in control	Interrupt endpoint control 6
0x34	EP6 out control	Spare
0x38	EP7 in control	Interrupt endpoint control 7
0x3c	EP7 out control	Spare
0x40	EP8 in control	Interrupt endpoint control 8
0x44	EP8 out control	Spare
0x48	EP9 in control	Interrupt endpoint control 9
0x4c	EP9 out control	Spare
0x50	EP10 in control	Interrupt endpoint control 10
0x54	EP10 out control	Spare
0x58	EP11 in control	Interrupt endpoint control 11
0x5c	EP11 out control	Spare
0x60	EP12 in control	Interrupt endpoint control 12
0x64	EP12 out control	Spare
0x68	EP13 in control	Interrupt endpoint control 13
0x6c	EP13 out control	Spare
0x70	EP14 in control	Interrupt endpoint control 14
0x74	EP14 out control	Spare
0x78	EP15 in control	Interrupt endpoint control 15
0x7c	EP15 out control	Spare
0x80	EP0 in buffer control	EPx buffer control
0x84	EP0 out buffer control	Spare
0x88	EP1 in buffer control	Interrupt endpoint buffer control 1
0x8c	EP1 out buffer control	Spare
0x90	EP2 in buffer control	Interrupt endpoint buffer control 2
0x94	EP2 out buffer control	Spare
0x98	EP3 in buffer control	Interrupt endpoint buffer control 3
0x9c	EP3 out buffer control	Spare
0xa0	EP4 in buffer control	Interrupt endpoint buffer control 4
0xa4	EP4 out buffer control	Spare
0xa8	EP5 in buffer control	Interrupt endpoint buffer control 5
0xac	EP5 out buffer control	Spare
0xb0	EP6 in buffer control	Interrupt endpoint buffer control 6

Offset	Device Function	Host Function
0xb4	EP6 out buffer control	Spare
0xb8	EP7 in buffer control	Interrupt endpoint buffer control 7
0xbc	EP7 out buffer control	Spare
0xc0	EP8 in buffer control	Interrupt endpoint buffer control 8
0xc4	EP8 out buffer control	Spare
0xc8	EP9 in buffer control	Interrupt endpoint buffer control 9
0xcc	EP9 out buffer control	Spare
0xd0	EP10 in buffer control	Interrupt endpoint buffer control 10
0xd4	EP10 out buffer control	Spare
0xd8	EP11 in buffer control	Interrupt endpoint buffer control 11
0xdc	EP11 out buffer control	Spare
0xe0	EP12 in buffer control	Interrupt endpoint buffer control 12
0xe4	EP12 out buffer control	Spare
0xe8	EP13 in buffer control	Interrupt endpoint buffer control 13
0xec	EP13 out buffer control	Spare
0xf0	EP14 in buffer control	Interrupt endpoint buffer control 14
0xf4	EP14 out buffer control	Spare
0xf8	EP15 in buffer control	Interrupt endpoint buffer control 15
0xfc	EP15 out buffer control	Spare
0x100	EP0 buffer 0 (shared between in and out)	EPx control
0x140	Optional EP0 buffer 1	Spare
0x180	Data buffers	

12.7.3.7.3. Endpoint Control Register

The endpoint control register is used to configure an endpoint. It defines:

- The endpoint type
- The base address of the endpoint's data buffer (or data buffers if double-buffered)
- Which endpoint events trigger the controller interrupt output

A device must support Endpoint 0 so that it can reply to [SETUP](#) packets and be enumerated. As a result, there is no endpoint control register for EP0. Its buffers begin at [0x100](#). All other endpoints can have either single or dual buffers and are mapped at the base address programmed. As EP0 has no endpoint control register, the interrupt enable controls for EP0 come from [SIE_CTRL](#).

Table 1146. Endpoint control register layout

Bit(s)	Device Function	Host Function
31	Endpoint enable	
30	Single buffered (64 bytes) = 0, Double buffered (64 bytes × 2) = 1	
29	Enable interrupt for every transferred buffer	

Bit(s)	Device Function	Host Function
28	Enable interrupt for every 2 transferred buffers (valid for double-buffered only)	
27:26	Endpoint Type: Control = 0, Isochronous = 1, Bulk = 2, Interrupt = 3	
25:18	N/A	The interval the host controller should poll this endpoint. Only applicable for interrupt endpoints. Specified in ms - 1. For example: a value of 9 would poll the endpoint every 10ms.
17	Interrupt on STALL	
16	Interrupt on NAK	
15:6	Address base offset in DPSRAM of data buffer(s)	

NOTE

The data buffer base address must be 64-byte aligned, since bits 0 through 5 are ignored.

12.7.3.7.4. Buffer Control Register

The buffer control register contains information about the state of the data buffers for that endpoint. It is shared between the processor and the controller. If the endpoint is configured to be single-buffered, only the first half (bits 0 through 15) of the buffer are used.

If double buffering, the buffer select starts at buffer 0. From then on, the buffer select flips between buffer 0 and 1 unless the **reset buffer select** bit is set (which resets the buffer select to buffer 0). The value of the buffer select is internal to the controller and not accessible by the processor.

For host interrupt and isochronous packets on **EPx**, the buffer full bit will be set on completion even if the transfer was unsuccessful. To determine the error, read the error bits in the **SIE_STATUS** register.

Table 1147. Buffer control register layout

Bit(s)	Function
31	Buffer 1 full. Should be set to 1 by the processor for an IN transaction and 0 for an OUT transaction. The controller sets this to 1 for an OUT transaction because it has filled the buffer. The controller sets it to 0 for an IN transaction because it has emptied the buffer. Only valid when double buffering.
30	Last buffer of transfer for buffer 1. Only valid when double buffering.
29	Data PID for buffer 1 - DATA0 = 0, DATA1 = 1. Only valid when double buffering.
27:28	Double buffer offset for isochronous mode (0 = 128, 1 = 256, 2 = 512, 3 = 1024).
26	Buffer 1 available. Whether the buffer can be used by the controller for a transfer. The processor sets this to 1 when the buffer is configured. The controller sets this to 0 after it has sent the data to the host for an IN transaction, or filled the buffer with data from the host for an OUT transaction. Only valid when double buffering.
25:16	Buffer 1 transfer length. Only valid when double buffering.
15	Buffer 0 full. Should be set to 1 by the processor for an IN transaction and 0 for an OUT transaction. The controller sets this to 1 for an OUT transaction because it has filled the buffer. The controller sets it to 0 for an IN transaction because it has emptied the buffer.
14	Last buffer of transfer for buffer 0.
13	Data PID for buffer 0 - DATA0 = 0, DATA1 = 1.
12	Reset buffer select to buffer 0 - cleared at end of transfer. For device <i>only</i> .
11	Send STALL for device, STALL received for host.

Bit(s)	Function
10	Buffer 0 available. Indicates whether the buffer can be used by the controller for a transfer. The processor sets this to 1 when the buffer is configured. The controller sets this to 0 after it has sent the data to the host for an IN transaction or filled the buffer with data from the host for an OUT transaction.
9:0	Buffer 0 transfer length.

⚠️ WARNING

If you run `clk_sys` and `clk_usb` at different speeds, set the available and stall bits *after* the other data in the buffer control register. Otherwise, the controller may initiate a transaction with data from a previous packet. The controller could see the available bit set, but get the data PID or length from the previous packet.

12.7.3.8. Device Controller

This section details how the device controller operates when it receives various packet types from the host.

12.7.3.8.1. SETUP

The device controller **MUST** always accept a **SETUP** packet from the host. DPSRAM dedicates its first 8 bytes to the setup packet.

The [USB 2.0 Specification](#) states that receiving a setup packet also clears any stall bits on `EP0`. For this reason, the stall bits for `EP0` are gated with two bits in the `EP_STALL_ARM` register. These bits are cleared when a setup packet is received. This means that to send a stall on `EP0`, you must set both the stall bit in the buffer control register and the appropriate bit in `EP_STALL_ARM`.

Barring any errors, the setup packet will be put into the setup packet buffer at DPSRAM offset `0x0`. The device controller will then reply with an **ACK**.

Finally, `SIE_STATUS.SETUP_REC` is set to indicate that a setup packet has been received. This will trigger an interrupt if the programmer has enabled the `SETUP_REC` interrupt (see `INTE`).

12.7.3.8.2. IN

From the device's point of view, an **IN** transfer means transferring data *into* the host. When an **IN** token is received from the host, the request is handled as follows:

TOKEN phase:

1. If **STALL** is set in the buffer control register (and if `EP0`, the appropriate `EP_STALL_ARM` bit is set), send a **STALL** response and go to idle.
2. If **AVAILABLE** and **FULL** bits are set in buffer control, go to the **DATA** phase.
3. If this is an isochronous endpoint, go to idle.
 - o Otherwise, send **NAK** and go to the **DATA** phase.

DATA phase:

1. Send data.
2. If this is an isochronous endpoint, go to idle.
 - o Otherwise, go to the **ACK** phase.

ACK phase:

1. Wait for **ACK** packet from host.
2. If there is a timeout, raise a timeout error.
3. If **ACK** is received, the packet is done, so go to **STATUS** phase.

STATUS phase:

1. If this was the last buffer in the transfer (i.e. if the **LAST_BUFFER** bit in the buffer control register was set), set **SIE_STATUS.TRANS_COMPLETE**.
2. If the endpoint is double buffered, flip the buffer select to the other buffer.
3. Set a bit in **BUFF_STATUS** to indicate the buffer is done. When handling this event, the programmer should read **BUFF_CPU_SHOULD_HANDLE** to see if it is buffer 0 or buffer 1 that is finished. If the endpoint is double-buffered, both buffers could be done. The cleared **BUFF_STATUS** bit will be set again, and **BUFF_CPU_SHOULD_HANDLE** will change in this instance.
4. Update status in the appropriate half of the buffer control register: **length**, **pid**, and **last_buff** are set. Everything else is written to zero.

If the host receives a **NAK**, the host will retry again later.

12.7.3.8.3. **OUT**

When an **OUT** token is received from the host, the request is handled as follows:

TOKEN phase:

1. If this is *not* an Isochronous endpoint and the data PID does not match the buffer control register, raise **SIE_STATUS.DATA_SEQ_ERROR** (isochronous data is always sent with a **DATA0** pid).
2. If the **AVAILABLE** bit is set and the **FULL** bit is clear, go to the **DATA** phase, unless the **STALL** bit is set in which case the device controller will reply with a **STALL**.

DATA phase:

1. Store received data in buffer. If this is an isochronous endpoint, go to the **STATUS** phase. Otherwise, go to the **ACK** phase.

ACK phase:

1. Send **ACK**. Go to the **STATUS** phase.

STATUS phase:

See **IN STATUS** phase: [\[usb-device-in-status-phase\]](#). There is one difference: the **FULL** bit is set in the buffer control register to indicate that data has been received. In the **IN** phase, the **FULL** bit is cleared to indicate that data has been sent.

12.7.3.8.4. **Suspend and Resume**

The USB device controller supports suspend, resume, and device-initiated remote resume (triggered with **SIE_CTRL.RESUME**). There is an interrupt / status bit in **SIE_STATUS**. It is not necessary to enable the suspend and resume interrupts, since suspend and resume are irrelevant to most devices.

The device goes into suspend when it does not see any start of frame packets (transmitted every 1ms) from the host.

NOTE

If you enable the suspend interrupt, it is likely you will see a suspend interrupt when the device first connects, but the bus is idle. The bus can be idle for a few milliseconds before the host begins sending start of frame packets. If you do not have a VBUS detect circuit connected, you will also see a suspend interrupt when the device disconnects. Without VBUS detection, it is impossible to tell the difference between being disconnected and suspended.

12.7.3.9. Host Controller

The host controller design is similar to the device controller. The host starts all transactions, so the host always deals with transactions it has started. For this reason, there is only one set of endpoint control and endpoint buffer control registers. The host controller also contains additional hardware to poll interrupt endpoints in the background when there are no software controlled transactions taking place.

The host needs to send keep-alive packets to the device every 1ms to keep the device from suspending. Full Speed mode uses a **SOF** (start of frame) packet. Low Speed mode uses an **EOP** (end of packet) instead. Set **SIE_CTRL.KEEP_ALIVE_EN** and **SIE_CTRL.SOF_EN** to enable these packets.

Several bits in **SIE_CTRL** are used to begin a host transaction:

- **SEND_SETUP** - Send a setup packet. Typically used with **RECEIVE_TRANS**, so the setup packet will be sent followed by the additional data transaction expected from the device.
- **SEND_TRANS** - This transfer is **OUT** from the host.
- **RECEIVE_TRANS** - This transfer is **IN** to the host.
- **START_TRANS** - Start the transfer (non-latching).
- **STOP_TRANS** - Stop the current transfer (non-latching).
- **PREAMBLE_ENABLE** - Used to send a packet to a Low Speed device on a Full Speed hub. Sends a **PRE** token packet before every packet the host sends (i.e. **PRE, TOKEN, PRE, DATA, pre, ACK**).
- **SOF_SYNC** - Used to delay the transaction until after the next **SOF**. Useful for interrupt and isochronous endpoints. The host controller prevents a transaction of 64 bytes from clashing with the **SOF** packets. For longer isochronous packets, software is responsible for preventing collisions. To prevent collisions in software, use **SOF_SYNC** and limit the number of packets sent in one frame. If a transaction is set up with multiple packets, **SOF_SYNC** only applies to the first packet.

The **START_TRANS** bit is synchronised separately from other control bits in the **SIE_CTRL** register because the processor clock **clk_sys** can be asynchronous to the **clk_usb** clock. Always set the **START_TRANS** bit separately from the rest of the data in the **SIE_CTRL** register. Always ensure that at least two **clk_usb** cycles pass between writing to **START_TRANS** and other bits in **SIE_CTRL**. This ensures that the register contents are stable when the controller is prompted to start a transfer.

Consider a scenario where **clk_sys** runs at 125MHz and **clk_usb** runs at 48MHz. Because $\lceil \frac{125}{48} \rceil \times 2 = 6$, you should issue 6 **nop** instructions between the writes to guarantee that at least two **clk_usb** cycles have passed.

12.7.3.9.1. SETUP

The **SETUP** packet sent from the host always comes from the dedicated 8 bytes of space at offset **0x0** of the DPSRAM. Like the device controller, there are no control registers associated with the setup packet. The parameters are hard-coded and loaded into the hardware when you write to **START_TRANS** with the **SEND_SETUP** bit set. Once the setup packet has been sent, the host state machine waits for an **ACK** from the device. If there is a timeout, an **RX_TIMEOUT** error will be raised. If the **SEND_TRANS** bit is set, the host state machine will move to the **OUT** phase. Typically, the **SEND_SETUP** packet is used with the **RECEIVE_TRANS** bit, so the controller moves to the **IN** phase after sending a setup packet.

12.7.3.9.2. IN

An **IN** transfer is triggered with the **RECEIVE_TRANS** bit set when the **START_TRANS** bit is set. If the **SEND_SETUP** bit was set, this may be preceded by a **SETUP** packet.

CONTROL phase:

1. Read the **EPx** control register located at **0x80** to get the following endpoint information:
 - o Is it double buffered?
 - o What interrupts are enabled?
 - o Base address of the data buffer (data buffers if in double-buffered mode)
 - o What is the endpoint type?
2. Read the **EPx** buffer control register at **0x100** to get endpoint buffer information, such as transfer length and data PID.
3. Set the **AVAILABLE** bit (the host state machine checks for it).
4. Clear the **FULL** bit.

TOKEN phase:

1. Send the **IN** token packet to the device. The target device address and endpoint come from the **ADDR_ENDP** register.

DATA phase:

1. Receive the first data packet from the device.
2. Raise Rx timeout error if the device doesn't reply.
3. If this is *not* an Isochronous endpoint and the data PID does not match the buffer control register, raise **SIE_STATUS.DATA_SEQ_ERROR** (isochronous data is always sent with a **DATA0** pid).

ACK phase:

1. Send **ACK** to device.

STATUS phase:

1. Set the **BUFF_STATUS** bit and update the buffer control register.
2. Set **FULL**, **DATA_PID**, **WR_LEN**, and **LAST_BUFF** if applicable.
3. If this is the last buffer in the transfer, set **TRANS_COMPLETE**.

CONTROL phase (continued):

The host state machine performs **IN** transactions until **LAST_BUFF** is seen in the **buffer_control** register.

If the host is in double buffered mode, the host controller toggles between the **BUF0** and **BUF1** sections of the buffer control register.

Otherwise, the controller reads the buffer control register for buffer 0, then waits for **FULL** to be clear and **AVAILABLE** to be set before starting the next **IN** transaction, waiting in the **CONTROL** phase.

If the host receives a zero length packet, the device has no more data. The host state machine stops listening for more data regardless of if the **LAST_BUFF** flag was set or not. To detect this from host software, check **BUFF_DONE** for a data length of 0 in the buffer control register.

12.7.3.9.3. OUT

An **OUT** transfer is triggered with the **SEND_TRANS** bit set when the **START_TRANS** bit is set. This may be preceded by a **SETUP** packet if the **SEND_SETUP** bit was set.

CONTROL phase:

1. Read the **EPx** control register to get endpoint information (same as [Section 12.7.3.9.2](#)).
2. Read the **EPx** buffer control register to get the transfer length and data PID. **AVAILABLE** and **FULL** must be set before the transfer can start.

TOKEN phase

1. Send an **OUT** packet to the device. The target device address and endpoint come from the **ADDR_ENDP** register.

DATA phase:

1. Send the first data packet to the device. If the endpoint type is isochronous, there is no **ACK** phase, so the host controller goes straight to status phase. If **ACK** is received, go to status phase. Otherwise:
 - o If the host receives no reply, raise **SIE_STATUS.RX_TIMEOUT**.
 - o If the host receives **NAK**, raise **SIE_STATUS.NAK_REC** and send the data packet again.
 - o If the host receives **STALL**, raise **SIE_STATUS.STALL_REC** and go to idle.

STATUS phase:

1. Set the **BUFF_STATUS** bit and update the buffer control register. **FULL** will be set to 0. **TRANS_COMPLETE** will be set if this is the last buffer in the transfer.

CONTROL phase (continued):

1. If this isn't the last buffer in the transfer, wait for **FULL** and **AVAILABLE** to be set in the **EPx** buffer control register again.

12.7.3.9.4. Interrupt Endpoints

The host controller can poll interrupt endpoints on a maximum of 15 endpoints. To enable interrupt endpoints, the programmer must:

- Pick the next free interrupt endpoint slot on the host controller (starting at 1, to a maximum of 15).
- Program the appropriate endpoint control register and buffer control register like you would with a normal **IN** or **OUT** transfer. Because interrupt endpoints are single-buffered, the **BUF1** part of the buffer control register is invalid.
- Set the address and endpoint of the device in the appropriate **ADDR_ENDP** register (**ADDR_ENDP1** to **ADDR_ENDP15**). If the device is Low Speed but attached to a Full Speed hub, the preamble bit should be set. The endpoint **direction** bit should also be set.
- Set the corresponding interrupt endpoint active bit (one of bits 1 through 15) in **INT_EP_CTRL**.

Typically, interrupt endpoints use an **IN** transfer. The host might poll a USB hub to see if the state of any of its ports have changed. If there is no change, the hub replies with a **NAK** to the controller, and nothing happens. Similarly, a mouse replies with a **NAK** unless the mouse has been moved since the last time the interrupt endpoint was polled.

Interrupt endpoints are polled by the controller once a **SOF** packet has been sent by the host controller.

The controller loops from 1 to 15 and attempts to poll any interrupt endpoint with the **EP_ACTIVE** bit set to 1 in **INT_EP_CTRL**. The controller will then read the endpoint control register and the buffer control register to see if there is an available buffer (i.e. **FULL** + **AVAILABLE** if an **OUT** transfer and **NOT FULL** + **AVAILABLE** for an **IN** transfer). If not, the controller will move onto the next interrupt endpoint slot.

If there is an available buffer, the transfer is dealt with the same as a normal **IN** or **OUT** transfer and the **BUFF_DONE** flag in **BUFF_STATUS** will be set when the interrupt endpoint has a valid buffer.

12.7.3.10. VBUS Control

The USB controller can be connected to GPIO pins (see [Chapter 9](#)) for the following VBUS controls:

- **VBUS enable**, used to enable VBUS in host mode. Set in **SIE_CTRL**.

- **VBUS detect**, used to detect that VBUS is present in device mode. Set via a bit in `SIE_STATUS`. Can also raise a `VBUS_DETECT` interrupt enabled in `INTE`.
- **VBUS overcurrent**, used to detect an overcurrent event. Applicable to both device and host. VBUS overcurrent is a bit in `SIE_STATUS`.

It is not necessary to connect up any of these pins to GPIO. The host can permanently supply VBUS and detect a device being connected when either the `DP` or `DM` pin is pulled high. VBUS detect can be forced in `USB_PWR`.

12.7.4. Programmer's Model

12.7.4.1. TinyUSB

The RP2350 TinyUSB port is the reference implementation for this USB controller. This port can be found in:

https://asic-git.pitowers.org/amethyst/tinyusb/-/blob/master/src/portable/raspberrypi/rp2040/dcd_rp2040.c

https://asic-git.pitowers.org/amethyst/tinyusb/-/blob/master/src/portable/raspberrypi/rp2040/hcd_rp2040.c

https://asic-git.pitowers.org/amethyst/tinyusb/-/blob/master/src/portable/raspberrypi/rp2040/rp2040_usb.h

12.7.4.2. Standalone Device Example

A standalone USB device example, `dev_lowlevel`, makes it easier to understand how to interact with the USB controller without needing to understand the TinyUSB abstractions. In addition to endpoint 0, the standalone device has two bulk endpoints: `EP1 OUT` and `EP2 IN`. The device is designed to send whatever data it receives on `EP1` to `EP2`. The example comes with a small Python script that writes "Hello World" into `EP1` and checks that it is correctly received on `EP2`.

The code included in this section explains setting up the USB device controller to receive. It also shows how software responds to a setup packet received from the host.

Figure 101. USB analyser trace of the `dev_lowlevel` USB device example. The control transfers are the device enumeration. The first bulk OUT (out from the host) transfer, highlighted in blue, is the host sending "Hello World" to the device. The second bulk transfer IN (in to the host), is the device returning "Hello World" to the host.

Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
0	S	GET	0	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
1	S	SET	0	0	SET_ADDRESS	New address 7	0x0000	0
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
2	S	GET	7	0	GET_DESCRIPTOR	DEVICE type	0x0000	DEVICE Descriptor
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
3	S	GET	7	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	CONFIGURATION Descriptor
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
4	S	GET	7	0	GET_DESCRIPTOR	CONFIGURATION type, Index 0	0x0000	4 Descriptors
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
5	S	GET	7	0	GET_DESCRIPTOR	STRING type, LANGID codes requested	Language ID 0x0000	Lang Supported
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
6	S	GET	7	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x0409	Pico Test Device
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descriptors
7	S	GET	7	0	GET_DESCRIPTOR	STRING type, Index 1	Language ID 0x0409	Raspberry Pi
Transfer	F	Control	ADDR	ENDP	bRequest	wValue	wIndex	wLength
8	S	SET	7	0	SET_CONFIGURATION	New Configuration 1	0x0000	0
Transfer	F	Bulk	ADDR	ENDP	Bytes Transferred			
9	S	OUT	7	1	12			
Transfer	F	Bulk	ADDR	ENDP	Bytes Transferred			
10	S	IN	7	2	12			

12.7.4.2.1. Device Controller Initialisation

The following code initialises the USB device:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c Lines 183 - 217

```

183 void usb_device_init() {
184     // Reset usb controller
185     reset_unreset_block_num_wait_blocking(RESET_USBCTRL);
186
187     // Clear any previous state in dpram just in case
188     memset(usb_dpram, 0, sizeof(*usb_dpram)); ①
189
190     // Enable USB interrupt at processor
191     irq_set_enabled(USBCTRL_IRQ, true);
192
193     // Mux the controller to the onboard usb phy
194     usb_hw->muxing = USB_USB_MUXING_TO_PHY_BITS | USB_USB_MUXING_SOFTCON_BITS;
195
196     // Force VBUS detect so the device thinks it is plugged into a host
197     usb_hw->pwr = USB_USB_PWR_VBUS_DETECT_BITS | USB_USB_PWR_VBUS_DETECT_OVERRIDE_EN_BITS;
198
199     // Enable the USB controller in device mode.
200     usb_hw->main_ctrl = USB_MAIN_CTRL_CONTROLLER_EN_BITS;
201
202     // Enable an interrupt per EP0 transaction
203     usb_hw->sie_ctrl = USB_SIE_CTRL_EP0_INT_1BUF_BITS; ②
204
205     // Enable interrupts for when a buffer is done, when the bus is reset,
206     // and when a setup packet is received
207     usb_hw->inte = USB_INTS_BUFF_STATUS_BITS |
208                     USB_INTS_BUS_RESET_BITS |
209                     USB_INTS_SETUP_REQ_BITS;
210
211     // Set up endpoints (endpoint control registers)
212     // described by device configuration
213     usb_setup_endpoints();
214
215     // Present full speed device by enabling pull up on DP
216     usb_hw_set->sie_ctrl = USB_SIE_CTRL_PULLUP_EN_BITS;
217 }
```

12.7.4.2.2. Configuring the Endpoint Control Registers for EP1 and EP2

The function `usb_configure_endpoints` loops through each endpoint defined in the device configuration (including EP0 in and EP0 out, which don't have an endpoint control register defined) and calls the `usb_configure_endpoint` function. This sets up the endpoint control register for that endpoint:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c Lines 149 - 164

```

149 void usb_setup_endpoint(const struct usb_endpoint_configuration *ep) {
150     printf("Set up endpoint 0x%x with buffer address 0x%p\n", ep->descriptor-
>bEndpointAddress, ep->data_buffer);
151
152     // EP0 doesn't have one so return if that is the case
153     if (!ep->endpoint_control) {
154         return;
155     }
156
157     // Get the data buffer as an offset of the USB controller's DPRAM
158     uint32_t dpram_offset = usb_buffer_offset(ep->data_buffer);
159     uint32_t reg = EP_CTRL_ENABLE_BITS
160                 | EP_CTRL_INTERRUPT_PER_BUFFER
161                 | (ep->descriptor->bmAttributes << EP_CTRL_BUFFER_TYPE_LSB)
```

```

162             | dpram_offset;
163     *ep->endpoint_control = reg;
164 }
```

12.7.4.2.3. Receiving a Setup Packet

An interrupt is raised when a setup packet is received, so the interrupt handler must handle this event:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c Lines 494 - 504

```

494 void isr_usbctrl(void) {
495     // USB interrupt handler
496     uint32_t status = usb_hw->ints;
497     uint32_t handled = 0;
498
499     // Setup packet received
500     if (status & USB_INTS_SETUP_REQ_BITS) {
501         handled |= USB_INTS_SETUP_REQ_BITS;
502         usb_hw_clear->sie_status = USB_SIE_STATUS_SETUP_REC_BITS;
503         usb_handle_setup_packet();
504     }
```

The controller writes the **SETUP** packet to the first 8 bytes of the DPSRAM, so the setup packet handler casts that area of memory to `struct usb_setup_packet *`:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c Lines 383 - 427

```

383 void usb_handle_setup_packet(void) {
384     volatile struct usb_setup_packet *pkt = (volatile struct usb_setup_packet *) &usb_dpram
385     ->setup_packet;
386     uint8_t req_direction = pkt->bmRequestType;
387     uint8_t req = pkt->bRequest;
388
389     // Reset PID to 1 for EP0 IN
390     usb_get_endpoint_configuration(EP0_IN_ADDR)->next_pid = 1u;
391
392     if (req_direction == USB_DIR_OUT) {
393         if (req == USB_REQUEST_SET_ADDRESS) {
394             usb_set_device_address(pkt);
395         } else if (req == USB_REQUEST_SET_CONFIGURATION) {
396             usb_set_device_configuration(pkt);
397         } else {
398             usb_acknowledge_out_request();
399             printf("Other OUT request (0x%u)\r\n", pkt->bRequest);
400         }
401     } else if (req_direction == USB_DIR_IN) {
402         if (req == USB_REQUEST_GET_DESCRIPTOR) {
403             uint16_t descriptor_type = pkt->wValue >> 8;
404
405             switch (descriptor_type) {
406                 case USB_DT_DEVICE:
407                     usb_handle_device_descriptor(pkt);
408                     printf("GET DEVICE DESCRIPTOR\r\n");
409                     break;
410
411                 case USB_DT_CONFIG:
412                     usb_handle_config_descriptor(pkt);
413                     printf("GET CONFIG DESCRIPTOR\r\n");
414                     break;
415             }
416         }
417     }
418 }
```

```

414
415     case USB_DT_STRING:
416         usb_handle_string_descriptor(pkt);
417         printf("GET STRING DESCRIPTOR\r\n");
418         break;
419
420     default:
421         printf("Unhandled GET_DESCRIPTOR type 0x%x\r\n", descriptor_type);
422     }
423 } else {
424     printf("Other IN request (0x%x)\r\n", pkt->bRequest);
425 }
426 }
427 }

```

12.7.4.2.4. Replying to a Setup Packet on EP0 IN

The host first requests the device descriptor. The following code handles that setup request:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c Lines 266 - 273

```

266 void usb_handle_device_descriptor(volatile struct usb_setup_packet *pkt) {
267     const struct usb_device_descriptor *d = dev_config.device_descriptor;
268     // EP0 in
269     struct usb_endpoint_configuration *ep = usb_get_endpoint_configuration(EP0_IN_ADDR);
270     // Always respond with pid 1
271     ep->next_pid = 1;
272     usb_start_transfer(ep, (uint8_t *) d, MIN(sizeof(struct usb_device_descriptor), pkt-
273         >wLength));
273 }

```

The `usb_start_transfer` function copies data to be sent into the appropriate hardware buffer and configures the buffer control register. Once the buffer control register has been written to, the device controller responds to the host with the data. Before this point, the device replies with a `NAK`:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/usb/device/dev_lowlevel/dev_lowlevel.c Lines 238 - 260

```

238 void usb_start_transfer(struct usb_endpoint_configuration *ep, uint8_t *buf, uint16_t len) {
239     // We are asserting that the length is <= 64 bytes for simplicity of the example.
240     // For multi packet transfers see the tinyusb port.
241     assert(len <= 64);
242
243     printf("Start transfer of len %d on ep addr 0x%x\n", len, ep->descriptor-
244         >bEndpointAddress);
245
246     // Prepare buffer control register value
247     uint32_t val = len | USB_BUF_CTRL_AVAIL;
248
249     if (ep_is_tx(ep)) {
250         // Need to copy the data from the user buffer to the usb memory
251         memcpy((void *) ep->data_buffer, (void *) buf, len);
252         // Mark as full
253         val |= USB_BUF_CTRL_FULL;
254     }
255
256     // Set pid and flip for next transfer
257     val |= ep->next_pid ? USB_BUF_CTRL_DATA1_PID : USB_BUF_CTRL_DATA0_PID;
258     ep->next_pid ^= 1u;
259 }

```

```

259     *ep->buffer_control = val;
260 }

```

12.7.5. List of Registers

The USB registers start at a base address of [0x50110000](#) (defined as `USBCTRL_REGS_BASE` in SDK).

Table 1148. List of USB registers

Offset	Name	Info
0x000	<code>ADDR_ENDP</code>	Device address and endpoint control
0x004	<code>ADDR_ENDP1</code>	Interrupt endpoint 1. Only valid for HOST mode.
0x008	<code>ADDR_ENDP2</code>	Interrupt endpoint 2. Only valid for HOST mode.
0x00c	<code>ADDR_ENDP3</code>	Interrupt endpoint 3. Only valid for HOST mode.
0x010	<code>ADDR_ENDP4</code>	Interrupt endpoint 4. Only valid for HOST mode.
0x014	<code>ADDR_ENDP5</code>	Interrupt endpoint 5. Only valid for HOST mode.
0x018	<code>ADDR_ENDP6</code>	Interrupt endpoint 6. Only valid for HOST mode.
0x01c	<code>ADDR_ENDP7</code>	Interrupt endpoint 7. Only valid for HOST mode.
0x020	<code>ADDR_ENDP8</code>	Interrupt endpoint 8. Only valid for HOST mode.
0x024	<code>ADDR_ENDP9</code>	Interrupt endpoint 9. Only valid for HOST mode.
0x028	<code>ADDR_ENDP10</code>	Interrupt endpoint 10. Only valid for HOST mode.
0x02c	<code>ADDR_ENDP11</code>	Interrupt endpoint 11. Only valid for HOST mode.
0x030	<code>ADDR_ENDP12</code>	Interrupt endpoint 12. Only valid for HOST mode.
0x034	<code>ADDR_ENDP13</code>	Interrupt endpoint 13. Only valid for HOST mode.
0x038	<code>ADDR_ENDP14</code>	Interrupt endpoint 14. Only valid for HOST mode.
0x03c	<code>ADDR_ENDP15</code>	Interrupt endpoint 15. Only valid for HOST mode.
0x040	<code>MAIN_CTRL</code>	Main control register
0x044	<code>SOF_WR</code>	Set the SOF (Start of Frame) frame number in the host controller. The SOF packet is sent every 1ms and the host will increment the frame number by 1 each time.
0x048	<code>SOF_RD</code>	Read the last SOF (Start of Frame) frame number seen. In device mode the last SOF received from the host. In host mode the last SOF sent by the host.
0x04c	<code>SIE_CTRL</code>	SIE control register
0x050	<code>SIE_STATUS</code>	SIE status register
0x054	<code>INT_EP_CTRL</code>	interrupt endpoint control register
0x058	<code>BUFF_STATUS</code>	Buffer status register. A bit set here indicates that a buffer has completed on the endpoint (if the buffer interrupt is enabled). It is possible for 2 buffers to be completed, so clearing the buffer status bit may instantly re set it on the next clock cycle.
0x05c	<code>BUFF_CPU_SHOULD_HANDLE</code>	Which of the double buffers should be handled. Only valid if using an interrupt per buffer (i.e. not per 2 buffers). Not valid for host interrupt endpoint polling because they are only single buffered.

Offset	Name	Info
0x060	EP_ABORT	Device only: Can be set to ignore the buffer control register for this endpoint in case you would like to revoke a buffer. A NAK will be sent for every access to the endpoint until this bit is cleared. A corresponding bit in EP_ABORT_DONE is set when it is safe to modify the buffer control register.
0x064	EP_ABORT_DONE	Device only: Used in conjunction with EP_ABORT. Set once an endpoint is idle so the programmer knows it is safe to modify the buffer control register.
0x068	EP_STALL_ARM	Device: this bit must be set in conjunction with the STALL bit in the buffer control register to send a STALL on EP0. The device controller clears these bits when a SETUP packet is received because the USB spec requires that a STALL condition is cleared when a SETUP packet is received.
0x06c	NAK_POLL	Used by the host controller. Sets the wait time in microseconds before trying again if the device replies with a NAK.
0x070	EP_STATUS_STALL_NAK	Device: bits are set when the IRQ_ON_NAK or IRQ_ON_STALL bits are set. For EP0 this comes from SIE_CTRL. For all other endpoints it comes from the endpoint control register.
0x074	USB_MUXING	Where to connect the USB controller. Should be to_phy by default.
0x078	USB_PWR	Overrides for the power signals in the event that the VBUS signals are not hooked up to GPIO. Set the value of the override and then the override enable to switch over to the override value.
0x07c	USBPHY_DIRECT	This register allows for direct control of the USB phy. Use in conjunction with usbphy_direct_override register to enable each override bit.
0x080	USBPHY_DIRECT_OVERRIDE	Override enable for each control in usbphy_direct
0x084	USBPHY_TRIM	Used to adjust trim values of USB phy pull down resistors.
0x088	LINESTATE_TUNING	Used for debug only.
0x08c	INTR	Raw Interrupts
0x090	INTE	Interrupt Enable
0x094	INTF	Interrupt Force
0x098	INTS	Interrupt status after masking & forcing
0x100	SOF_TIMESTAMP_RAW	Device only. Raw value of free-running PHY clock counter @48MHz. Used to calculate time between SOF events.
0x104	SOF_TIMESTAMP_LAST	Device only. Value of free-running PHY clock counter @48MHz when last SOF event occurred.
0x108	SM_STATE	
0x10c	EP_TX_ERROR	TX error count for each endpoint. Write to each field to reset the counter to 0.
0x110	EP_RX_ERROR	RX error count for each endpoint. Write to each field to reset the counter to 0.

Offset	Name	Info
0x114	DEV_SM_WATCHDOG	Watchdog that forces the device state machine to idle and raises an interrupt if the device stays in a state that isn't idle for the configured limit. The counter is reset on every state transition. Set limit while enable is low and then set the enable.

USB: ADDR_ENDP Register

Offset: 0x000

Description

Device address and endpoint control

Table 1149.
ADDR_ENDP Register

Bits	Name	Description	Type	Reset
31:20	Reserved.	-	-	-
19:16	ENDPOINT	Device endpoint to send data to. Only valid for HOST mode.	RW	0x0
15:7	Reserved.	-	-	-
6:0	ADDRESS	In device mode, the address that the device should respond to. Set in response to a SET_ADDR setup packet from the host. In host mode set to the address of the device to communicate with.	RW	0x00

USB: ADDR_ENDP1, ADDR_ENDP2, ..., ADDR_ENDP14, ADDR_ENDP15 Registers

Offsets: 0x004, 0x008, ..., 0x038, 0x03c

Description

Interrupt endpoint N. Only valid for HOST mode.

Table 1150.
ADDR_ENDP1,
ADDR_ENDP2, ...,
ADDR_ENDP14,
ADDR_ENDP15
Registers

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26	INTEP_PREAMBLE	Interrupt EP requires preamble (is a low speed device on a full speed hub)	RW	0x0
25	INTEP_DIR	Direction of the interrupt endpoint. In=0, Out=1	RW	0x0
24:20	Reserved.	-	-	-
19:16	ENDPOINT	Endpoint number of the interrupt endpoint	RW	0x0
15:7	Reserved.	-	-	-
6:0	ADDRESS	Device address	RW	0x00

USB: MAIN_CTRL Register

Offset: 0x040

Description

Main control register

Table 1151.
MAIN_CTRL Register

Bits	Name	Description	Type	Reset
31	SIM_TIMING	Reduced timings for simulation	RW	0x0

Bits	Name	Description	Type	Reset
30:3	Reserved.	-	-	-
2	PHY_ISO	Isolates USB phy after controller power-up Remove isolation once software has configured the controller Not isolated = 0, Isolated = 1	RW	0x1
1	HOST_NDEVICE	Device mode = 0, Host mode = 1	RW	0x0
0	CONTROLLER_EN	Enable controller	RW	0x0

USB: SOF_WR Register

Offset: 0x044

Description

Set the SOF (Start of Frame) frame number in the host controller. The SOF packet is sent every 1ms and the host will increment the frame number by 1 each time.

Table 1152. SOF_WR Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10:0	COUNT		WF	0x000

USB: SOF_RD Register

Offset: 0x048

Description

Read the last SOF (Start of Frame) frame number seen. In device mode the last SOF received from the host. In host mode the last SOF sent by the host.

Table 1153. SOF_RD Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10:0	COUNT		RO	0x000

USB: SIE_CTRL Register

Offset: 0x04c

Description

SIE control register

Table 1154. SIE_CTRL Register

Bits	Name	Description	Type	Reset
31	EP0_INT_STALL	Device: Set bit in EP_STATUS_STALL_NAK when EP0 sends a STALL	RW	0x0
30	EP0_DOUBLE_BUF	Device: EP0 single buffered = 0, double buffered = 1	RW	0x0
29	EP0_INT_1BUF	Device: Set bit in BUFF_STATUS for every buffer completed on EP0	RW	0x0
28	EP0_INT_2BUF	Device: Set bit in BUFF_STATUS for every 2 buffers completed on EP0	RW	0x0
27	EP0_INT_NAK	Device: Set bit in EP_STATUS_STALL_NAK when EP0 sends a NAK	RW	0x0

Bits	Name	Description	Type	Reset
26	DIRECT_EN	Direct bus drive enable	RW	0x0
25	DIRECT_DP	Direct control of DP	RW	0x0
24	DIRECT_DM	Direct control of DM	RW	0x0
23:20	Reserved.	-	-	-
19	EP0_STOP_ON_SHORT_PACKET	Device: Stop EP0 on a short packet.	RW	0x0
18	TRANSCEIVER_PD	Power down bus transceiver	RW	0x0
17	RPU_OPT	Device: Pull-up strength (0=1K2, 1=2k3)	RW	0x0
16	PULLUP_EN	Device: Enable pull up resistor	RW	0x0
15	PULLDOWN_EN	Host: Enable pull down resistors	RW	0x1
14	Reserved.	-	-	-
13	RESET_BUS	Host: Reset bus	SC	0x0
12	RESUME	Device: Remote wakeup. Device can initiate its own resume after suspend.	SC	0x0
11	VBUS_EN	Host: Enable VBUS	RW	0x0
10	KEEP_ALIVE_EN	Host: Enable keep alive packet (for low speed bus)	RW	0x0
9	SOF_EN	Host: Enable SOF generation (for full speed bus)	RW	0x0
8	SOF_SYNC	Host: Delay packet(s) until after SOF	RW	0x0
7	Reserved.	-	-	-
6	PREAMBLE_EN	Host: Preamble enable for LS device on FS hub	RW	0x0
5	Reserved.	-	-	-
4	STOP_TRANS	Host: Stop transaction	SC	0x0
3	RECEIVE_DATA	Host: Receive transaction (IN to host)	RW	0x0
2	SEND_DATA	Host: Send transaction (OUT from host)	RW	0x0
1	SEND_SETUP	Host: Send Setup packet	RW	0x0
0	START_TRANS	Host: Start transaction	SC	0x0

USB: SIE_STATUS Register

Offset: 0x050

Description

SIE status register

Table 1155.
SIE_STATUS Register

Bits	Name	Description	Type	Reset
31	DATA_SEQ_ERROR	<p>Data Sequence Error.</p> <p>The device can raise a sequence error in the following conditions:</p> <ul style="list-style-type: none"> * A SETUP packet is received followed by a DATA1 packet (data phase should always be DATA0) * An OUT packet is received from the host but doesn't match the data pid in the buffer control register read from DPRAM <p>The host can raise a data sequence error in the following conditions:</p> <ul style="list-style-type: none"> * An IN packet from the device has the wrong data PID 	WC	0x0
30	ACK_REC	ACK received. Raised by both host and device.	WC	0x0
29	STALL_REC	Host: STALL received	WC	0x0
28	NAK_REC	Host: NAK received	WC	0x0
27	RX_TIMEOUT	RX timeout is raised by both the host and device if an ACK is not received in the maximum time specified by the USB spec.	WC	0x0
26	RX_OVERFLOW	RX overflow is raised by the Serial RX engine if the incoming data is too fast.	WC	0x0
25	BIT_STUFF_ERROR	Bit Stuff Error. Raised by the Serial RX engine.	WC	0x0
24	CRC_ERROR	CRC Error. Raised by the Serial RX engine.	WC	0x0
23	ENDPOINT_ERROR	An endpoint has encountered an error. Read the ep_rx_error and ep_tx_error registers to find out which endpoint had an error.	WC	0x0
22:20	Reserved.	-	-	-
19	BUS_RESET	Device: bus reset received	WC	0x0
18	TRANS_COMPLETE	<p>Transaction complete.</p> <p>Raised by device if:</p> <ul style="list-style-type: none"> * An IN or OUT packet is sent with the LAST_BUFF bit set in the buffer control register <p>Raised by host if:</p> <ul style="list-style-type: none"> * A setup packet is sent when no data in or data out transaction follows * An IN packet is received and the LAST_BUFF bit is set in the buffer control register * An IN packet is received with zero length * An OUT packet is sent and the LAST_BUFF bit is set 	WC	0x0
17	SETUP_REC	Device: Setup packet received	WC	0x0
16	CONNECTED	Device: connected	RO	0x0
15:13	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
12	RX_SHORT_PACKET	Device or Host has received a short packet. This is when the data received is less than configured in the buffer control register. Device: If using double buffered mode on device the buffer select will not be toggled after writing status back to the buffer control register. This is to prevent any further transactions on that endpoint until the user has reset the buffer control registers. Host: the current transfer will be stopped early.	WC	0x0
11	RESUME	Host: Device has initiated a remote resume. Device: host has initiated a resume.	WC	0x0
10	VBUS_OVER_CURRENT	VBUS over current detected	RO	0x0
9:8	SPEED	Host: device speed. Disconnected = 00, LS = 01, FS = 10	RO	0x0
7:5	Reserved.	-	-	-
4	SUSPENDED	Bus in suspended state. Valid for device and host. Host and device will go into suspend if neither Keep Alive / SOF frames are enabled.	RO	0x0
3:2	LINE_STATE	USB bus line state	RO	0x0
1	Reserved.	-	-	-
0	VBUS_DETECTED	Device: VBUS Detected	RO	0x0

USB: INT_EP_CTRL Register

Offset: 0x054

Description

interrupt endpoint control register

Table 1156.
INT_EP_CTRL Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:1	INT_EP_ACTIVE	Host: Enable interrupt endpoint 1 → 15	RW	0x0000
0	Reserved.	-	-	-

USB: BUFF_STATUS Register

Offset: 0x058

Description

Buffer status register. A bit set here indicates that a buffer has completed on the endpoint (if the buffer interrupt is enabled). It is possible for 2 buffers to be completed, so clearing the buffer status bit may instantly re set it on the next clock cycle.

Table 1157.
BUFF_STATUS Register

Bits	Name	Description	Type	Reset
31	EP15_OUT		WC	0x0
30	EP15_IN		WC	0x0
29	EP14_OUT		WC	0x0
28	EP14_IN		WC	0x0

Bits	Name	Description	Type	Reset
27	EP13_OUT		WC	0x0
26	EP13_IN		WC	0x0
25	EP12_OUT		WC	0x0
24	EP12_IN		WC	0x0
23	EP11_OUT		WC	0x0
22	EP11_IN		WC	0x0
21	EP10_OUT		WC	0x0
20	EP10_IN		WC	0x0
19	EP9_OUT		WC	0x0
18	EP9_IN		WC	0x0
17	EP8_OUT		WC	0x0
16	EP8_IN		WC	0x0
15	EP7_OUT		WC	0x0
14	EP7_IN		WC	0x0
13	EP6_OUT		WC	0x0
12	EP6_IN		WC	0x0
11	EP5_OUT		WC	0x0
10	EP5_IN		WC	0x0
9	EP4_OUT		WC	0x0
8	EP4_IN		WC	0x0
7	EP3_OUT		WC	0x0
6	EP3_IN		WC	0x0
5	EP2_OUT		WC	0x0
4	EP2_IN		WC	0x0
3	EP1_OUT		WC	0x0
2	EP1_IN		WC	0x0
1	EP0_OUT		WC	0x0
0	EP0_IN		WC	0x0

USB: BUFF_CPU_SHOULD_HANDLE Register

Offset: 0x05c

Description

Which of the double buffers should be handled. Only valid if using an interrupt per buffer (i.e. not per 2 buffers). Not valid for host interrupt endpoint polling because they are only single buffered.

Table 1158.
BUFF_CPU_SHOULD_HANDLE Register

Bits	Name	Description	Type	Reset
31	EP15_OUT		RO	0x0
30	EP15_IN		RO	0x0

Bits	Name	Description	Type	Reset
29	EP14_OUT		RO	0x0
28	EP14_IN		RO	0x0
27	EP13_OUT		RO	0x0
26	EP13_IN		RO	0x0
25	EP12_OUT		RO	0x0
24	EP12_IN		RO	0x0
23	EP11_OUT		RO	0x0
22	EP11_IN		RO	0x0
21	EP10_OUT		RO	0x0
20	EP10_IN		RO	0x0
19	EP9_OUT		RO	0x0
18	EP9_IN		RO	0x0
17	EP8_OUT		RO	0x0
16	EP8_IN		RO	0x0
15	EP7_OUT		RO	0x0
14	EP7_IN		RO	0x0
13	EP6_OUT		RO	0x0
12	EP6_IN		RO	0x0
11	EP5_OUT		RO	0x0
10	EP5_IN		RO	0x0
9	EP4_OUT		RO	0x0
8	EP4_IN		RO	0x0
7	EP3_OUT		RO	0x0
6	EP3_IN		RO	0x0
5	EP2_OUT		RO	0x0
4	EP2_IN		RO	0x0
3	EP1_OUT		RO	0x0
2	EP1_IN		RO	0x0
1	EP0_OUT		RO	0x0
0	EP0_IN		RO	0x0

USB: EP_ABORT Register

Offset: 0x060

Description

Device only: Can be set to ignore the buffer control register for this endpoint in case you would like to revoke a buffer. A NAK will be sent for every access to the endpoint until this bit is cleared. A corresponding bit in EP_ABORT_DONE is set when it is safe to modify the buffer control register.

Table 1159.
EP_ABORT Register

Bits	Name	Description	Type	Reset
31	EP15_OUT		RW	0x0
30	EP15_IN		RW	0x0
29	EP14_OUT		RW	0x0
28	EP14_IN		RW	0x0
27	EP13_OUT		RW	0x0
26	EP13_IN		RW	0x0
25	EP12_OUT		RW	0x0
24	EP12_IN		RW	0x0
23	EP11_OUT		RW	0x0
22	EP11_IN		RW	0x0
21	EP10_OUT		RW	0x0
20	EP10_IN		RW	0x0
19	EP9_OUT		RW	0x0
18	EP9_IN		RW	0x0
17	EP8_OUT		RW	0x0
16	EP8_IN		RW	0x0
15	EP7_OUT		RW	0x0
14	EP7_IN		RW	0x0
13	EP6_OUT		RW	0x0
12	EP6_IN		RW	0x0
11	EP5_OUT		RW	0x0
10	EP5_IN		RW	0x0
9	EP4_OUT		RW	0x0
8	EP4_IN		RW	0x0
7	EP3_OUT		RW	0x0
6	EP3_IN		RW	0x0
5	EP2_OUT		RW	0x0
4	EP2_IN		RW	0x0
3	EP1_OUT		RW	0x0
2	EP1_IN		RW	0x0
1	EP0_OUT		RW	0x0
0	EP0_IN		RW	0x0

USB: EP_ABORT_DONE Register

Offset: 0x064

Description

Device only: Used in conjunction with EP_ABORT. Set once an endpoint is idle so the programmer knows it is safe to modify the buffer control register.

Table 1160.
EP_ABORT_DONE
Register

Bits	Name	Description	Type	Reset
31	EP15_OUT		WC	0x0
30	EP15_IN		WC	0x0
29	EP14_OUT		WC	0x0
28	EP14_IN		WC	0x0
27	EP13_OUT		WC	0x0
26	EP13_IN		WC	0x0
25	EP12_OUT		WC	0x0
24	EP12_IN		WC	0x0
23	EP11_OUT		WC	0x0
22	EP11_IN		WC	0x0
21	EP10_OUT		WC	0x0
20	EP10_IN		WC	0x0
19	EP9_OUT		WC	0x0
18	EP9_IN		WC	0x0
17	EP8_OUT		WC	0x0
16	EP8_IN		WC	0x0
15	EP7_OUT		WC	0x0
14	EP7_IN		WC	0x0
13	EP6_OUT		WC	0x0
12	EP6_IN		WC	0x0
11	EP5_OUT		WC	0x0
10	EP5_IN		WC	0x0
9	EP4_OUT		WC	0x0
8	EP4_IN		WC	0x0
7	EP3_OUT		WC	0x0
6	EP3_IN		WC	0x0
5	EP2_OUT		WC	0x0
4	EP2_IN		WC	0x0
3	EP1_OUT		WC	0x0
2	EP1_IN		WC	0x0
1	EP0_OUT		WC	0x0
0	EP0_IN		WC	0x0

USB: EP_STALL_ARM Register

Offset: 0x068**Description**

Device: this bit must be set in conjunction with the [STALL](#) bit in the buffer control register to send a STALL on EP0. The device controller clears these bits when a SETUP packet is received because the USB spec requires that a STALL condition is cleared when a SETUP packet is received.

Table 1161.
EP_STALL_ARM Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	EP0_OUT		RW	0x0
0	EP0_IN		RW	0x0

USB: NAK_POLL Register**Offset:** 0x06c**Description**

Used by the host controller. Sets the wait time in microseconds before trying again if the device replies with a NAK.

Table 1162.
NAK_POLL Register

Bits	Name	Description	Type	Reset
31:28	RETRY_COUNT_HI	Bits 9:6 of nak_retry count	RO	0x0
27	EPX_STOPPED_O_N_NAK	EPX polling has stopped because a nak was received	WC	0x0
26	STOP_EPX_ON_NAK	Stop polling epx when a nak is received	RW	0x0
25:16	DELAY_FS	NAK polling interval for a full speed device	RW	0x010
15:10	RETRY_COUNT_L0	Bits 5:0 of nak_retry_count	RO	0x00
9:0	DELAY_LS	NAK polling interval for a low speed device	RW	0x010

USB: EP_STATUS_STALL_NAK Register**Offset:** 0x070**Description**

Device: bits are set when the [IRQ_ON_NAK](#) or [IRQ_ON_STALL](#) bits are set. For EP0 this comes from [SIE_CTRL](#). For all other endpoints it comes from the endpoint control register.

Table 1163.
EP_STATUS_STALL_NAK Register

Bits	Name	Description	Type	Reset
31	EP15_OUT		WC	0x0
30	EP15_IN		WC	0x0
29	EP14_OUT		WC	0x0
28	EP14_IN		WC	0x0
27	EP13_OUT		WC	0x0
26	EP13_IN		WC	0x0
25	EP12_OUT		WC	0x0
24	EP12_IN		WC	0x0
23	EP11_OUT		WC	0x0

Bits	Name	Description	Type	Reset
22	EP11_IN		WC	0x0
21	EP10_OUT		WC	0x0
20	EP10_IN		WC	0x0
19	EP9_OUT		WC	0x0
18	EP9_IN		WC	0x0
17	EP8_OUT		WC	0x0
16	EP8_IN		WC	0x0
15	EP7_OUT		WC	0x0
14	EP7_IN		WC	0x0
13	EP6_OUT		WC	0x0
12	EP6_IN		WC	0x0
11	EP5_OUT		WC	0x0
10	EP5_IN		WC	0x0
9	EP4_OUT		WC	0x0
8	EP4_IN		WC	0x0
7	EP3_OUT		WC	0x0
6	EP3_IN		WC	0x0
5	EP2_OUT		WC	0x0
4	EP2_IN		WC	0x0
3	EP1_OUT		WC	0x0
2	EP1_IN		WC	0x0
1	EP0_OUT		WC	0x0
0	EP0_IN		WC	0x0

USB: USB_MUXING Register

Offset: 0x074

Description

Where to connect the USB controller. Should be to_phy by default.

Table 1164.
USB_MUXING Register

Bits	Name	Description	Type	Reset
31	SWAP_DPDM	Swap the USB PHY DP and DM pins and all related controls and flip receive differential data. Can be used to switch USB DP/DP on the PCB. This is done at a low level so overrides all other controls.	RW	0x0
30:5	Reserved.	-	-	-
4	USBPHY_AS_GPIO	Use the usb DP and DM pins as GPIO pins instead of connecting them to the USB controller.	RW	0x0
3	SOFTCON		RW	0x0
2	TO_DIGITAL_PAD		RW	0x0

Bits	Name	Description	Type	Reset
1	TO_EXTPHY		RW	0x0
0	TO_PHY		RW	0x1

USB: USB_PWR Register

Offset: 0x078

Description

Overrides for the power signals in the event that the VBUS signals are not hooked up to GPIO. Set the value of the override and then the override enable to switch over to the override value.

Table 1165. USB_PWR Register

Bits	Name	Description	Type	Reset
31:6	Reserved.	-	-	-
5	OVERCURR_DETECT_EN		RW	0x0
4	OVERCURR_DETECT		RW	0x0
3	VBUS_DETECT_OVERRIDE_EN		RW	0x0
2	VBUS_DETECT		RW	0x0
1	VBUS_EN_OVERRIDE_EN		RW	0x0
0	VBUS_EN		RW	0x0

USB: USBPHY_DIRECT Register

Offset: 0x07c

Description

This register allows for direct control of the USB phy. Use in conjunction with usbphy_direct_override register to enable each override bit.

Table 1166. USBPHY_DIRECT Register

Bits	Name	Description	Type	Reset
31:26	Reserved.	-	-	-
25	RX_DM_OVERRIDE	Override rx_dm value into controller	RW	0x0
24	RX_DP_OVERRIDE	Override rx_dp value into controller	RW	0x0
23	RX_DD_OVERRIDE	Override rx_dd value into controller	RW	0x0
22	DM_OVV	DM over voltage	RO	0x0
21	DP_OVV	DP over voltage	RO	0x0
20	DM_OVCN	DM overcurrent	RO	0x0
19	DP_OVCN	DP overcurrent	RO	0x0
18	RX_DM	DPM pin state	RO	0x0
17	RX_DP	DPP pin state	RO	0x0
16	RX_DD	Differential RX	RO	0x0
15	TX_DIFFMODE	TX_DIFFMODE=0: Single ended mode TX_DIFFMODE=1: Differential drive mode (TX_DM, TX_DM_OE ignored)	RW	0x0

Bits	Name	Description	Type	Reset
14	TX_FSSLEW	TX_FSSLEW=0: Low speed slew rate TX_FSSLEW=1: Full speed slew rate	RW	0x0
13	TX_PD	TX power down override (if override enable is set). 1 = powered down.	RW	0x0
12	RX_PD	RX power down override (if override enable is set). 1 = powered down.	RW	0x0
11	TX_DM	Output data. TX_DIFFMODE=1, Ignored TX_DIFFMODE=0, Drives DPM only. TX_DM_OE=1 to enable drive. DPM=TX_DM	RW	0x0
10	TX_DP	Output data. If TX_DIFFMODE=1, Drives DPP/DPM diff pair. TX_DP_OE=1 to enable drive. DPP=TX_DP, DPM=~TX_DP If TX_DIFFMODE=0, Drives DPP only. TX_DP_OE=1 to enable drive. DPP=TX_DP	RW	0x0
9	TX_DM_OE	Output enable. If TX_DIFFMODE=1, Ignored. If TX_DIFFMODE=0, OE for DPM only. 0 - DPM in Hi-Z state; 1 - DPM driving	RW	0x0
8	TX_DP_OE	Output enable. If TX_DIFFMODE=1, OE for DPP/DPM diff pair. 0 - DPP/DPM in Hi-Z state; 1 - DPP/DPM driving If TX_DIFFMODE=0, OE for DPP only. 0 - DPP in Hi-Z state; 1 - DPP driving	RW	0x0
7	Reserved.	-	-	-
6	DM_PULLDN_EN	DM pull down enable	RW	0x0
5	DM_PULLUP_EN	DM pull up enable	RW	0x0
4	DM_PULLUP_HISEL	Enable the second DM pull up resistor. 0 - Pull = Rpu2; 1 - Pull = Rpu1 + Rpu2	RW	0x0
3	Reserved.	-	-	-
2	DP_PULLDN_EN	DP pull down enable	RW	0x0
1	DP_PULLUP_EN	DP pull up enable	RW	0x0
0	DP_PULLUP_HISEL	Enable the second DP pull up resistor. 0 - Pull = Rpu2; 1 - Pull = Rpu1 + Rpu2	RW	0x0

USB: USBPHY_DIRECT_OVERRIDE Register

Offset: 0x080

Description

Override enable for each control in usbphy_direct

Table 1167.
USBPHY_DIRECT_OVERRIDE Register

Bits	Name	Description	Type	Reset
31:19	Reserved.	-	-	-
18	RX_DM_OVERRIDE_EN		RW	0x0
17	RX_DP_OVERRIDE_EN		RW	0x0
16	RX_DD_OVERRIDE_EN		RW	0x0
15	TX_DIFFMODE_OVERRIDE_EN		RW	0x0

Bits	Name	Description	Type	Reset
14:13	Reserved.	-	-	-
12	DM_PULLUP_OVERRIDE_EN		RW	0x0
11	TX_FSSLEW_OVERRIDE_EN		RW	0x0
10	TX_PD_OVERRIDE_EN		RW	0x0
9	RX_PD_OVERRIDE_EN		RW	0x0
8	TX_DM_OVERRIDE_EN		RW	0x0
7	TX_DP_OVERRIDE_EN		RW	0x0
6	TX_DM_OE_OVERRIDE_EN		RW	0x0
5	TX_DP_OE_OVERRIDE_EN		RW	0x0
4	DM_PULLDN_EN_OVERRIDE_EN		RW	0x0
3	DP_PULLDN_EN_OVERRIDE_EN		RW	0x0
2	DP_PULLUP_EN_OVERRIDE_EN		RW	0x0
1	DM_PULLUP_HISEL_OVERRIDE_EN		RW	0x0
0	DP_PULLUP_HISEL_OVERRIDE_EN		RW	0x0

USB: USBPHY_TRIM Register

Offset: 0x084

Description

Used to adjust trim values of USB phy pull down resistors.

Table 1168.
USBPHY_TRIM
Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12:8	DM_PULLDN_TRIM	Value to drive to USB PHY DM pulldown resistor trim control Experimental data suggests that the reset value will work, but this register allows adjustment if required	RW	0x1f
7:5	Reserved.	-	-	-
4:0	DP_PULLDN_TRIM	Value to drive to USB PHY DP pulldown resistor trim control Experimental data suggests that the reset value will work, but this register allows adjustment if required	RW	0x1f

USB: LINESTATE_TUNING Register

Offset: 0x088

Description

Used for debug only.

Table 1169.
LINESTATE_TUNING
Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:8	SPARE_FIX		RW	0x0

Bits	Name	Description	Type	Reset
7	DEV_LS_WAKE_FI X	Device - exit suspend on any non-idle signalling, not qualified with a 1ms timer	RW	0x1
6	DEV_RX_ERR QUI ESCE	Device - suppress repeated errors until the device FSM is next in the process of decoding an inbound packet.	RW	0x1
5	SIE_RX_CHATTER _SEO_FIX	RX - when recovering from line chatter or bitstuff errors, treat SEO as the end of chatter as well as 8 consecutive idle bits.	RW	0x1
4	SIE_RX_BITSTUFF _FIX	RX - when a bitstuff error is signalled by rx_dasm, unconditionally terminate RX decode to avoid a hang during certain packet phases.	RW	0x1
3	DEV_BUFF_CONT ROL_DOUBLE_RE AD_FIX	Device - the controller FSM performs two reads of the buffer status memory address to avoid sampling metastable data. An enabled buffer is only used if both reads match.	RW	0x1
2	MULTI_HUB_FIX	Host - increase inter-packet and turnaround timeouts to accommodate worst-case hub delays.	RW	0x0
1	LINESTATE_DELA Y	Device/Host - add an extra 1-bit debounce of linestate sampling.	RW	0x0
0	RCV_DELAY	Device - register the received data to account for hub bit dribble before EOP. Only affects certain hubs.	RW	0x0

USB: INTR Register

Offset: 0x08c

Description

Raw Interrupts

Table 1170. INTR Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23	EPX_STOPPED_O N_NAK	Source: NAK_POLL.EPX_STOPPED_ON_NAK	RO	0x0
22	DEV_SM_WATCH DOG_FIRED	Source: DEV_SM_WATCHDOG.FIRED	RO	0x0
21	ENDPOINT_ERRO R	Source: SIE_STATUS.ENDPOINT_ERROR	RO	0x0
20	RX_SHORT_PACK ET	Source: SIE_STATUS.RX_SHORT_PACKET	RO	0x0
19	EP_STALL_NAK	Raised when any bit in EP_STATUS_STALL_NAK is set. Clear by clearing all bits in EP_STATUS_STALL_NAK.	RO	0x0
18	ABORT_DONE	Raised when any bit in ABORT_DONE is set. Clear by clearing all bits in ABORT_DONE.	RO	0x0
17	DEV_SOF	Set every time the device receives a SOF (Start of Frame) packet. Cleared by reading SOF_RD	RO	0x0
16	SETUP_REQ	Device. Source: SIE_STATUS.SETUP_REC	RO	0x0

Bits	Name	Description	Type	Reset
15	DEV_RESUME_FROM_HOST	Set when the device receives a resume from the host. Cleared by writing to SIE_STATUS.RESUME	RO	0x0
14	DEV_SUSPEND	Set when the device suspend state changes. Cleared by writing to SIE_STATUS.SUSPENDED	RO	0x0
13	DEV_CONN_DIS	Set when the device connection state changes. Cleared by writing to SIE_STATUS.CONNECTED	RO	0x0
12	BUS_RESET	Source: SIE_STATUS.BUS_RESET	RO	0x0
11	VBUS_DETECT	Source: SIE_STATUS.VBUS_DETECTED	RO	0x0
10	STALL	Source: SIE_STATUSSTALL_REC	RO	0x0
9	ERROR_CRC	Source: SIE_STATUS.CRC_ERROR	RO	0x0
8	ERROR_BIT_STUFF	Source: SIE_STATUS.BIT_STUFF_ERROR	RO	0x0
7	ERROR_RX_OVERFLOW	Source: SIE_STATUS.RX_OVERFLOW	RO	0x0
6	ERROR_RX_TIMEOUT	Source: SIE_STATUS.RX_TIMEOUT	RO	0x0
5	ERROR_DATA_SEQ	Source: SIE_STATUS.DATA_SEQ_ERROR	RO	0x0
4	BUFF_STATUS	Raised when any bit in BUFF_STATUS is set. Clear by clearing all bits in BUFF_STATUS.	RO	0x0
3	TRANS_COMPLETE	Raised every time SIE_STATUS.TRANS_COMPLETE is set. Clear by writing to this bit.	RO	0x0
2	HOST_SOF	Host: raised every time the host sends a SOF (Start of Frame). Cleared by reading SOF_RD	RO	0x0
1	HOST_RESUME	Host: raised when a device wakes up the host. Cleared by writing to SIE_STATUS.RESUME	RO	0x0
0	HOST_CONN_DIS	Host: raised when a device is connected or disconnected (i.e. when SIE_STATUS.SPEED changes). Cleared by writing to SIE_STATUS.SPEED	RO	0x0

USB: INTE Register

Offset: 0x090

Description

Interrupt Enable

Table 1171. INTE Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23	EPX_STOPPED_ON_NAK	Source: NAK_POLL.EPX_STOPPED_ON_NAK	RW	0x0
22	DEV_SM_WATCHDOG_FIRED	Source: DEV_SM_WATCHDOG.FIRED	RW	0x0
21	ENDPOINT_ERROR	Source: SIE_STATUS.ENDPOINT_ERROR	RW	0x0

Bits	Name	Description	Type	Reset
20	RX_SHORT_PACKET	Source: SIE_STATUS.RX_SHORT_PACKET	RW	0x0
19	EP_STALL_NAK	Raised when any bit in EP_STATUS_STALL_NAK is set. Clear by clearing all bits in EP_STATUS_STALL_NAK.	RW	0x0
18	ABORT_DONE	Raised when any bit in ABORT_DONE is set. Clear by clearing all bits in ABORT_DONE.	RW	0x0
17	DEV_SOF	Set every time the device receives a SOF (Start of Frame) packet. Cleared by reading SOF_RD	RW	0x0
16	SETUP_REQ	Device. Source: SIE_STATUS.SETUP_REC	RW	0x0
15	DEV_RESUME_FRAME_HOST	Set when the device receives a resume from the host. Cleared by writing to SIE_STATUS.RESUME	RW	0x0
14	DEV_SUSPEND	Set when the device suspend state changes. Cleared by writing to SIE_STATUS.SUSPENDED	RW	0x0
13	DEV_CONN_DIS	Set when the device connection state changes. Cleared by writing to SIE_STATUS.CONNECTED	RW	0x0
12	BUS_RESET	Source: SIE_STATUS.BUS_RESET	RW	0x0
11	VBUS_DETECT	Source: SIE_STATUS.VBUS_DETECTED	RW	0x0
10	STALL	Source: SIE_STATUS.STALL_REC	RW	0x0
9	ERROR_CRC	Source: SIE_STATUS.CRC_ERROR	RW	0x0
8	ERROR_BIT_STUFF	Source: SIE_STATUS.BIT_STUFF_ERROR	RW	0x0
7	ERROR_RX_OVERFLOW	Source: SIE_STATUS.RX_OVERFLOW	RW	0x0
6	ERROR_RX_TIMEOUT	Source: SIE_STATUS.RX_TIMEOUT	RW	0x0
5	ERROR_DATA_SEQ	Source: SIE_STATUS.DATA_SEQ_ERROR	RW	0x0
4	BUFF_STATUS	Raised when any bit in BUFF_STATUS is set. Clear by clearing all bits in BUFF_STATUS.	RW	0x0
3	TRANS_COMPLETE	Raised every time SIE_STATUS.TRANS_COMPLETE is set. Clear by writing to this bit.	RW	0x0
2	HOST_SOF	Host: raised every time the host sends a SOF (Start of Frame). Cleared by reading SOF_RD	RW	0x0
1	HOST_RESUME	Host: raised when a device wakes up the host. Cleared by writing to SIE_STATUS.RESUME	RW	0x0
0	HOST_CONN_DIS	Host: raised when a device is connected or disconnected (i.e. when SIE_STATUS.SPEED changes). Cleared by writing to SIE_STATUS.SPEED	RW	0x0

USB: INTF Register

Offset: 0x094

Description

Interrupt Force

Table 1172. INTF Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23	EPX_STOPPED_O N_NAK	Source: NAK_POLL.EPX_STOPPED_ON_NAK	RW	0x0
22	DEV_SM_WATCH DOG_FIRED	Source: DEV_SM_WATCHDOG.FIRED	RW	0x0
21	ENDPOINT_ERRO R	Source: SIE_STATUS.ENDPOINT_ERROR	RW	0x0
20	RX_SHORT_PAC ET	Source: SIE_STATUS.RX_SHORT_PACKET	RW	0x0
19	EP_STALL_NAK	Raised when any bit in EP_STATUS_STALL_NAK is set. Clear by clearing all bits in EP_STATUS_STALL_NAK.	RW	0x0
18	ABORT_DONE	Raised when any bit in ABORT_DONE is set. Clear by clearing all bits in ABORT_DONE.	RW	0x0
17	DEV_SOF	Set every time the device receives a SOF (Start of Frame) packet. Cleared by reading SOF_RD	RW	0x0
16	SETUP_REQ	Device. Source: SIE_STATUS.SETUP_REC	RW	0x0
15	DEV_RESUME_FR OM_HOST	Set when the device receives a resume from the host. Cleared by writing to SIE_STATUS.RESUME	RW	0x0
14	DEV_SUSPEND	Set when the device suspend state changes. Cleared by writing to SIE_STATUS.SUSPENDED	RW	0x0
13	DEV_CONN_DIS	Set when the device connection state changes. Cleared by writing to SIE_STATUS.CONNECTED	RW	0x0
12	BUS_RESET	Source: SIE_STATUS.BUS_RESET	RW	0x0
11	VBUS_DETECT	Source: SIE_STATUS.VBUS_DETECTED	RW	0x0
10	STALL	Source: SIE_STATUS.STALL_REC	RW	0x0
9	ERROR_CRC	Source: SIE_STATUS.CRC_ERROR	RW	0x0
8	ERROR_BIT_STUF F	Source: SIE_STATUS.BIT_STUFF_ERROR	RW	0x0
7	ERROR_RX_OVER FLOW	Source: SIE_STATUS.RX_OVERFLOW	RW	0x0
6	ERROR_RX_TIME OUT	Source: SIE_STATUS.RX_TIMEOUT	RW	0x0
5	ERROR_DATA_SE Q	Source: SIE_STATUS.DATA_SEQ_ERROR	RW	0x0
4	BUFF_STATUS	Raised when any bit in BUFF_STATUS is set. Clear by clearing all bits in BUFF_STATUS.	RW	0x0
3	TRANS_COMPLET E	Raised every time SIE_STATUS.TRANS_COMPLETE is set. Clear by writing to this bit.	RW	0x0
2	HOST_SOF	Host: raised every time the host sends a SOF (Start of Frame). Cleared by reading SOF_RD	RW	0x0

Bits	Name	Description	Type	Reset
1	HOST_RESUME	Host: raised when a device wakes up the host. Cleared by writing to SIE_STATUS.RESUME	RW	0x0
0	HOST_CONN_DIS	Host: raised when a device is connected or disconnected (i.e. when SIE_STATUS.SPEED changes). Cleared by writing to SIE_STATUS.SPEED	RW	0x0

USB: INTS Register

Offset: 0x098

Description

Interrupt status after masking & forcing

Table 1173. INTS Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23	EPX_STOPPED_ON_NAK	Source: NAK_POLL.EPX_STOPPED_ON_NAK	RO	0x0
22	DEV_SM_WATCHDOG_FIRED	Source: DEV_SM_WATCHDOG.FIRED	RO	0x0
21	ENDPOINT_ERROR	Source: SIE_STATUS.ENDPOINT_ERROR	RO	0x0
20	RX_SHORT_PACKET	Source: SIE_STATUS.RX_SHORT_PACKET	RO	0x0
19	EP_STALL_NAK	Raised when any bit in EP_STATUS_STALL_NAK is set. Clear by clearing all bits in EP_STATUS_STALL_NAK.	RO	0x0
18	ABORT_DONE	Raised when any bit in ABORT_DONE is set. Clear by clearing all bits in ABORT_DONE.	RO	0x0
17	DEV_SOF	Set every time the device receives a SOF (Start of Frame) packet. Cleared by reading SOF_RD	RO	0x0
16	SETUP_REQ	Device. Source: SIE_STATUS.SETUP_REC	RO	0x0
15	DEV_RESUME_FROM_HOST	Set when the device receives a resume from the host. Cleared by writing to SIE_STATUS.RESUME	RO	0x0
14	DEV_SUSPEND	Set when the device suspend state changes. Cleared by writing to SIE_STATUS.SUSPENDED	RO	0x0
13	DEV_CONN_DIS	Set when the device connection state changes. Cleared by writing to SIE_STATUS.CONNECTED	RO	0x0
12	BUS_RESET	Source: SIE_STATUS.BUS_RESET	RO	0x0
11	VBUS_DETECT	Source: SIE_STATUS.VBUS_DETECTED	RO	0x0
10	STALL	Source: SIE_STATUSSTALL_REC	RO	0x0
9	ERROR_CRC	Source: SIE_STATUS.CRC_ERROR	RO	0x0
8	ERROR_BIT_STUFF	Source: SIE_STATUS.BIT_STUFF_ERROR	RO	0x0
7	ERROR_RX_OVERFLOW	Source: SIE_STATUS.RX_OVERFLOW	RO	0x0

Bits	Name	Description	Type	Reset
6	ERROR_RX_TIME_OUT	Source: SIE_STATUS.RX_TIMEOUT	RO	0x0
5	ERROR_DATA_SEQ_Q	Source: SIE_STATUS.DATA_SEQ_ERROR	RO	0x0
4	BUFF_STATUS	Raised when any bit in BUFF_STATUS is set. Clear by clearing all bits in BUFF_STATUS.	RO	0x0
3	TRANS_COMPLETE	Raised every time SIE_STATUS.TRANS_COMPLETE is set. Clear by writing to this bit.	RO	0x0
2	HOST_SOF	Host: raised every time the host sends a SOF (Start of Frame). Cleared by reading SOF_RD	RO	0x0
1	HOST_RESUME	Host: raised when a device wakes up the host. Cleared by writing to SIE_STATUS.RESUME	RO	0x0
0	HOST_CONN_DIS	Host: raised when a device is connected or disconnected (i.e. when SIE_STATUS.SPEED changes). Cleared by writing to SIE_STATUS.SPEED	RO	0x0

USB: SOF_TIMESTAMP_RAW Register

Offset: 0x100

Table 1174.
SOF_TIMESTAMP_RAW Register

Bits	Description	Type	Reset
31:21	Reserved.	-	-
20:0	Device only. Raw value of free-running PHY clock counter @48MHz. Used to calculate time between SOF events.	RO	0x000000

USB: SOF_TIMESTAMP_LAST Register

Offset: 0x104

Table 1175.
SOF_TIMESTAMP_LAST Register

Bits	Description	Type	Reset
31:21	Reserved.	-	-
20:0	Device only. Value of free-running PHY clock counter @48MHz when last SOF event occurred.	RO	0x000000

USB: SM_STATE Register

Offset: 0x108

Table 1176.
SM_STATE Register

Bits	Name	Description	Type	Reset
31:12	Reserved.	-	-	-
11:8	RX_DASM		RO	0x0
7:5	BC_STATE		RO	0x0
4:0	STATE		RO	0x00

USB: EP_TX_ERROR Register

Offset: 0x10c

Description

TX error count for each endpoint. Write to each field to reset the counter to 0.

Table 1177.
EP_TX_ERROR
Register

Bits	Name	Description	Type	Reset
31:30	EP15		WC	0x0
29:28	EP14		WC	0x0
27:26	EP13		WC	0x0
25:24	EP12		WC	0x0
23:22	EP11		WC	0x0
21:20	EP10		WC	0x0
19:18	EP9		WC	0x0
17:16	EP8		WC	0x0
15:14	EP7		WC	0x0
13:12	EP6		WC	0x0
11:10	EP5		WC	0x0
9:8	EP4		WC	0x0
7:6	EP3		WC	0x0
5:4	EP2		WC	0x0
3:2	EP1		WC	0x0
1:0	EP0		WC	0x0

USB: EP_RX_ERROR Register

Offset: 0x110

Description

RX error count for each endpoint. Write to each field to reset the counter to 0.

Table 1178.
EP_RX_ERROR
Register

Bits	Name	Description	Type	Reset
31	EP15_SEQ		WC	0x0
30	EP15_TRANSACTION		WC	0x0
29	EP14_SEQ		WC	0x0
28	EP14_TRANSACTION		WC	0x0
27	EP13_SEQ		WC	0x0
26	EP13_TRANSACTION		WC	0x0
25	EP12_SEQ		WC	0x0
24	EP12_TRANSACTION		WC	0x0
23	EP11_SEQ		WC	0x0
22	EP11_TRANSACTION		WC	0x0
21	EP10_SEQ		WC	0x0
20	EP10_TRANSACTION		WC	0x0

Bits	Name	Description	Type	Reset
19	EP9_SEQ		WC	0x0
18	EP9_TRANSACTION		WC	0x0
17	EP8_SEQ		WC	0x0
16	EP8_TRANSACTION		WC	0x0
15	EP7_SEQ		WC	0x0
14	EP7_TRANSACTION		WC	0x0
13	EP6_SEQ		WC	0x0
12	EP6_TRANSACTION		WC	0x0
11	EP5_SEQ		WC	0x0
10	EP5_TRANSACTION		WC	0x0
9	EP4_SEQ		WC	0x0
8	EP4_TRANSACTION		WC	0x0
7	EP3_SEQ		WC	0x0
6	EP3_TRANSACTION		WC	0x0
5	EP2_SEQ		WC	0x0
4	EP2_TRANSACTION		WC	0x0
3	EP1_SEQ		WC	0x0
2	EP1_TRANSACTION		WC	0x0
1	EP0_SEQ		WC	0x0
0	EP0_TRANSACTION		WC	0x0

USB: DEV_SM_WATCHDOG Register

Offset: 0x114

Description

Watchdog that forces the device state machine to idle and raises an interrupt if the device stays in a state that isn't idle for the configured limit. The counter is reset on every state transition.

Set limit while enable is low and then set the enable.

Table 1179.
DEV_SM_WATCHDOG
Register

Bits	Name	Description	Type	Reset
31:21	Reserved.	-	-	-
20	FIRE		WC	0x0
19	RESET	Set to 1 to forcibly reset the device state machine on watchdog expiry	RW	0x0
18	ENABLE		RW	0x0
17:0	LIMIT		RW	0x00000

12.8. System Timers

12.8.1. Overview

The system timer peripheral on RP2350 provides a global microsecond timebase for the system, and generates interrupts based on this timebase. It supports the following features:

- A single 64-bit counter, incrementing once per microsecond
 - Read from a pair of latching registers for race-free reads over a 32-bit bus
- Four alarms that match on the lower 32 bits of the counter and generate IRQ on match

The timer uses a one microsecond reference generated in the watchdog (see [Section 8.5](#)), and derived from the reference clock ([Figure 30](#)), which itself is usually connected directly to the crystal oscillator ([Section 8.2](#)).

The 64-bit counter effectively cannot overflow (thousands of years at 1MHz), so the system timer is completely monotonic in practice.

12.8.1.1. Changes from RP2040

- RP2350 now has two timer instances: `TIMER0` and `TIMER1`
- On RP2350, the tick source for each timer comes from the ticks block (see [Section 8.5](#))
- RP2350 added two new registers: `LOCKED` is used to disable write access to the timer, and `SOURCE` allows the timer to count system clock cycles rather than a 1 μ s tick

12.8.1.2. Other Timer Resources on RP2350

The system timer provides a global timebase for software. RP2350 has a number of other programmable counter resources which can provide regular interrupts, or trigger DMA transfers.

- The PWM ([Section 12.5](#)) contains 12× 16-bit programmable counters. These counters:
 - run at up to system speed
 - can generate interrupts to either of two system IRQ lines
 - can be continuously reprogrammed via the DMA
 - can trigger DMA transfers to other peripherals
- 12× PIO state machines ([Chapter 11](#)) can count 32-bit values at system speed, and generate interrupts.
- The DMA ([Section 12.6](#)) has four internal pacing timers which trigger transfers at regular intervals.
- Each Cortex-M33 core ([Section 3.7](#)) has a standard 24-bit SysTick timer, counting either the microsecond tick ([Section 8.5](#)) or the system clock.
- SIO has a standard 64-bit RISC-V platform timer ([Section 3.1.8](#)). Arm and RISC-V software can use this timer.
- The Power Manager ([Chapter 6](#)) incorporates a 64-bit timer (AON-Timer) which nominally counts milliseconds (see [Section 12.10](#)). This is the only timer that runs when the chip is in its lowest power state, with all switchable power domains powered down. It is used to schedule power-ups.

12.8.2. Counter

The timer has a 64-bit counter, but RP2350 only has a 32-bit data bus. This means that the `TIME` value is accessed through a pair of registers. These are:

- `TIMEHW` and `TIMELW` to write the time
- `TIMEHR` and `TIMELR` to read the time

To use these pairs, access the lower register, `L`, followed by the higher register, `H`. In the read case, reading the `L` register latches the value in the `H` register to provide an accurate time. To read the raw time without any latching, use `TIMERAWH` and `TIMERAWL`.

⚠ CAUTION

Don't write to the `TIMEHW` and `TIMELW` registers to force a new time value if other software may be using the timer. The SDK uses the time value for timeouts, elapsed time, and more, and expects the value to increase monotonically.

12.8.3. Alarms

The timer has 4 alarms, and outputs a separate interrupt for each alarm. The alarms match on the lower 32 bits of the 64-bit counter, which means they can be fired at a maximum of 2^{32} microseconds into the future. This is equivalent to:

- $2^{32} \div 10^6: \sim 4295$ seconds
- $4295 \div 60: \sim 72$ minutes

ℹ NOTE

This timer supports alarm intervals on the order of one microsecond to one hour. For a longer alarm, see [Section 12.10](#).

To enable an alarm:

1. Enable the interrupt at the timer with a write to the appropriate alarm bit in `INTE` (e.g. `(1 << 0)` for `ALARM0`).
2. Enable the appropriate timer interrupt at the processor (see [Section 3.2](#)).
3. Write the time you would like the interrupt to fire to `ALARM0` (i.e. the current value in `TIMERAWL` plus your desired alarm time in microseconds). Writing the time to the `ALARM` register sets the `ARMED` bit as a side effect.

Once the alarm has fired, the `ARMED` bit clears to `0`. To clear the latched interrupt, write a `1` to the appropriate bit in `INTR`.

12.8.4. Programmer's Model

ℹ NOTE

The timer's tick (see [Section 8.5](#)) must be running for the timer to start counting. The SDK starts this tick as part of the platform initialisation code.

12.8.4.1. Reading the time

ℹ NOTE

Time here refers to the number of microseconds since the timer was started, not a clock. For a clock, see [Section 12.10](#).

To read the 64-bit time, read `TIMELR` followed by `TIMEHR`. Reading `TIMELR` latches (stops) the value in `TIMEHR` until `TIMEHR` is read. Because RP2350 has 2 cores, it is unsafe to do this if the second core executes code that can also access the timer, or if the timer is read concurrently in an IRQ handler and in thread mode. If one core reads `TIMELR` followed by another core reading `TIMELR`, the value in `TIMEHR` isn't necessarily accurate. The example below shows the simplest form

of getting the 64-bit time:

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/timer/timer_lowlevel/timer_lowlevel.c Lines 15 - 23

```
15 // Simplest form of getting 64 bit time from the timer.
16 // It isn't safe when called from 2 cores because of the latching
17 // so isn't implemented this way in the sdk
18 static uint64_t get_time(void) {
19     // Reading low latches the high value
20     uint32_t lo = timer_hw->timelr;
21     uint32_t hi = timer_hw->timehr;
22     return ((uint64_t) hi << 32u) | lo;
23 }
```

The SDK provides a `time_us_64` function that uses a more thorough method to get the 64-bit time, which makes use of the `TIMERAWH` and `TIMERAWL` registers. The `RAW` registers don't latch, making `time_us_64` safe to call from multiple cores at once.

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_timer/timer.c Lines 57 - 73

```
57 uint64_t timer_time_us_64(timer_hw_t *timer) {
58     // Need to make sure that the upper 32 bits of the timer
59     // don't change, so read that first
60     uint32_t hi = timer->timerawh;
61     uint32_t lo;
62     do {
63         // Read the lower 32 bits
64         lo = timer->timerawl;
65         // Now read the upper 32 bits again and
66         // check that it hasn't incremented. If it has loop around
67         // and read the lower 32 bits again to get an accurate value
68         uint32_t next_hi = timer->timerawh;
69         if (hi == next_hi) break;
70         hi = next_hi;
71     } while (true);
72     return ((uint64_t) hi << 32u) | lo;
73 }
```

12.8.4.2. Set an alarm

The standalone timer example, `timer_lowlevel`, demonstrates how to set an alarm at a hardware level without the additional abstraction over the timer provided by SDK. To use these abstractions, see [Section 12.8.4.4](#).

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/timer/timer_lowlevel/timer_lowlevel.c Lines 27 - 73

```
27 // Use alarm 0
28 #define ALARM_NUM 0
29 #define ALARM_IRQ hardware_alarm_get_irq_num(timer_hw, ALARM_NUM)
30
31 // Alarm interrupt handler
32 static volatile bool alarm_fired;
33
34 static void alarm_irq(void) {
35     // Clear the alarm irq
36     hw_clear_bits(&timer_hw->intr, 1u << ALARM_NUM);
37
38     // Assume alarm 0 has fired
39     printf("Alarm IRQ fired\n");
```

```

40     alarm_fired = true;
41 }
42
43 static void alarm_in_us(uint32_t delay_us) {
44     // Enable the interrupt for our alarm (the timer outputs 4 alarm irqs)
45     hw_set_bits(&timer_hw->inte, 1u << ALARM_NUM);
46     // Set irq handler for alarm irq
47     irq_set_exclusive_handler(ALARM_IRQ, alarm_irq);
48     // Enable the alarm irq
49     irq_set_enabled(ALARM_IRQ, true);
50     // Enable interrupt in block and at processor
51
52     // Alarm is only 32 bits so if trying to delay more
53     // than that need to be careful and keep track of the upper
54     // bits
55     uint64_t target = timer_hw->timerawl + delay_us;
56
57     // Write the lower 32 bits of the target time to the alarm which
58     // will arm it
59     timer_hw->alarm[ALARM_NUM] = (uint32_t) target;
60 }
61
62 int main() {
63     stdio_init_all();
64     printf("Timer lowlevel!\n");
65
66     // Set alarm every 2 seconds
67     while (1) {
68         alarm_fired = false;
69         alarm_in_us(1000000 * 2);
70         // Wait for alarm to fire
71         while (!alarm_fired);
72     }
73 }

```

12.8.4.3. Busy wait

If you don't want to use an alarm to wait for a period of time, use a while loop instead. The SDK provides various `busy_wait_` functions to do this:

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_timer/timer.c Lines 77 - 122

```

77 void timer_busy_wait_us_32(timer_hw_t *timer, uint32_t delay_us) {
78     if (0 <= (int32_t)delay_us) {
79         // we only allow 31 bits, otherwise we could have a race in the loop below with
80         // values very close to 2^32
81         uint32_t start = timer->timerawl;
82         while (timer->timerawl - start < delay_us) {
83             tight_loop_contents();
84         }
85     } else {
86         busy_wait_us(delay_us);
87     }
88 }
89
90 void timer_busy_wait_us(timer_hw_t *timer, uint64_t delay_us) {
91     uint64_t base = timer_time_us_64(timer);
92     uint64_t target = base + delay_us;
93     if (target < base) {
94         target = (uint64_t)-1;

```

```

95      }
96      absolute_time_t t;
97      update_us_since_boot(&t, target);
98      timer_busy_wait_until(timer, t);
99  }
100
101 void timer_busy_wait_ms(timer_hw_t *timer, uint32_t delay_ms)
102 {
103     if (delay_ms <= 0x7fffffff / 1000) {
104         timer_busy_wait_us_32(timer, delay_ms * 1000);
105     } else {
106         timer_busy_wait_us(timer, delay_ms * 1000ull);
107     }
108 }
109
110 void timer_busy_wait_until(timer_hw_t *timer, absolute_time_t t) {
111     uint64_t target = to_us_since_boot(t);
112     uint32_t hi_target = (uint32_t)(target >> 32u);
113     uint32_t hi = timer->timerawh;
114     while (hi < hi_target) {
115         hi = timer->timerawh;
116         tight_loop_contents();
117     }
118     while (hi == hi_target && timer->timerawl < (uint32_t) target) {
119         hi = timer->timerawh;
120         tight_loop_contents();
121     }
122 }

```

12.8.4.4. Complete example using SDK

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/timer/hello_timer/hello_timer.c Lines 11 - 57

```

11 volatile bool timer_fired = false;
12
13 int64_t alarm_callback(alarm_id_t id, void *user_data) {
14     printf("Timer %d fired!\n", (int) id);
15     timer_fired = true;
16     // Can return a value here in us to fire in the future
17     return 0;
18 }
19
20 bool repeating_timer_callback(struct repeating_timer *t) {
21     printf("Repeat at %lld\n", time_us_64());
22     return true;
23 }
24
25 int main() {
26     stdio_init_all();
27     printf("Hello Timer!\n");
28
29     // Call alarm_callback in 2 seconds
30     add_alarm_in_ms(2000, alarm_callback, NULL, false);
31
32     // Wait for alarm callback to set timer_fired
33     while (!timer_fired) {
34         tight_loop_contents();
35     }
36
37     // Create a repeating timer that calls repeating_timer_callback.

```

```

38     // If the delay is > 0 then this is the delay between the previous callback ending and the
39     // next starting.
40     // If the delay is negative (see below) then the next call to the callback will be exactly
41     // 500ms after the
42     // start of the call to the last callback
43     struct repeating_timer timer;
44     add_repeating_timer_ms(500, repeating_timer_callback, NULL, &timer);
45     sleep_ms(3000);
46     bool cancelled = cancel_repeating_timer(&timer);
47     printf("cancelled... %d\n", cancelled);
48     sleep_ms(2000);
49
50     // Negative delay so means we will call repeating_timer_callback, and call it again
51     // 500ms later regardless of how long the callback took to execute
52     add_repeating_timer_ms(-500, repeating_timer_callback, NULL, &timer);
53     sleep_ms(3000);
54     cancelled = cancel_repeating_timer(&timer);
55     printf("cancelled... %d\n", cancelled);
56     sleep_ms(2000);
57     printf("Done\n");
58     return 0;
59 }

```

12.8.5. List of Registers

The **TIMER0** and **TIMER1** registers start at base addresses of **0x400b0000** and **0x400b8000** respectively (defined as **TIMER0_BASE** and **TIMER1_BASE** in SDK).

Table 1180. List of
TIMER registers

Offset	Name	Info
0x00	TIMEHW	Write to bits 63:32 of time always write timelw before timehw
0x04	TIMELW	Write to bits 31:0 of time writes do not get copied to time until timehw is written
0x08	TIMEHR	Read from bits 63:32 of time always read timelr before timehr
0x0c	TIMELR	Read from bits 31:0 of time
0x10	ALARM0	Arm alarm 0, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM0 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.
0x14	ALARM1	Arm alarm 1, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM1 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.
0x18	ALARM2	Arm alarm 2, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM2 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.
0x1c	ALARM3	Arm alarm 3, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM3 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.

Offset	Name	Info
0x20	ARMED	Indicates the armed/disarmed status of each alarm. A write to the corresponding ALARMx register arms the alarm. Alarms automatically disarm upon firing, but writing ones here will disarm immediately without waiting to fire.
0x24	TIMERAWH	Raw read from bits 63:32 of time (no side effects)
0x28	TIMERAWL	Raw read from bits 31:0 of time (no side effects)
0x2c	DBGPAUSE	Set bits high to enable pause when the corresponding debug ports are active
0x30	PAUSE	Set high to pause the timer
0x34	LOCKED	Set locked bit to disable write access to timer Once set, cannot be cleared (without a reset)
0x38	SOURCE	Selects the source for the timer. Defaults to the normal tick configured in the ticks block (typically configured to 1 microsecond). Writing to 1 will ignore the tick and count clk_sys cycles instead.
0x3c	INTR	Raw Interrupts
0x40	INTE	Interrupt Enable
0x44	INTF	Interrupt Force
0x48	INTS	Interrupt status after masking & forcing

TIMER: TIMEHW Register

Offset: 0x00

Table 1181. TIMEHW Register

Bits	Description	Type	Reset
31:0	Write to bits 63:32 of time always write timelw before timehw	WF	0x00000000

TIMER: TIMELW Register

Offset: 0x04

Table 1182. TIMELW Register

Bits	Description	Type	Reset
31:0	Write to bits 31:0 of time writes do not get copied to time until timehw is written	WF	0x00000000

TIMER: TIMEHR Register

Offset: 0x08

Table 1183. TIMEHR Register

Bits	Description	Type	Reset
31:0	Read from bits 63:32 of time always read timelr before timehr	RO	0x00000000

TIMER: TIMELR Register

Offset: 0x0c

Table 1184. TIMELR Register

Bits	Description	Type	Reset
31:0	Read from bits 31:0 of time	RO	0x00000000

TIMER: ALARM0 Register

Offset: 0x10

Table 1185. ALARM0 Register

Bits	Description	Type	Reset
31:0	Arm alarm 0, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM0 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.	RW	0x00000000

TIMER: ALARM1 Register

Offset: 0x14

Table 1186. ALARM1 Register

Bits	Description	Type	Reset
31:0	Arm alarm 1, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM1 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.	RW	0x00000000

TIMER: ALARM2 Register

Offset: 0x18

Table 1187. ALARM2 Register

Bits	Description	Type	Reset
31:0	Arm alarm 2, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM2 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.	RW	0x00000000

TIMER: ALARM3 Register

Offset: 0x1c

Table 1188. ALARM3 Register

Bits	Description	Type	Reset
31:0	Arm alarm 3, and configure the time it will fire. Once armed, the alarm fires when TIMER_ALARM3 == TIMELR. The alarm will disarm itself once it fires, and can be disarmed early using the ARMED status register.	RW	0x00000000

TIMER: ARMED Register

Offset: 0x20

Table 1189. ARMED Register

Bits	Description	Type	Reset
31:4	Reserved.	-	-
3:0	Indicates the armed/disarmed status of each alarm. A write to the corresponding ALARMx register arms the alarm. Alarms automatically disarm upon firing, but writing ones here will disarm immediately without waiting to fire.	WC	0x0

TIMER: TIMERAWH Register

Offset: 0x24

Table 1190. TIMERAWH Register

Bits	Description	Type	Reset
31:0	Raw read from bits 63:32 of time (no side effects)	RO	0x00000000

TIMER: TIMERAWL Register

Offset: 0x28

Table 1191. TIMERAWL Register

Bits	Description	Type	Reset
31:0	Raw read from bits 31:0 of time (no side effects)	RO	0x00000000

TIMER: DBGPAUSE Register

Offset: 0x2c

Description

Set bits high to enable pause when the corresponding debug ports are active

Table 1192. DBGPAUSE Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	DBG1	Pause when processor 1 is in debug mode	RW	0x1
1	DBG0	Pause when processor 0 is in debug mode	RW	0x1
0	Reserved.	-	-	-

TIMER: PAUSE Register

Offset: 0x30

Table 1193. PAUSE Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Set high to pause the timer	RW	0x0

TIMER: LOCKED Register

Offset: 0x34

Table 1194. LOCKED Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Set locked bit to disable write access to timer Once set, cannot be cleared (without a reset)	RW	0x0

TIMER: SOURCE Register

Offset: 0x38

Description

Selects the source for the timer. Defaults to the normal tick configured in the ticks block (typically configured to 1 microsecond). Writing to 1 will ignore the tick and count clk_sys cycles instead.

Table 1195. SOURCE Register

Bits	Name	Description	Type	Reset
31:1	Reserved.	-	-	-
0	CLK_SYS	0x0 → TICK 0x1 → CLK_SYS	RW	0x0

TIMER: INTR Register

Offset: 0x3c

Description

Raw Interrupts

Table 1196. INTR Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	ALARM_3		WC	0x0
2	ALARM_2		WC	0x0
1	ALARM_1		WC	0x0
0	ALARM_0		WC	0x0

TIMER: INTE Register

Offset: 0x40

Description

Interrupt Enable

Table 1197. INTE Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	ALARM_3		RW	0x0
2	ALARM_2		RW	0x0
1	ALARM_1		RW	0x0
0	ALARM_0		RW	0x0

TIMER: INTF Register

Offset: 0x44

Description

Interrupt Force

Table 1198. INTF Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	ALARM_3		RW	0x0
2	ALARM_2		RW	0x0
1	ALARM_1		RW	0x0
0	ALARM_0		RW	0x0

TIMER: INTS Register

Offset: 0x48

Description

Interrupt status after masking & forcing

Table 1199. INTS Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	ALARM_3		RO	0x0
2	ALARM_2		RO	0x0
1	ALARM_1		RO	0x0
0	ALARM_0		RO	0x0

12.9. Watchdog

12.9.1. Overview

The watchdog is a countdown timer that can restart parts of the chip if it reaches zero. Use the watchdog to restart the processor if software gets stuck in an infinite loop. When the watchdog is enabled the programmer must periodically write to it to stop it from reaching zero.

The watchdog is reset by `rst_n_run`, which is deasserted when the RUN pin is high and the digital core supply (DVDD) is powered and stable. This allows the watchdog reset to feed into the power-on state machine (see [Section 7.4](#)) and reset controller (see [Chapter 7](#)), resetting their dependants if they are selected in the `WDSEL` register. A `WDSEL` register exists in the power manager (POWMAN [WATCHDOG](#)), the power-on state machine (PSM [WDSEL](#)), and the reset controller (RESETS [WDSEL](#)).

NOTE

The PSM is the device which sequences resets of critical logic components following a power-up of the switched core domain. It is distinct from POWMAN.

12.9.2. Changes from RP2040

On RP2040, the watchdog contained a tick generator used to generate a 1 μ s tick for the watchdog. This was also distributed to the system timer. On RP2350, the watchdog instead takes a tick input from the system-level ticks block. See [Section 8.5](#).

As on RP2040 the watchdog can trigger a PSM (Power-on State Machine) sequence to reset system components or selected subsystem components. On RP2350, the watchdog can also trigger a chip level reset. See [Section 7.3](#).

12.9.3. Watchdog Counter

The watchdog counter is loaded by the `LOAD` register. The current value can be seen in `CTRL.TIME`.

12.9.4. Control Watchdog Reset Levels

To control the level of reset triggered by a watchdog event, use the registers outside the watchdog register block:

- `POWMAN_WATCHDOG` allows the watchdog to trigger chip level resets
- `PSM_WDSEL` allows the watchdog to trigger system resets by running a full or partial PSM sequence (Power-on State Machine)
- `RESETS_WDSEL` allows the watchdog to trigger subsystem resets

These are described in the Resets section, see [Chapter 7](#).

12.9.5. Scratch Registers

The watchdog contains eight 32-bit scratch registers that can store information between soft resets of the chip. The scratch registers reset when:

- the watchdog is used to trigger a chip level reset
- a `rst_n_run` event occurs, triggered by toggling the RUN pin or cycling the digital core supply (DVDD)

The bootrom checks the watchdog scratch registers for a magic number on boot. You can use this to soft reset the chip into user-specified code. See [Section 5.3.2](#) for more information.

NOTE

Additional general-purpose scratch registers are available in POWMAN `SCRATCH0` through `SCRATCH7`. These registers also survive power cycling the switched core domain.

12.9.6. Programmer's Model

The SDK provides a `hardware_watchdog` driver to control the watchdog.

12.9.6.1. Enabling the watchdog

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_watchdog/watchdog.c Lines 41 - 73

```

41 // Helper function used by both watchdog_enable and watchdog_reboot
42 void _watchdog_enable(uint32_t delay_ms, bool pause_on_debug) {
43     valid_params_if(HARDWARE_WATCHDOG, delay_ms <= WATCHDOG_LOAD_BITS / (1000 *
44         WATCHDOG_XFACTOR));
45     hw_clear_bits(&watchdog_hw->ctrl, WATCHDOG_CTRL_ENABLE_BITS);
46     // Reset everything apart from ROSC and XOSC
47     hw_set_bits(&psm_hw->wdsel, PSM_WDSEL_BITS & ~(PSM_WDSEL_ROSC_BITS |
48         PSM_WDSEL_XOSC_BITS));

```

```

48
49     uint32_t dbg_bits = WATCHDOG_CTRL_PAUSE_DBG0_BITS |
50                     WATCHDOG_CTRL_PAUSE_DBG1_BITS |
51                     WATCHDOG_CTRL_PAUSE_JTAG_BITS;
52
53     if (pause_on_debug) {
54         hw_set_bits(&watchdog_hw->ctrl, dbg_bits);
55     } else {
56         hw_clear_bits(&watchdog_hw->ctrl, dbg_bits);
57     }
58
59     if (!delay_ms) {
60         hw_set_bits(&watchdog_hw->ctrl, WATCHDOG_CTRL_TRIGGER_BITS);
61     } else {
62         load_value = delay_ms * 1000;
63 #if PICO_RP2040
64         load_value *= 2;
65 #endif
66         if (load_value > WATCHDOG_LOAD_BITS)
67             load_value = WATCHDOG_LOAD_BITS;
68
69         watchdog_update();
70
71         hw_set_bits(&watchdog_hw->ctrl, WATCHDOG_CTRL_ENABLE_BITS);
72     }
73 }

```

12.9.6.2. Updating the watchdog counter

SDK: https://asic-git.pitowers.org/amethyst/pico-sdk/-/blob/master/src/rp2_common/hardware_watchdog/watchdog.c Lines 23 - 27

```

23 static uint32_t load_value;
24
25 void watchdog_update(void) {
26     watchdog_hw->load = load_value;
27 }

```

12.9.6.3. Usage

The Pico Examples repository provides a `hello_watchdog` example that uses the `hardware_watchdog` to demonstrate use of the watchdog.

Pico Examples: https://asic-git.pitowers.org/amethyst/pico-examples/-/blob/master/watchdog/hello_watchdog/hello_watchdog.c Lines 11 - 33

```

11 int main() {
12     stdio_init_all();
13
14     if (watchdog_caused_reboot()) {
15         printf("Rebooted by Watchdog!\n");
16         return 0;
17     } else {
18         printf("Clean boot\n");
19     }
20
21     // Enable the watchdog, requiring the watchdog to be updated every 100ms or the chip will
22     // reboot
23     // second arg is pause on debug which means the watchdog will pause when stepping through

```

```

code
23  watchdog_enable(100, 1);
24
25  for (uint i = 0; i < 5; i++) {
26      printf("Updating watchdog %d\n", i);
27      watchdog_update();
28  }
29
30  // Wait in an infinite loop and don't update the watchdog so it reboots us
31  printf("Waiting to be rebooted by watchdog\n");
32  while(1);
33 }

```

12.9.7. List of Registers

The watchdog registers start at a base address of **0x400d8000** (defined as **WATCHDOG_BASE** in SDK).

Table 1200. List of WATCHDOG registers

Offset	Name	Info
0x00	CTRL	Watchdog control The rst_wdsel register determines which subsystems are reset when the watchdog is triggered. The watchdog can be triggered in software.
0x04	LOAD	Load the watchdog timer. The maximum setting is 0xffffffff which corresponds to approximately 16 seconds.
0x08	REASON	Logs the reason for the last reset. Both bits are zero for the case of a hardware reset. Additionally, as of RP2350, a debugger warm reset of either core (SYSRESETREQ or hartreset) will also clear the watchdog reason register, so that software loaded under the debugger following a watchdog timeout will not continue to see the timeout condition.
0x0c	SCRATCH0	Scratch register. Information persists through soft reset of the chip.
0x10	SCRATCH1	Scratch register. Information persists through soft reset of the chip.
0x14	SCRATCH2	Scratch register. Information persists through soft reset of the chip.
0x18	SCRATCH3	Scratch register. Information persists through soft reset of the chip.
0x1c	SCRATCH4	Scratch register. Information persists through soft reset of the chip.
0x20	SCRATCH5	Scratch register. Information persists through soft reset of the chip.
0x24	SCRATCH6	Scratch register. Information persists through soft reset of the chip.
0x28	SCRATCH7	Scratch register. Information persists through soft reset of the chip.

WATCHDOG: CTRL Register

Offset: 0x00

Description

Watchdog control

The `rst_wsel` register determines which subsystems are reset when the watchdog is triggered.

The watchdog can be triggered in software.

Table 1201. CTRL Register

Bits	Name	Description	Type	Reset
31	TRIGGER	Trigger a watchdog reset	SC	0x0
30	ENABLE	When not enabled the watchdog timer is paused	RW	0x0
29:27	Reserved.	-	-	-
26	PAUSE_DBG1	Pause the watchdog timer when processor 1 is in debug mode	RW	0x1
25	PAUSE_DBG0	Pause the watchdog timer when processor 0 is in debug mode	RW	0x1
24	PAUSE_JTAG	Pause the watchdog timer when JTAG is accessing the bus fabric	RW	0x1
23:0	TIME	Indicates the time in usec before a watchdog reset will be triggered	RO	0x000000

WATCHDOG: LOAD Register

Offset: 0x04

Table 1202. LOAD Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Load the watchdog timer. The maximum setting is 0xffffffff which corresponds to approximately 16 seconds.	WF	0x000000

WATCHDOG: REASON Register

Offset: 0x08

Description

Logs the reason for the last reset. Both bits are zero for the case of a hardware reset.

Additionally, as of RP2350, a debugger warm reset of either core (SYSRESETREQ or hartreset) will also clear the watchdog reason register, so that software loaded under the debugger following a watchdog timeout will not continue to see the timeout condition.

Table 1203. REASON Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	FORCE		RO	0x0
0	TIMER		RO	0x0

WATCHDOG: SCRATCH0, SCRATCH1, ..., SCRATCH6, SCRATCH7 Registers

Offsets: 0x0c, 0x10, ..., 0x24, 0x28

Table 1204.
 SCRATCH0,
 SCRATCH1, ...,
 SCRATCH6,
 SCRATCH7 Registers

Bits	Description	Type	Reset
31:0	Scratch register. Information persists through soft reset of the chip.	RW	0x00000000

12.10. Always-On Timer

12.10.1. Overview

The always-on timer (AON-Timer) is the only timer that operates in all power modes. It can be used as a real-time counter or an interval timer and incorporates an alarm which can be used to trigger a power-up event or an interrupt. It incorporates a 64-bit counter intended to count 1ms ticks, but the tick generator can be configured to run faster or slower if required. Note that the AON-Timer tick generator is independent of all other tick generators on the chip.

The default tick source is the 32kHz on-chip low-power oscillator (LPOSC), see [Section 8.4](#). The LPOSC frequency is not precise and may vary with voltage and temperature. When the chip core is powered, the tick source can be switched to the on-chip crystal oscillator (XOSC) for greater precision. If greater precision is also required when the chip core is unpowered, then a 32kHz clock or a 1ms tick can be supplied from an external source. Alternatively, the AON-Timer can be synchronised to an external 1Hz source.

The AON-Timer is integrated with the power manager (POWMAN) and shares the POWMAN register block. Writes are limited to 16 bits because a key ([0x5afe](#)) is required in the top 16 bits to prevent erroneous writes from locking up the chip. Most AON-Timer registers can be enabled for write by Non-secure software, unlike other POWMAN registers. However, the registers used to select an external clock, select an external tick source, and enable power-up on alarm can only be written by Secure software.

12.10.2. Changes from RP2040

The RP2040 Real Time Clock (RTC) is not used in RP2350. Instead, RP2350 has a timer in the Always-On power domain which is used for scheduling power-up events and can also be used as a real-time counter. The AON-Timer works differently from the RP2040 RTC. It counts milliseconds to approximately 58 bits (a 48-bit seconds counter, and a separate milliseconds counter) and this value can be used to calculate the date and time in software.

12.10.3. Accessing the AON-Timer

To start and stop the AON-Timer, write to [TIMER.RUN](#).

To read the current 64-bit AON-Timer value, use the following 2×32 -bit read-only registers:

- [READ_TIME_UPPER](#)
- [READ_TIME_LOWER](#)

Because the AON-Timer can increment during a read, use the following procedure to protect against erroneous reads:

1. Read [READ_TIME_UPPER](#)
2. Read [READ_TIME_LOWER](#)
3. Read [READ_TIME_UPPER](#)
4. If the [READ_TIME_UPPER](#) value changes between steps 1 and 3, repeat the whole procedure

When used as a real time clock, the 64-bit time value is set using 4×16 -bit registers. These registers can only be written when the AON-Timer is stopped by writing a 0 to [TIMER.RUN](#):

- [SET_TIME_63TO48](#)

- [SET_TIME_47T032](#)
- [SET_TIME_31T016](#)
- [SET_TIME_15T00](#)

These registers cannot be used to read the time value.

When used as an interval timer, write a 1 to [TIMER.CLEAR](#) to clear the timer value. It is not necessary to stop the AON-Timer to do this. The [TIMER.CLEAR](#) register is self-clearing: it returns to 0 when the operation completes. This allows easy implementation of an alarm that wakes the chip or generates an interrupt at regular intervals.

12.10.4. Using the Alarm

To set the alarm time, use the following 4×16 -bit registers:

- [ALARM_TIME_63T048](#)
- [ALARM_TIME_47T032](#)
- [ALARM_TIME_31T016](#)
- [ALARM_TIME_15T00](#)

To avoid false alarms, disable the alarm before setting the alarm time.

To enable the alarm, use [TIMER.ALARM_ENAB](#).

When the alarm fires, the AON-Timer sets the alarm status flag [TIMER.ALARM](#).

To clear the alarm status flag, write a 1 to the alarm status flag.

To configure the alarm to trigger a power-up, set [TIMER.PWRUP_ON_ALARM](#). This feature is not available to Non-secure code.

The alarm can be configured to trigger an interrupt. The interrupt is handled in the standard way using the following register fields:

- [INTR.TIMER](#) - raw interrupt
- [INTE.TIMER](#) - interrupt enable
- [INTF.TIMER](#) - force interrupt
- [INTS.TIMER](#) - interrupt status

12.10.5. Selecting the AON-Timer Tick Source

The AON-Timer indicates the current configuration with read-only flags. [Table 1205](#) provides a list of sources supported by the 1kHz AON-Timer tick.

Table 1205. AON-Timer tick generators

Tick source	Read-only flag
LPOSC clock division	TIMER.USING_LPOSC
XOSC clock division	TIMER.USING_XOSC
external 1kHz tick	TIMER.USING_GPIO_1KHZ

NOTE

The LPOSC clock can be substituted by an external 32kHz clock.

12.10.5.1. Using LPOSC as the AON-Timer Tick Source

LPOSC is the default source and can be used in all power modes. It nominally runs at 32.768kHz and can only be tuned to 1% accuracy. The AON-Timer derives the 1ms tick from the LPOSC using a 6.16 bit fractional divider whose divisor is initialised to 32.768. The divisor can be modified to achieve greater accuracy. Because the LPOSC frequency varies with supply voltage and temperature, accuracy is limited unless supply voltage and temperature are stable. To modify the divisor, write to the following registers:

- [LPOSC_FREQ_KHZ_INT](#) (default value: 32)
- [LPOSC_FREQ_KHZ_FRAC](#) (default value: 0.768)

These registers should only be written when [TIMER.RUN](#) = 0 or [TIMER.USING_LPOSC](#) = 0.

If the tick source is not LPOSC, you can switch it back to LPOSC by writing a 1 to [TIMER.USE_LPOSC](#). It is not necessary to stop the AON-Timer to do this. The newly selected tick will be synchronised to the current tick, so the operation may take up to 1 tick cycle (1ms in normal operation). When the operation is complete, [TIMER.USE_LPOSC](#) will self-clear and [TIMER.USING_LPOSC](#) will be set. Due to sampling, a small error of up to 2 periods of the newly selected clock will be subtracted from the time. When switching to LPOSC at 32kHz, an error of up to 62 μ s will be subtracted.

12.10.5.2. Using an External Clock in Place of LPOSC

If LPOSC is not sufficiently accurate, an external 32.768kHz clock can be used. This will be multiplexed onto the internal low-power clock and will therefore drive all components that are driven by that clock, including the power sequencer components. The external clock can be used in all power modes. When an external clock is in use, you can stop the LPOSC (see [Section 8.4](#)).

To select an external 32kHz clock:

1. Configure the GPIO source as described in [Section 12.10.7](#).
2. Switch to the external LPOSC by setting [EXT_TIME_REF.DRIVE_LPCK](#). This register should only be written when [TIMER.RUN](#) = 0 and the power sequencer is inactive. You can only write to this register from Secure code.

The external 32kHz clock replaces the clock from LPOSC. Therefore the same registers are used for AON-Timer configuration (see [Section 12.10.5.1](#)):

- [TIMER.USE_LPOSC](#)
- [TIMER.USING_LPOSC](#)
- [LPOSC_FREQ_KHZ_INT](#)
- [LPOSC_FREQ_KHZ_FRAC](#)

12.10.5.3. Using the XOSC as the AON-Timer Tick Source

The XOSC clock is provided via the reference clock ([clk_ref](#)). The user must ensure the reference clock is being driven from the XOSC before selecting it as the source of the AON-Timer tick. This is the normal configuration following boot. To check, look for [CLK_REF_SELECTED](#) = `0x4`. The reference clock may be a divided version of the XOSC. The divisor defaults to 1 and can be read from [CLK_REF_DIV.INT](#). If the chip is operated with a faster XOSC, the clock sent to the AON-Timer must not exceed 29MHz.

The AON-Timer derives the 1ms tick from the XOSC using a 16.16 bit fractional divider whose divisor is initialised to 12000.0. This assumes a 12MHz crystal is used and the reference clock divisor is 1. If that is not the case, the divisor in the AON-Timer can be modified by writing to the following registers:

- [XOSC_FREQ_KHZ_INT](#) (default value: 12000)
- [XOSC_FREQ_KHZ_FRAC](#) (default value: 0)

These registers should only be written when [TIMER.RUN](#) = 0 or [TIMER.USING_XOSC](#) = 0.

To select the XOSC as the AON-Timer tick source, write a 1 to [TIMER.USE_XOSC](#). It is not necessary to stop the AON-Timer to do this. The newly selected tick will be synchronised to the current tick, so the operation may take up to 1 tick cycle (1ms in normal operation). When the operation is complete [TIMER.USE_XOSC](#) will self-clear and [TIMER.USING_XOSC](#) will be set. Due to sampling, a small error of up to 2 periods of the newly selected clock will be subtracted from the time. When switching to XOSC at 12MHz an error of up to 167ns will be subtracted.

When the chip core is powered down the XOSC will stop. If [TIMER.USING_XOSC](#) is set, the power-down sequencer automatically reverts to [TIMER.USING_LPOSC](#) before the XOSC stops.

12.10.5.4. Using an External 1ms Tick Source

To select an external 1ms tick source, configure the GPIO source as described in [Section 12.10.7](#). Then, write a 1 to [TIMER.USE_GPIO_1KHZ](#). It is not necessary to stop the AON-Timer to do this, however the newly selected tick will not be synchronised to the current tick, so the operation so the operation will advance the time by up to 1ms. If using an external 1ms tick it is recommended to set the time after selecting the source. When the operation is complete [TIMER.USE_GPIO_1KHZ](#) will self-clear and [TIMER.USING_GPIO_1KHZ](#) will be set.

The tick is triggered from the falling edge of the selected GPIO. For correct sampling, the GPIO pulse width and interval must both be greater than the period of LPOSC (>31us). This limits the maximum frequency of the external tick to 16kHz.

The external 1ms tick can be used in all power modes.

12.10.6. Synchronising the AON-Timer to an External 1Hz Clock

In applications that use GPS, a 1s tick may be available. This can be used to synchronise the AON-Timer and thus compensate for inaccuracy in the LPOSC frequency. It can be used with any tick source, but there is little to be gained if the selected source is already reasonably accurate.

If the LPOSC is fast, the ms counter pauses at a 1 second step until the 1s tick is received. If the LPOSC is slow, the 1s tick causes the ms counter to run very quickly until reaching the 1 second step. This ensures that all ms values are counted, ensuring that any alarm set to ms precision will fire. A more sophisticated synchronisation method can be implemented in software.

To use the hardware synchronisation feature, configure the GPIO source as described in [Section 12.10.7](#). Then, enable the feature by writing a 1 to [TIMER.USE_GPIO_1HZ](#). This can be set at any time, it is not necessary to stop the AON-Timer. When the operation is complete [TIMER.USE_GPIO_1HZ](#) will self-clear and [TIMER.USING_GPIO_1HZ](#) will be set.

The tick is triggered from the falling edge of the selected GPIO. For correct sampling, the GPIO pulse width and interval must be greater than the period of LPOSC (>31us).

The external 1s tick can be used in all power modes.

12.10.7. Using an external clock or tick from GPIO

The following features use a GPIO as a clock or a tick:

- external 32kHz clock source
- external 1kHz tick
- external 1Hz tick

Only 4 GPIOs are available for these features. You can only select one, because they share the same GPIO selection

logic. The set of 4 GPIOs differs between package types. The selection is controlled by a 2-bit register field.

QFN-80 uses the following GPIOs:

- [EXT_TIME_REF.SOURCE_SEL](#) = 0 → GPIO12
- [EXT_TIME_REF.SOURCE_SEL](#) = 1 → GPIO20
- [EXT_TIME_REF.SOURCE_SEL](#) = 2 → GPIO14
- [EXT_TIME_REF.SOURCE_SEL](#) = 3 → GPIO22

QFN-60 uses the following GPIOs:

- [EXT_TIME_REF.SOURCE_SEL](#) = 0 → GPIO21
- [EXT_TIME_REF.SOURCE_SEL](#) = 1 → GPIO34
- [EXT_TIME_REF.SOURCE_SEL](#) = 2 → GPIO23
- [EXT_TIME_REF.SOURCE_SEL](#) = 3 → GPIO36

12.10.8. Using a Tick Faster than 1ms

The tick rate can be increased by scaling the value written to the LPOSC and XOSC frequency registers. For example, if the frequency value is divided by 4 then the AON-Timer will tick 4 times per ms. The minimum value that can be written to the frequency registers is 2.0, therefore the maximum upscaling using this method with LPOSC is 16, giving a time resolution of 1/16th of 1 ms (= 62.5μs).

As described previously, the external tick is limited to 16kHz, so the maximum upscaling using this method is also 16. This gives a time resolution of 1/16th of 1 ms (62.5μs).

These limitations can be overcome either by using a faster external clock (see [Section 12.10.5.2](#)) or keeping the chip core powered so the AON-Timer is always running from the XOSC. If a faster external clock is used then the power sequencer timings will also need to be adjusted.

For example, suppose 1μsec timer precision is required. The user could supply an external 2-25MHz clock in place of the LPOSC and program both the LPOSC and XOSC frequency registers in MHz units rather than kHz. The maximum frequency of the external clock is 29MHz.

12.10.9. List of Registers

The AON-Timer shares a register address space with the power management subsystems in the always-on domain. The address space is referred to as [POWMAN](#) elsewhere in this document. A complete list of [POWMAN](#) registers is provided in [Section 6.6](#), but information on registers associated with the always-on timer is repeated here.

The [POWMAN](#) registers start at a base address of [0x40100000](#) (defined as [POWMAN_BASE](#) in SDK).

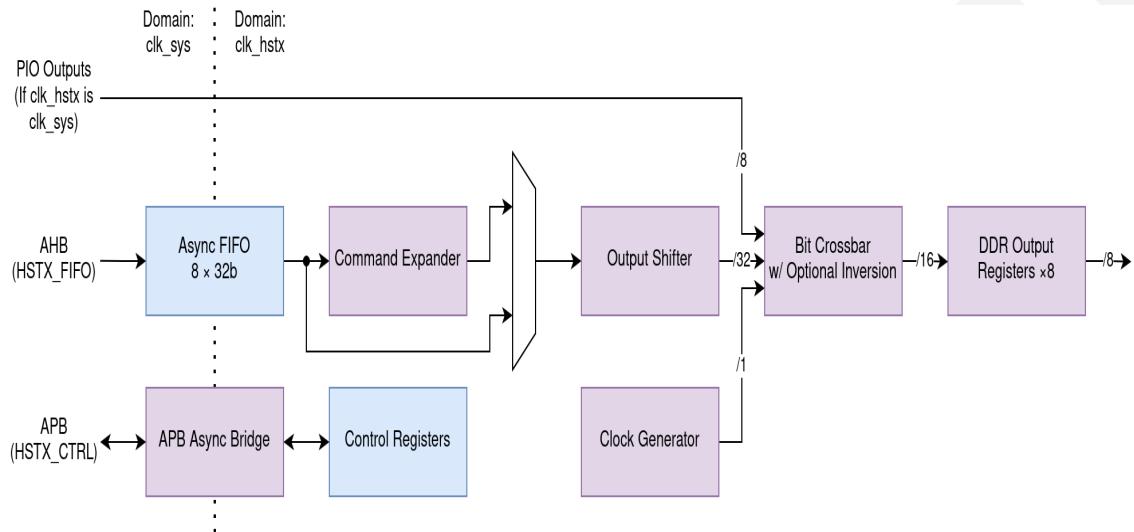
- [SET_TIME_63T048](#)
- [SET_TIME_47T032](#)
- [SET_TIME_31T016](#)
- [SET_TIME_15T00](#)
- [READ_TIME_UPPER](#)
- [READ_TIME_LOWER](#)
- [ALARM_TIME_63T048](#)
- [ALARM_TIME_47T032](#)
- [ALARM_TIME_31T016](#)

- [ALARM_TIME_15TO0](#)
- [TIMER](#)

12.11. HSTX

The high-speed serial transmit (HSTX) streams data from the system clock domain to up to 8 GPIOs at a rate independent of the system clock. On RP2350, GPIOs 12 through 19 are HSTX-capable. HSTX is output-only.

Figure 102. The HSTX is asynchronous from the rest of the system. A 32-bit-wide FIFO provides high-bandwidth access from the system DMA. The command expander performs simple manipulation of the datastream, and the output shift register portions the 32-bit data out over successive HSTX clock cycles, swizzled by the bit crossbar. The outputs are double-data-rate: up to two bits per pin per clock cycle.



HSTX drives data through GPIOs using DDR output registers to transfer up to two bits per clock cycle per pin. The HSTX balances all delays to GPIO outputs within 300 picoseconds, minimising common-mode components when using neighbouring GPIOs as a pseudo-differential driver. This also helps maintain destination setup and hold time when a clock is driven alongside the output data.

The maximum frequency for the HSTX clock is 150MHz, the same as the system clock. With DDR output operation, this is a maximum data rate of 300Mb/s per pin. There are no limits on the frequency ratio of the system and HSTX clocks, however each clock must be individually fast enough to maintain your required throughput. Very low system clock frequencies coupled with very high HSTX frequencies may encounter system DMA bandwidth limitations, since the DMA is capped at one HSTX FIFO write per system clock cycle.

12.11.1. Data FIFO

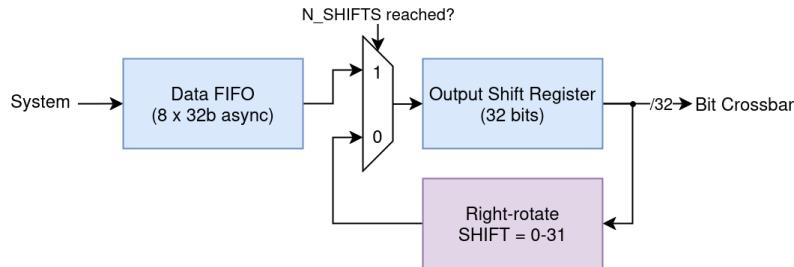
An 8-entry, 32-bit-wide FIFO buffers data between the system clock domain (`clk_sys`) and the HSTX clock domain (`clk_hstx`). This is accessed through the AHB **FASTPERI** arbiter, providing single-cycle write access from the DMA. The FIFO status is also available through this same bus interface, for faster polled processor IO; see [Section 12.11.8](#).

The FIFO is accessed through a bus interface separate from the control registers ([Section 12.11.7](#)), which take multiple cycles to access due to the asynchronous bus crossing. This design avoids incurring bus stalls on the system DMA or the **FASTPERI** arbiter when accessing the FIFO.

The HSTX side also pops 32 bits at a time from the FIFO. The word data stream from the FIFO is optionally manipulated by the command expander ([Section 12.11.5](#)) before being passed to the output shift register.

12.11.2. Output Shift Register

Figure 103. Every cycle, the output shift register either refills 32 bits from the FIFO or recirculates data through a right-rotate function. The rotate can be used to perform left or right shifts, and to repeat data.



The HSTX's internal data paths are 32 bits wide, but the output is narrower: no more than 16 bits can be output per HSTX cycle (8 GPIOs \times DDR). The output shift register adapts these mismatched data widths. The output shift register is a 32-bit shift register, which always refills 32 bits at a time, either from the command expander output or directly from the data FIFO.

The source of data for the output shift register is configured by the `CSR.EXPAND_EN` field:

- when set, the command expander interposes the FIFO and the output shift register
- when clear, the command expander is bypassed, popping the FIFO directly into the shift register

Whenever `CSR.EN` is low, the shift register is flushed to empty. Once HSTX has been configured, and `EN` is set high, the shift register is ready to accept data, and will pop data as soon as it becomes available.

After popping the first data word, the shift register will now shift every *HSTX clock cycle* until it becomes empty. The shift behaviour is configured by:

- `CSR.N_SHIFTS`, which determines how many times to shift before the register is considered empty
- `CSR.SHIFT`, which is a *right-rotate* applied to the shift register every cycle

`CSR.N_SHIFTS` and `CSR.SHIFT` must only be changed when `CSR.EN` is low. It is safe to change these fields in the same register write that sets `EN` from low to high.

`SHIFT \times N_SHIFTS` is not necessarily less than or equal to 32. For example, a `SHIFT` of 31 might be used to shift the register *left* by one bit per cycle, since right-rotate is a modular operation, and -1 is equal to 31 under a modulus of 32.

When the shift register is about to become empty, it will immediately refill with fresh data from the command expander or FIFO if data is available. When data is available, the shift register is never empty for any cycle. If data is not available, the shift register becomes empty and stops shifting until more data is provided. Once data is provided, the shift register refills and begins shifting once again.

12.11.3. Bit Crossbar

The bit crossbar controls which bits of the output shift register appear on which GPIOs during the first and second half of each HSTX clock cycle. There is a configuration register for each pin, `BIT0` through `BIT7`:

- `BITx.SEL_P` selects which shift register bit (0 through 31) is output for the first half of each HSTX clock cycle
- `BITx.SEL_N` selects which shift register bit (0 through 31) is output for the second half of each clock cycle
- `BITx.INV` inverts the output (logical NOT)
- `BITx.CLK` indicates that this pin should be connected to the clock generator (Section 12.11.4) rather than the output shift register

To disable DDR behaviour set `SEL_N` equal to `SEL_P`. To implement a differential output, configure two pins identically except for the `INV` bit, which should be set for one pin and clear for the other.

12.11.3.1. Examples: One Pin

Together with the `SHIFT` and `N_SHIFTS` controls for the shift register, the pin configuration determines the data layout

passed through the HSTX. Since not all of us are accustomed to thinking in four dimensions, it's worth going through some examples with a single pin:

- $N_SHIFTS = 32, SHIFT = 1, SEL_P = 0, SEL_N = 0$:
 - Shift out one bit per HSTX clock cycle, LSB-first.
 - Each cycle, the shift register advances to the right by one, and the least-significant bit at that time is presented to the pin for both halves of the cycle, since SEL_P and SEL_N both select the same bit.
- $N_SHIFTS = 32, SHIFT = 31, SEL_P = 31, SEL_N = 31$:
 - Shift out one bit per HSTX clock cycle, MSB-first.
 - Each cycle, the shift register advances to the left by one (or rather, wraps around the right-hand edge of the register and ends up one bit left of where it started), and the most-significant bit at that time is presented to the pin.
- $N_SHIFTS = 16, SHIFT = 2, SEL_P = 0, SEL_N = 1$:
 - Shift out two bits per HSTX clock cycle, LSB-first.
 - Each cycle, the shift register advances to the right by two. The least-significant bit is presented to the pin for the first half of that cycle, and the neighbouring bit is presented for the second half.
- $N_SHIFTS = 16, SHIFT = 30, SEL_P = 31, SEL_N = 30$:
 - Shift out two bits per HSTX clock cycle, MSB-first.
 - Each cycle, the shift register advances to the left by two. The most-significant bit is presented to the pin for the first half of that cycle, and the neighbouring bit is presented for the latter half.
- $N_SHIFTS = 8, SHIFT = 4, SEL_P = 0, SEL_N = 0$:
 - Shift out the least-significant bit in each group of 4 bits, over the course of 8 clock cycles.
 - Each cycle, the shift register advances by to the right by four. The least-significant bit of the shift register is presented to the pin. The bit indices presented to the pin are therefore 0, 4, 8, 12, 16, 20, 24, and 28.
- $N_SHIFTS = 32, SHIFT = 4, SEL_P = 0, SEL_N = 0$:
 - Same as the previous, but repeats the 8-cycle pattern four times before refreshing the shift register.
 - Rotating by 32 restores the original value that was popped into the shift register from the FIFO or command expander.

12.11.3.2. Examples: Multiple Pins

The separation of shift register and bit crossbar allows both zipped and unzipped multi-bit records, once multiple pins are involved. For example, compare these two configurations:

- $N_SHIFTS = 8, SHIFT = 4, BIT0.SEL_P = 0, BIT0.SEL_N = 2, BIT1.SEL_P = 1, BIT1.SEL_N = 3$:
 - Each 32-bit word consists of 16 bit-pairs, and a new bit-pair is presented to $BIT0$ and $BIT1$ twice per cycle.
 - The shift register advances by 4 every cycle, introducing two new bit-pairs to the rightmost four bits of the shift register
- $N_SHIFTS = 8, SHIFT = 2, BIT0.SEL_P = 0, BIT0.SEL_N = 1, BIT1.SEL_P = 16, BIT1.SEL_N = 17$:
 - Each 32-bit word consists of a pair of 16-bit values, each of which is shifted to one pin out of $BIT0$ and $BIT1$ at a rate of two bits per cycle.
 - The shift register advances by two every cycle, introducing a new bit-pair to bits 1:0 for the $BIT0$ pin, and also introducing a new bit-pair to bits 17:16 for the $BIT1$ pin.

Depending on software needs, it may be preferable to pack together all of the bits output on the same cycle (zipped records), or all of the bits that go through the same pin (unzipped records), so HSTX supports both.

As a final, concrete example, take TMDS (used in DVI): here each 32-bit word contains 3×10 -bit TMDS symbols, each of which is serialised to a differential pair over the course of 10 TMDS bit times. For performance, it's preferable to make each HSTX clock period equal to two TMDS bit periods, by leveraging the DDR capability. A possible configuration would therefore be:

- **CSR:** `N_SHIFTS = 5, SHIFT = 2`
- **BIT0:** `SEL_P = 0, SEL_N = 1, INV = 0`
- **BIT1:** `SEL_P = 0, SEL_N = 1, INV = 1`
- **BIT2:** `SEL_P = 10, SEL_N = 11, INV = 0`
- **BIT3:** `SEL_P = 10, SEL_N = 11, INV = 1`
- **BIT4:** `SEL_P = 20, SEL_N = 21, INV = 0`
- **BIT5:** `SEL_P = 20, SEL_N = 21, INV = 1`

The missing piece for TMDS is the clock, which has a period of 10 TMDS bit periods, or 5 HSTX clock periods when shifting two bits per cycle per pin. HSTX has a special-purpose clock generator so that pseudo-clock bits do not have to be packed into the FIFO data stream. The clock generator is covered in the next section.

12.11.4. Clock Generator

The clock generator is a counter which provides a periodic signal over the course of `n` HSTX clock cycles, configured by `CSR.CLKDIV`. The clock period is always an integer number of HSTX clock cycles, in the range 1 to 16 inclusive. The clock generator supports both odd and even periods, using the DDR outputs to support mid-HSTX-cycle output transitions. There is only a single clock generator – to emulate multiple clocks, pack pseudo-clock bits into FIFO data.

The clock generator increments on cycles where the output shift register is shifted. Generally, the clock period will be a divisor of `CSR.N_SHIFTS` so that clock and data maintain a consistent alignment. In the TMDS example in the previous section, a `CLKDIV` of 5 would be suitable, so that the clock repeats every time the shift register refreshes. This matches the requirement for a TMDS clock period of 10 bit periods, since two bits are transferred every cycle.

The clock generator output is connected to any pin whose `BITx.CLK` bit is set (e.g. `BIT0.CLK`). To produce differential clock outputs, connect the clock to two pins, and invert one of them.

The `CSR.CLKPHASE` field defines the initial phase (count) of the clock generator, configured in units of one half HSTX clock cycle. The clock generator resets whenever `CSR.EN` is low and holds at this initial phase. Once `CSR.EN` is set and the output shift register begins to shift, the clock generator advances.

Clock generator output whilst `CSR.EN` is low is determined by the relation of clock period and initial clock phase: if the initial clock phase is less than one half clock period, then the output is initially low. Otherwise, it is initially high. The clock generator can be thought of as being low for the first half of each generation period, and high for the second half.

The maximum `CSR.CLKPHASE` is only 15 *half* HSTX clock cycles. The maximum `CSR.CLKDIV` is 16 *full* HSTX clock cycles: initial phases of greater than or equal to 180 degrees with the maximum clock period require the inversion of the clock using the bit crossbar inversion controls.

Only change `CSR.CLKPHASE` and `CSR.CLKDIV` when `CSR.EN` is low. It is safe to modify them in the same register write that sets `EN` from low to high.

12.11.4.1. Example: Centre-aligned Clock

When transmitting source-synchronous data, the data sink (the receiver) must not see data transitions too late before or too soon after the active edges of the clock. Violating these setup and hold constraints can lead to undefined operation of the external data sink.

Since the HSTX output delays are all mutually balanced, you can meet these constraints by placing clock transitions halfway between data transitions, known as centre-aligned clocking.

Since this positions the clock with a temporal resolution of one half of a bit time, the maximum data rate is one bit per HSTX clock cycle per pin. Because the clock already uses DDR, you cannot use DDR to increase the data rate. Therefore for all **BIT0** through **BIT7**, **BITx.SEL_N** is equal to **BITx.SEL_P**.

For single-data-rate data, with an active rising edge, use the following clock generator settings:

- **CSR.CLKDIV** = 1 (1 HSTX clock period)
- **CSR.CLKPHASE** = 1 (1/2 HSTX clock period)

The clock is delayed by half an HSTX cycle, to offset it from the launch of the first data.

For single-data-rate data, with an active *falling* edge, use the following clock generator settings:

- **CSR.CLKDIV** = 1 (1 HSTX clock period)
- **CSR.CLKPHASE** = 2 (1 HSTX clock period)

Alternatively, you could use the same settings as an active-rising edge clock, with the clock output inverted via the bit crossbar configuration.

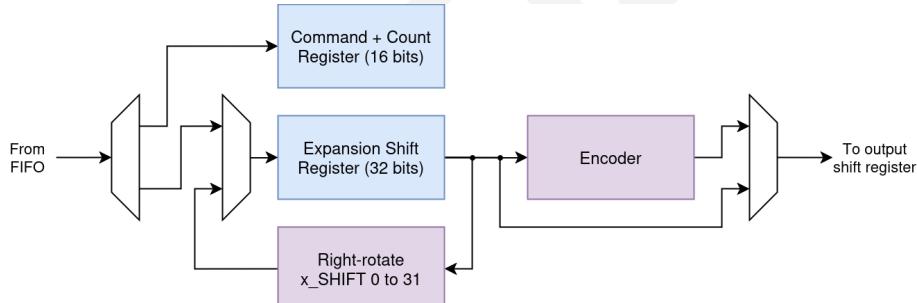
For double-data-rate data, with active rising and active falling edges, use the following clock generator settings:

- **CSR.CLKDIV** = 2 (2 HSTX clock period)
- **CSR.CLKPHASE** = 1 (1/2 HSTX clock period)

In all three cases, the data rate is the same, at 1 bit per HSTX clock cycle, per pin.

12.11.5. Command Expander

Figure 104. A mixture of commands and data are popped from the FIFO. Data can be repeated or shifted through the expansion shift register, and optionally passed through an encoder before passing on to the output shift register.



The command expander can be inserted inline between the data FIFO and the output shift register to manipulate the stream of data words. In general, the output stream is larger than the input stream, hence the name expander. The commander expander is enabled by setting **CSR.EXPAND_EN**. Only modify this field when **CSR.EN** is low. It is safe to modify this field in the same register write that sets **EN** from low to high. When the command expander is disabled, data passes directly from the data FIFO to the output shift register without being modified by the expander.

When the command expander is enabled, the data FIFO carries a mixture of data and commands for the expander. Each command consists of a 4-bit opcode and a 12-bit length, packed in the 16 LSBs of a data FIFO word, with the opcode in bits 15 through 12, and the length in bits 11 through 0. The available commands are:

- **0x0: RAW**
- **0x1: RAW_REPEAT**
- **0x2: TMDS**
- **0x3: TMDS_REPEAT**
- **0xf: NOP**

When the HSTX is first enabled, if the command expander is enabled, it expects the first word in the data FIFO to be a command. If this command is not a **NOP**, it will be followed by some amount of data, then another command. Operation continues in this manner, with runs of data interspersed with commands. A command always acts as a prefix to the

data that follows it in the FIFO.

The count field determines the number of words output by this command to the output shift register downstream, from 1 to 4095. A count of 0 is reserved to mean "infinite". The number of words that this command reads from the data FIFO in order to produce the specified quantity of downstream data depends on the command and the [EXPAND_SHIFT.ENC_N_SHIFTS](#) and [EXPAND_SHIFT.RAW_N_SHIFTS](#) register fields.

The expansion shift register always pops from the FIFO once at the beginning of the command. After this point, commands bearing the [x_REPEAT](#) suffix continue to circulate the same contents through the shift register, rotating right by [EXPAND_SHIFT.ENC_SHIFT](#) or [EXPAND_SHIFT.RAW_SHIFT](#) each time the output shift register pulls new data from the command expander. Use a shift of 0 to repeat identical data without shifting. This is useful, for example, for transmitting runs of the same TMDS control symbol during horizontal blanking periods in DVI.

RP2350 only implements a TMDS encoder, reserving the remaining opcode space for additional encoders in the future. [RAW](#) and [RAW_REPEAT](#) commands bypass the encoder. [TMDS](#) and [TMDS_REPEAT](#) commands are TMDS-encoded before being passed to the output shift register. [NOP](#) commands have no data, therefore whether they bypass the encoder or not is a philosophical question beyond the scope of this datasheet.

The [EXPAND_SHIFT](#) register has two copies for each of its fields. Fields prefixed with [RAW](#)_ are used for [RAW](#) and [RAW_REPEAT](#) commands. All other commands use fields prefixed with [ENC](#)_, which pass through the encoder. For example, in DVI, TMDS control symbols using [RAW_REPEAT](#) commands may be unshifted. Pixel data using TMDS commands may be shifted out one pixel at a time, so it is useful to have banked shift controls.

The [EXPAND_SHIFT.ENC_N_SHIFTS](#) and [EXPAND_SHIFT.RAW_N_SHIFTS](#) fields control how often the expansion shift register is refilled for encoded and raw commands respectively. [x_REPEAT](#) commands ignore these fields since they never refill from the FIFO, and function similarly to the [CSR.N_SHIFTS](#) field which controls the output shift register.

The command expander can only pop from the data FIFO once per cycle, so heavy use of commands (particularly [NOP](#) commands) can impact HSTX throughput. For use cases that output from the HSTX on every cycle, configure the output shift register with [CSR.N_SHIFTS](#) > 1. This is required because the command expander cannot output data on the cycle where it pops a command from the FIFO, so the expansion shift register is empty for at least one cycle.

12.11.6. PIO-to-HSTX Coupled Mode

HSTX can connect up to 8 PIO pin outputs to the bit crossbar. Only use the bit crossbar when [clk_hstx](#) connects directly to [clk_sys](#) ([CLK_HSTX_CTRL.AUXSRC](#) must select [clk_sys](#)).

NOTE

Running the two clocks at the same frequency is not sufficient. You must select [clk_sys](#) directly.

To enable coupled mode, set [CSR.COUPLED_MODE](#). The [COUPLED_SEL](#) field in the same register selects the PIO instance, 0 through 2, to couple to HSTX. When coupled mode is enabled, IO outputs 12 through 19 inclusive on the selected PIO instance appear at bit crossbar [PSEL_N](#) and [PSEL_P](#) indices 31:24, replacing the most significant 8 bits of the output shift register from the point of view of the bit crossbar.

This mode allows PIO programs to make use of the HSTX's DDR outputs. You can use this mode to drive a clock at the full system clock rate or to position clock transitions relative to data transitions with half-system-clock-cycle resolution.

The PIO outputs used for couple mode are always bits 19 through 12 of the pin outputs driven from that GPIO, independent of [\[reg-pio-GPIOBASE\]](#). When [GPIOBASE](#) is 0, the PIO outputs used for coupled mode are those that would normally appear on the HSTX pins. When [GPIOBASE](#) is 16, this uses the PIO outputs that would appear on GPIOs 28 through 35.

The operation of PIO is not affected in any way by coupled mode being enabled.

Outputs presented through the HSTX coupled mode interface have one additional system clock cycle of delay compared to those presented directly from PIO to the pads.

12.11.7. List of Control Registers

The control registers start at a base address of `0x400c0000` (defined as `HSTX_CTRL_BASE` in the SDK). They are accessed through an asynchronous bus crossing, so each bus access takes several cycles, the exact figure depending on the ratio of `clk_sys` and `clk_hstx`.

Table 1206. List of HSTX_CTRL registers

Offset	Name	Info
0x00	CSR	
0x04	BIT0	Data control register for output bit 0
0x08	BIT1	Data control register for output bit 1
0x0c	BIT2	Data control register for output bit 2
0x10	BIT3	Data control register for output bit 3
0x14	BIT4	Data control register for output bit 4
0x18	BIT5	Data control register for output bit 5
0x1c	BIT6	Data control register for output bit 6
0x20	BIT7	Data control register for output bit 7
0x24	EXPAND_SHIFT	Configure the optional shifter inside the command expander
0x28	EXPAND_TMDS	Configure the optional TMDS encoder inside the command expander

HSTX_CTRL: CSR Register

Offset: 0x00

Table 1207. CSR Register

Bits	Name	Description	Type	Reset
31:28	CLKDIV	<p>Clock period of the generated clock, measured in HSTX clock cycles. Can be odd or even. The generated clock advances only on cycles where the shift register shifts.</p> <p>For example, a clkdiv of 5 would generate a complete output clock period for every 5 HSTX clocks (or every 10 half-clocks).</p> <p>A CLKDIV value of 0 is mapped to a period of 16 HSTX clock cycles.</p>	RW	0x1

Bits	Name	Description	Type	Reset
27:24	CLKPHASE	<p>Set the initial phase of the generated clock.</p> <p>A CLKPHASE of 0 means the clock is initially low, and the first rising edge occurs after one half period of the generated clock (i.e. CLKDIV/2 cycles of clk_hstx).</p> <p>Incrementing CLKPHASE by 1 will advance the initial clock phase by one half clk_hstx period. For example, if CLKDIV=2 and CLKPHASE=1:</p> <ul style="list-style-type: none"> * The clock will be initially low * The first rising edge will be 0.5 clk_hstx cycles after asserting first data * The first falling edge will be 1.5 clk_hstx cycles after asserting first data <p>This configuration would be suitable for serialising at a bit rate of clk_hstx with a centre-aligned DDR clock.</p> <p>When the HSTX is halted by clearing CSR_EN, the clock generator will return to its initial phase as configured by the CLKPHASE field.</p> <p>Note CLKPHASE must be strictly less than double the value of CLKDIV (one full period), else its operation is undefined.</p>	RW	0x0
23:21	Reserved.	-	-	-
20:16	N_SHIFTS	<p>Number of times to shift the shift register before refilling it from the FIFO. (A count of how many times it has been shifted, not the total shift distance.)</p> <p>A register value of 0 means shift 32 times.</p>	RW	0x05
15:13	Reserved.	-	-	-
12:8	SHIFT	<p>How many bits to right-rotate the shift register by each cycle.</p> <p>The use of a rotate rather than a shift allows left shifts to be emulated, by subtracting the left-shift amount from 32. It also allows data to be repeated, when the product of SHIFT and N_SHIFTS is greater than 32.</p>	RW	0x06
7	Reserved.	-	-	-
6:5	COUPLED_SEL	Select which PIO to use for coupled mode operation.	RW	0x0

Bits	Name	Description	Type	Reset
4	COUPLED_MODE	<p>Enable the PIO-to-HSTX 1:1 connection. The HSTX must be clocked directly from the system clock (not just from some other clock source of the same frequency) for this synchronous interface to function correctly.</p> <p>When COUPLED_MODE is set, BITx_SEL_P and SEL_N indices 24 through 31 will select bits from the 8-bit PIO-to-HSTX path, rather than shifter bits. Indices of 0 through 23 will still index the shift register as normal.</p> <p>The PIO outputs connected to the PIO-to-HSTX bus are those same outputs that would appear on the HSTX-capable pins if those pins' FUNCSELs were set to PIO instead of HSTX.</p> <p>For example, if HSTX is on GPIOs 12 through 19, then PIO outputs 12 through 19 are connected to the HSTX when coupled mode is engaged.</p>	RW	0x0
3:2	Reserved.	-	-	-
1	EXPAND_EN	<p>Enable the command expander. When 0, raw FIFO data is passed directly to the output shift register. When 1, the command expander can perform simple operations such as run length decoding on data between the FIFO and the shift register.</p> <p>Do not change CXPD_EN whilst EN is set. It's safe to set CXPD_EN simultaneously with setting EN.</p>	RW	0x0
0	EN	<p>When EN is 1, the HSTX will shift out data as it appears in the FIFO. As long as there is data, the HSTX shift register will shift once per clock cycle, and the frequency of popping from the FIFO is determined by the ratio of SHIFT and SHIFT_THRESH.</p> <p>When EN is 0, the FIFO is not popped. The shift counter and clock generator are also reset to their initial state for as long as EN is low. Note the initial phase of the clock generator can be configured by the CLKPHASE field.</p> <p>Once the HSTX is enabled again, and data is pushed to the FIFO, the generated clock's first rising edge will be one half-period after the first data is launched.</p>	RW	0x0

HSTX_CTRL: BIT0 Register

Offset: 0x04

Description

Data control register for output bit 0

Table 1208. BIT0 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: BIT1 Register

Offset: 0x08

Description

Data control register for output bit 1

Table 1209. BIT1 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: BIT2 Register

Offset: 0x0c

Description

Data control register for output bit 2

Table 1210. BIT2 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0

Bits	Name	Description	Type	Reset
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: BIT3 Register

Offset: 0x10

Description

Data control register for output bit 3

Table 1211. BIT3 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: BIT4 Register

Offset: 0x14

Description

Data control register for output bit 4

Table 1212. BIT4 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: BIT5 Register

Offset: 0x18

Description

Data control register for output bit 5

Table 1213. BIT5 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: BIT6 Register

Offset: 0x1c

Description

Data control register for output bit 6

Table 1214. BIT6 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: BIT7 Register

Offset: 0x20

Description

Data control register for output bit 7

Table 1215. BIT7 Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	CLK	Connect this output to the generated clock, rather than the data shift register. SEL_P and SEL_N are ignored if this bit is set, but INV can still be set to generate an antiphase clock.	RW	0x0
16	INV	Invert this data output (logical NOT)	RW	0x0
15:13	Reserved.	-	-	-
12:8	SEL_N	Shift register data bit select for the second half of the HSTX clock cycle	RW	0x00
7:5	Reserved.	-	-	-
4:0	SEL_P	Shift register data bit select for the first half of the HSTX clock cycle	RW	0x00

HSTX_CTRL: EXPAND_SHIFT Register

Offset: 0x24

Description

Configure the optional shifter inside the command expander

Table 1216. EXPAND_SHIFT Register

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28:24	ENC_N_SHIFTS	Number of times to consume from the shift register before refilling it from the FIFO, when the current command is an encoded data command (e.g. TMDS). A register value of 0 means shift 32 times.	RW	0x01
23:21	Reserved.	-	-	-
20:16	ENC_SHIFT	How many bits to right-rotate the shift register by each time data is pushed to the output shifter, when the current command is an encoded data command (e.g. TMDS).	RW	0x00
15:13	Reserved.	-	-	-
12:8	RAW_N_SHIFTS	Number of times to consume from the shift register before refilling it from the FIFO, when the current command is a raw data command. A register value of 0 means shift 32 times.	RW	0x01
7:5	Reserved.	-	-	-
4:0	RAW_SHIFT	How many bits to right-rotate the shift register by each time data is pushed to the output shifter, when the current command is a raw data command.	RW	0x00

HSTX_CTRL: EXPAND_TMDS Register

Offset: 0x28

Description

Configure the optional TMDS encoder inside the command expander

Table 1217.
EXPAND_TMDs
Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:21	L2_NBITS	Number of valid data bits for the lane 2 TMDS encoder, starting from bit 7 of the rotated data. Field values of 0 → 7 encode counts of 1 → 8 bits.	RW	0x0
20:16	L2_ROT	Right-rotate applied to the current shifter data before the lane 2 TMDS encoder.	RW	0x00
15:13	L1_NBITS	Number of valid data bits for the lane 1 TMDS encoder, starting from bit 7 of the rotated data. Field values of 0 → 7 encode counts of 1 → 8 bits.	RW	0x0
12:8	L1_ROT	Right-rotate applied to the current shifter data before the lane 1 TMDS encoder.	RW	0x00
7:5	L0_NBITS	Number of valid data bits for the lane 0 TMDS encoder, starting from bit 7 of the rotated data. Field values of 0 → 7 encode counts of 1 → 8 bits.	RW	0x0
4:0	L0_ROT	Right-rotate applied to the current shifter data before the lane 0 TMDS encoder.	RW	0x00

12.11.8. List of FIFO Registers

The FIFO registers start at a base address of [0x50600000](#) (defined as [HSTX_FIFO_BASE](#) in the SDK).

Table 1218. List of
HSTX_FIFO registers

Offset	Name	Info
0x0	STAT	FIFO status
0x4	FIFO	Write access to FIFO

HSTX_FIFO: STAT Register

Offset: 0x0

Description

FIFO status

Table 1219. STAT
Register

Bits	Name	Description	Type	Reset
31:11	Reserved.	-	-	-
10	WOF	FIFO was written when full. Write 1 to clear.	WC	0x0
9	EMPTY		RO	-
8	FULL		RO	-
7:0	LEVEL		RO	0x00

HSTX_FIFO: FIFO Register

Offset: 0x4

Table 1220. FIFO Register

Bits	Description	Type	Reset
31:0	Write access to FIFO	WF	0x00000000

12.12. TRNG

12.12.1. Overview

RP2350 contains an Arm IP-based True Random Number Generator block. It supports the following features:

- Compliance with FIPS Publication 140-2, BSI AIS-31, and NIST SP 800-90B
- Produces approximately 7.5Kib/s of entropy when the core runs at 150MHz

On request, the TRNG block generates a block of 192 entropy bits generated by automatically processing a series of periodic samples from the TRNG block's internal Ring Oscillator (ROSC).

The TRNG block's ROSC is a free-running oscillator with no direct connection to the system clocks on RP2350. As a result, the ROSC generally runs asynchronously to the system clocks.

After a sufficient number of samples have been collected, the TRNG block completes the generation process and presents the random number in the [EHR_DATA\[x\]](#) result registers.

For more information, see [ARM IP - True Random Number Generator](#)

12.12.2. Configuration

The TRNG block contains three different built-in entropy checking mechanisms: At reset, these are all enabled by default and hence do not require explicit enabling.

You can configure the TRNG block in the following ways:

- Configure the frequency of the ROSC by selecting of one of four ROSC **chain lengths**, see [TRNG_CONFIG](#).
- Configure the ROSC sampling period in terms of system clock ticks, see [SAMPLE_CNT1](#).

Because the system clock generally runs much faster than the ROSC, the sampling period is expected to be at least a few tens of system clock ticks.

Because the characteristics of the TRNG ROSC and system clock frequency will differ for each implementation of the TRNG IP block, Arm details a TRNG characterisation procedure to determine the most appropriate ROSC chain length and sampling frequency settings on each SoC design. For details about that characterisation procedure, see [ARM TrustZone True Number Generator](#).

Software drivers for the RP2350 TRNG block do not utilise the standard approach (see [Section 12.12.4](#)). As a result, software does not configure the ROSC length and sample count settings provided by the Arm characterisation procedure.

When configuring the TRNG block, consider the following principles:

- As average generation time increases, result quality increases and failed entropy checks decrease.
- A low sample count decreases average generation time, but increases the chance of NIST test-failing results and failed entropy checks.

For acceptable results with an average generation time of about 2 milliseconds, use ROSC chain length settings of 0 or 1 and sample count settings of 20-25.

Larger sample count settings (e.g. 100) provide proportionately slower average generation times. These settings significantly reduce, but do not eliminate NIST test failures and entropy check failures. Results occasionally take an

especially long time to generate.

12.12.3. Operation

To initiate TRNG generation, set the [RND_SRC_EN](#) bit in [RND_SOURCE_ENABLE](#). The TRNG will run until:

- It has successfully completed the generation of a random number.
- One, or more, of the internal entropy checking mechanisms indicates a failed run.

In either case, you can read the resultant status from [RNG_ISR](#).

To generate TRNG block interrupts, set bits in [RNG_IMR](#). Use [RNG_ICR](#) to clear active interrupt status bits.

The [EHR_DATA\[x\]](#) registers read 0 until successful generation has occurred, so the CPU cannot read random number results during generation,

After successful generation, read the last result register, [EHR_DATA\[5\]](#) to clear all of the result registers. If the result fails an entropy check, no results are presented and the [EHR_DATA\[x\]](#) registers all read as 0.

After TRNG generation and when not in use, the [RND_SRC_EN](#) bit should be cleared.

12.12.4. Caveats

The generation of random numbers by the TRNG block is not a deterministic process.

Although the modal and mean average times required to generate random numbers are quite similar, the generation process can occasionally take *much* longer to complete: in excess of 100 times the average. Any run resulting in a failed entropy check discards the result, requiring another generation process.

You can accommodate these unpredictable generation times in your system design. For example, you might generate a small pool of random numbers, initiating subsequent generation whenever space becomes available in the pool.

In the interests of simplicity and timing predictability, alternative approaches were adopted for the RP2350 bootrom and the SDK TRNG block drivers. The methodologies used can be found via the links below. However, nothing in the TRNG block in RP2350 precludes using the block as specified in Arm documentation.

[TO DO link to bootrom source](#) [TO DO link to SDK TRNG source](#)

12.12.5. List of Registers

The TRNG control registers start at a base address of [0x400f0000](#) (defined as [TRNG_BASE](#) in the SDK).

Table 1221. List of TRNG registers

Offset	Name	Info
0x100	RNG_IMR	Interrupt masking.
0x104	RNG_ISR	RNG status register. If corresponding RNG_IMR bit is unmasked, an interrupt will be generated.
0x108	RNG_ICR	Interrupt/status bit clear Register.
0x10c	TRNG_CONFIG	Selecting the inverter-chain length.
0x110	TRNG_VALID	192 bit collection indication.
0x114	EHR_DATA0	RNG collected bits.
0x118	EHR_DATA1	RNG collected bits.
0x11c	EHR_DATA2	RNG collected bits.

Offset	Name	Info
0x120	EHR_DATA3	RNG collected bits.
0x124	EHR_DATA4	RNG collected bits.
0x128	EHR_DATA5	RNG collected bits.
0x12c	RND_SOURCE_ENABLE	Enable signal for the random source.
0x130	SAMPLE_CNT1	Counts clocks between sampling of random bit.
0x134	AUTOCORR_STATISTIC	Statistic about Autocorrelation test activations.
0x138	TRNG_DEBUG_CONTROL	Debug register.
0x140	TRNG_SW_RESET	Generate internal SW reset within the RNG block.
0x1b4	RNG_DEBUG_EN_INPUT	Enable the RNG debug mode
0x1b8	TRNG_BUSY	RNG Busy indication.
0x1bc	RST_BITS_COUNTER	Reset the counter of collected bits in the RNG.
0x1c0	RNG_VERSION	Displays the version settings of the TRNG.
0x1e0	RNG_BIST_CNTR_0	Collected BIST results.
0x1e4	RNG_BIST_CNTR_1	Collected BIST results.
0x1e8	RNG_BIST_CNTR_2	Collected BIST results.

TRNG: RNG_IMR Register

Offset: 0x100

Description

Interrupt masking.

Table 1222. RNG_IMR Register

Bits	Name	Description	Type	Reset
31:4	RESERVED	RESERVED	RO	0x0000000
3	VN_ERR_INT_MAS K	1'b1-mask interrupt, no interrupt will be generated. See RNG_ISR for an explanation on this interrupt.	RW	0x1
2	CRNGT_ERR_INT_ MASK	1'b1-mask interrupt, no interrupt will be generated. See RNG_ISR for an explanation on this interrupt.	RW	0x1
1	AUTOCORR_ERR_I NT_MASK	1'b1-mask interrupt, no interrupt will be generated. See RNG_ISR for an explanation on this interrupt.	RW	0x1
0	EHR_VALID_INT_ MASK	1'b1-mask interrupt, no interrupt will be generated. See RNG_ISR for an explanation on this interrupt.	RW	0x1

TRNG: RNG_ISR Register

Offset: 0x104

Description

RNG status register. If corresponding RNG_IMR bit is unmasked, an interrupt will be generated.

Table 1223. RNG_ISR Register

Bits	Name	Description	Type	Reset
31:4	RESERVED	RESERVED	RO	0x0000000

Bits	Name	Description	Type	Reset
3	VN_ERR	1'b1 indicates Von Neuman error. Error in von Neuman occurs if 32 consecutive collected bits are identical, ZERO or ONE.	RO	0x0
2	CRNGT_ERR	1'b1 indicates CRNGT in the RNG test failed. Failure occurs when two consecutive blocks of 16 collected bits are equal.	RO	0x0
1	AUTOCORR_ERR	1'b1 indicates Autocorrelation test failed four times in a row. When set, RNG cease from functioning until next reset.	RO	0x0
0	EHR_VALID	1'b1 indicates that 192 bits have been collected in the RNG, and are ready to be read.	RO	0x0

TRNG: RNG_ICR Register

Offset: 0x108

Description

Interrupt/status bit clear Register.

Table 1224. RNG_ICR Register

Bits	Name	Description	Type	Reset
31:4	RESERVED	RESERVED	RO	0x0000000
3	VN_ERR	Write 1'b1 - clear corresponding bit in RNG_ISR.	RW	0x0
2	CRNGT_ERR	Write 1'b1 - clear corresponding bit in RNG_ISR.	RW	0x0
1	AUTOCORR_ERR	Cannot be cleared by SW! Only RNG reset clears this bit.	RW	0x0
0	EHR_VALID	Write 1'b1 - clear corresponding bit in RNG_ISR.	RW	0x0

TRNG: TRNG_CONFIG Register

Offset: 0x10c

Description

Selecting the inverter-chain length.

Table 1225. TRNG_CONFIG Register

Bits	Name	Description	Type	Reset
31:2	RESERVED	RESERVED	RO	0x00000000
1:0	RND_SRC_SEL	Selects the number of inverters (out of four possible selections) in the ring oscillator (the entropy source).	RW	0x0

TRNG: TRNG_VALID Register

Offset: 0x110

Description

192 bit collection indication.

Table 1226.
TRNG_VALID Register

Bits	Name	Description	Type	Reset
31:1	RESERVED	RESERVED	RO	0x00000000
0	EHR_VALID	1'b1 indicates that collection of bits in the RNG is completed, and data can be read from EHR_DATA register.	RO	0x0

TRNG: EHR_DATA0, EHR_DATA1, ..., EHR_DATA4, EHR_DATA5 Registers

Offsets: 0x114, 0x118, ..., 0x124, 0x128

Description

RNG collected bits.

Table 1227.
EHR_DATA0,
EHR_DATA1, ...,
EHR_DATA4,
EHR_DATA5 Registers

Bits	Description	Type	Reset
31:0	Bits [(32*(i+1))-1:(32*i)] of Entropy Holding Register.	RO	0x00000000

TRNG: RND_SOURCE_ENABLE Register

Offset: 0x12c

Description

Enable signal for the random source.

Table 1228.
RND_SOURCE_ENABLE Register

Bits	Name	Description	Type	Reset
31:1	RESERVED	RESERVED	RO	0x00000000
0	RND_SRC_EN	* 1'b1 - entropy source is enabled. *1'b0 - entropy source is disabled	RW	0x0

TRNG: SAMPLE_CNT1 Register

Offset: 0x130

Description

Counts clocks between sampling of random bit.

Table 1229.
SAMPLE_CNT1 Register

Bits	Name	Description	Type	Reset
31:0	SAMPLE_CNTR1	Sets the number of rng_clk cycles between two consecutive ring oscillator samples. Note! If the Von-Neuman is bypassed, the minimum value for sample counter must not be less than decimal seventeen	RW	0x0000ffff

TRNG: AUTOCORR_STATISTIC Register

Offset: 0x134

Description

Statistic about Autocorrelation test activations.

Table 1230.
AUTOCORR_STATISTI C Register

Bits	Name	Description	Type	Reset
31:22	RESERVED	RESERVED	RO	0x000
21:14	AUTOCORR_FAILS	Count each time an autocorrelation test fails. Any write to the register reset the counter. Stop collecting statistic if one of the counters reached the limit.	RW	0x00

Bits	Name	Description	Type	Reset
13:0	AUTOCORR_TRYS	Count each time an autocorrelation test starts. Any write to the register reset the counter. Stop collecting statistic if one of the counters reached the limit.	RW	0x0000

TRNG: TRNG_DEBUG_CONTROL Register

Offset: 0x138

Description

Debug register.

Table 1231.
TRNG_DEBUG CONTR
OL Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	AUTO_CORRELAT E_BYPASS	When set, the autocorrelation test in the TRNG module is bypassed.	RW	0x0
2	TRNG_CRNGT_BY PASS	When set, the CRNGT test in the RNG is bypassed.	RW	0x0
1	VNC_BYPASS	When set, the Von-Neuman balancer is bypassed (including the 32 consecutive bits test).	RW	0x0
0	RESERVED	N/A	RO	0x0

TRNG: TRNG_SW_RESET Register

Offset: 0x140

Description

Generate internal SW reset within the RNG block.

Table 1232.
TRNG_SW_RESET
Register

Bits	Name	Description	Type	Reset
31:1	RESERVED	RESERVED	RO	0x00000000
0	TRNG_SW_RESET	Writing 1'b1 to this register causes an internal RNG reset.	RW	0x0

TRNG: RNG_DEBUG_EN_INPUT Register

Offset: 0x1b4

Description

Enable the RNG debug mode

Table 1233.
RNG_DEBUG_EN_INPU
T Register

Bits	Name	Description	Type	Reset
31:1	RESERVED	RESERVED	RO	0x00000000
0	RNG_DEBUG_EN	* 1'b1 - debug mode is enabled. *1'b0 - debug mode is disabled	RW	0x0

TRNG: TRNG_BUSY Register

Offset: 0x1b8

Description

RNG Busy indication.

Table 1234.
TRNG_BUSY Register

Bits	Name	Description	Type	Reset
31:1	RESERVED	RESERVED	RO	0x00000000
0	TRNG_BUSY	Reflects rng_busy status.	RO	0x0

TRNG: RST_BITS_COUNTER Register

Offset: 0x1bc

Description

Reset the counter of collected bits in the RNG.

Table 1235.
RST_BITS_COUNTER Register

Bits	Name	Description	Type	Reset
31:1	RESERVED	RESERVED	RO	0x00000000
0	RST_BITS_COUNTER	Writing any value to this address will reset the bits counter and RNG valid registers. RND_SOURCE_ENABLE register must be unset in order for the reset to take place.	RW	0x0

TRNG: RNG_VERSION Register

Offset: 0x1c0

Description

Displays the version settings of the TRNG.

Table 1236.
RNG_VERSION Register

Bits	Name	Description	Type	Reset
31:8	RESERVED	RESERVED	RO	0x0000000
7	RNG_USE_5_SBOXES	* 1'b1 - 5 SBOX AES. *1'b0 - 20 SBOX AES	RO	0x0
6	RESEEDING_EXISTS	* 1'b1 - Exists. *1'b0 - Does not exist	RO	0x0
5	KAT_EXISTS	* 1'b1 - Exists. *1'b0 - Does not exist	RO	0x0
4	PRNG_EXISTS	* 1'b1 - Exists. *1'b0 - Does not exist	RO	0x0
3	TRNG_TESTS_BY_PASS_EN	* 1'b1 - Exists. *1'b0 - Does not exist	RO	0x0
2	AUTOCORR_EXISTS	* 1'b1 - Exists. *1'b0 - Does not exist	RO	0x0
1	CRNGT_EXISTS	* 1'b1 - Exists. *1'b0 - Does not exist	RO	0x0
0	EHR_WIDTH_192	* 1'b1 - 192-bit EHR. *1'b0 - 128-bit EHR	RO	0x0

TRNG: RNG_BIST_CNTR_0 Register

Offset: 0x1e0

Description

Collected BIST results.

Table 1237.
RNG_BIST_CNTR_0
Register

Bits	Name	Description	Type	Reset
31:22	RESERVED	RESERVED	RO	0x000
21:0	ROSC_CNTR_VAL	Reflects the results of RNG BIST counter.	RO	0x000000

TRNG: RNG_BIST_CNTR_1 Register

Offset: 0x1e4

Description

Collected BIST results.

Table 1238.
RNG_BIST_CNTR_1
Register

Bits	Name	Description	Type	Reset
31:22	RESERVED	RESERVED	RO	0x000
21:0	ROSC_CNTR_VAL	Reflects the results of RNG BIST counter.	RO	0x000000

TRNG: RNG_BIST_CNTR_2 Register

Offset: 0x1e8

Description

Collected BIST results.

Table 1239.
RNG_BIST_CNTR_2
Register

Bits	Name	Description	Type	Reset
31:22	RESERVED	RESERVED	RO	0x000
21:0	ROSC_CNTR_VAL	Reflects the results of RNG BIST counter.	RO	0x000000

12.13. SHA-256 Accelerator

RP2350 is equipped with an implementation of the SHA-256 hash algorithm, as defined in the FIPS 180-4 standard available from NIST publications. A hash algorithm digests an arbitrary-length stream of data, known as the **message**, and produces a fixed-size result, known as a **hash**. In the case of SHA-256, the result is always 256 bits in size. Hash algorithms are designed such that:

- Given the hash, it is impossible (or implausibly computationally hard) to recover the original message.
- Small changes to the original message result, on average, in large changes to the hash.
- Given a message with a particular hash, it is impossible (or implausibly computationally hard) to generate a different message with the same hash.

These properties make hash algorithms useful for checking the integrity of data, in the face of both accidental bit flips and deliberate tampering.

To compute a SHA-256 with the RP2350 SHA-256 accelerator:

- Initialise the algorithm state by writing a 1 to **CSR.START**.
- Write the message to the **WDATA** register, polling **CSR.WDATA_RDY** in between writes.
- Write additional trailer and padding data to **WDATA**, as described in [Section 12.13.1](#) below.
- Poll **CSR.SUM_VLD** to wait for the last block to be digested.
- Read the 256-bit result from the 8 read-only result registers starting at **SUM0**.

12.13.1. Message Padding

Pad message content according to the standard SHA-256 method as described in the [FIPS 180-4 Secure Hash Standard](#): append the message with single bit 1, then a number of 0 bits, then a 64-bit count of the number of message bits. So for a message M with length L bits the padded message should be:

1. message M
2. 1
3. k zero bits, where k is the smallest non-negative solution to the equation: $L + 1 + k = 448 \text{ mod } 512$
4. a 64-bit block indicating L (the length of the message) in binary

For example, the 8 bit ASCII message `abc` has a length of 24 bits. This is padded with 1, then $448 - (24 + 1) = 423$ 0 bits, and then the message length as a 64-bit value as follows:

```
01100001 01100010 01100011 1 00000000 000...0 00000000 000...0 00011000
|-----message-----| 1 |--423 0 bits--| |-----64 bit len-----|
```

12.13.2. Throughput

SHA-256 processes data one 512-bit block at a time. This requires 16 32-bit writes, 32 16-bit writes, or 64 8-bit writes to the [WDATA](#) register. An APB register write costs 4 cycles, so it takes at least 64 system clock cycles to write a data block.

Once a full block is transferred, the SHA core takes a further 57 cycles to complete the block digest. [CSR.WDATA_RDY](#) goes low, and you must not write to [WDATA](#) during this time.

The maximum throughput is therefore one block per 121 system clock cycles, or 0.53 bytes per cycle. At a 150 MHz system clock this is 79.3 MB/s. This throughput is achieved when you use 32-bit transfers from the DMA. Using narrower transfers result in lower throughput, as does polling the [CSR.WDATA_RDY](#) flag when transferring data from the processor.

12.13.3. Data Size and Endianness

Data is sent in message blocks of 512 bits, padded as described in [Section 12.13.1](#). The SHA-256 accelerator updates its 256-bit output state for each input block. The SHA-256 algorithm is defined in terms of big-endian message words, but this accelerator provides a byte swap function via [CSR.BSWAP](#) to support little-endian data. [BSWAP](#) is set by default. For more information, see the register descriptions.

[WDATA](#) supports 8-bit, 16-bit and 32-bit writes. The bus interface accumulates 8 and 16-bit writes in a 32-bit shift register before passing them into the SHA-256 algorithm core. This means you must take care when mixing writes of different sizes, because taking the shift register level from less than to greater than 32 bits in a single write will silently drop data. You can avoid this issue by not mixing [WDATA](#) write sizes within a single SHA-256 message block (64 bytes).

12.13.4. DMA DREQ Interface

The block can request the DMA controller to send entire blocks of data at once. Configure transfer size using [CSR.DMA_SIZE](#) so that the DMA controller requests the correct number of transfers.

The DREQ always requests one full SHA block of data at a time. Do not start a DMA on a non-block boundary.

12.13.5. List of Registers

The SHA-256 registers start at a base address of **0x400f8000** (defined as **SHA256_BASE** in SDK).

Table 1240. List of SHA256 registers

Offset	Name	Info
0x00	CSR	Control and status register
0x04	WDATA	Write data register
0x08	SUM0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.
0x0c	SUM1	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.
0x10	SUM2	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.
0x14	SUM3	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.
0x18	SUM4	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.
0x1c	SUM5	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.
0x20	SUM6	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.
0x24	SUM7	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.

SHA256: CSR Register

Offset: 0x00

Description

Control and status register

Table 1241. CSR Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
12	BSWAP	<p>Enable byte swapping of 32-bit values at the point they are committed to the SHA message scheduler.</p> <p>This block's bus interface assembles byte/halfword data into message words in little-endian order, so that DMAing the same buffer with different transfer sizes always gives the same result on a little-endian system like RP2350.</p> <p>However, when marshalling bytes into blocks, SHA expects that the first byte is the most significant in each message word. To resolve this, once the bus interface has accumulated 32 bits of data (either a word write, two halfword writes in little-endian order, or four byte writes in little-endian order) the final value can be byte-swapped before passing to the actual SHA core.</p> <p>This feature is enabled by default because using the SHA core to checksum byte buffers is expected to be more common than having preformatted SHA message words lying around.</p>	RW	0x1
11:10	Reserved.	-	-	-
9:8	DMA_SIZE	<p>Configure DREQ logic for the correct DMA data size. Must be configured before the DMA channel is triggered.</p> <p>The SHA-256 core's DREQ logic requests one entire block of data at once, since there is no FIFO, and data goes straight into the core's message schedule and digest hardware. Therefore, when transferring data with DMA, CSR_DMA_SIZE must be configured in advance so that the correct number of transfers can be requested per block.</p> <p>0x0 → 8bit 0x1 → 16bit 0x2 → 32bit</p>	RW	0x2
7:5	Reserved.	-	-	-
4	ERR_WDATA_NOT_RDY	Set when a write occurs whilst the SHA-256 core is not ready for data (WDATA_RDY is low). Write one to clear.	WC	0x0
3	Reserved.	-	-	-
2	SUM_VLD	<p>If 1, the SHA-256 checksum presented in registers SUM0 through SUM7 is currently valid.</p> <p>Goes low when WDATA is first written, then returns high once 16 words have been written and the digest of the current 512-bit block has subsequently completed.</p>	RO	0x1
1	WDATA_RDY	<p>If 1, the SHA-256 core is ready to accept more data through the WDATA register.</p> <p>After writing 16 words, this flag will go low for 57 cycles whilst the core completes its digest.</p>	RO	0x1

Bits	Name	Description	Type	Reset
0	START	<p>Write 1 to prepare the SHA-256 core for a new checksum.</p> <p>The SUMx registers are initialised to the proper values (fractional bits of square roots of first 8 primes) and internal counters are cleared. This immediately forces WDATA_RDY and SUM_VLD high.</p> <p>START must be written before initiating a DMA transfer to the SHA-256 core, because the core will always request 16 transfers at a time (1 512-bit block). Additionally, the DMA channel should be configured for a multiple of 16 32-bit transfers.</p>	SC	0x0

SHA256: WDATA Register

Offset: 0x04

Description

Write data register

Table 1242. WDATA Register

Bits	Description	Type	Reset
31:0	<p>After pulsing START and writing 16 words of data to this register, WDATA_RDY will go low and the SHA-256 core will complete the digest of the current 512-bit block.</p> <p>Software is responsible for ensuring the data is correctly padded and terminated to a whole number of 512-bit blocks.</p> <p>After this, WDATA_RDY will return high, and more data can be written (if any).</p> <p>This register supports word, halfword and byte writes, so that DMA from non-word-aligned buffers can be supported. The total amount of data per block remains the same (16 words, 32 halfwords or 64 bytes) and byte/halfword transfers must not be mixed within a block.</p>	WF	0x00000000

SHA256: SUM0 Register

Offset: 0x08

Table 1243. SUM0 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

SHA256: SUM1 Register

Offset: 0x0c

Table 1244. SUM1 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

SHA256: SUM2 Register

Offset: 0x10

Table 1245. SUM2 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

SHA256: SUM3 Register

Offset: 0x14

Table 1246. SUM3 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

SHA256: SUM4 Register

Offset: 0x18

Table 1247. SUM4 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

SHA256: SUM5 Register

Offset: 0x1c

Table 1248. SUM5 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

SHA256: SUM6 Register

Offset: 0x20

Table 1249. SUM6 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

SHA256: SUM7 Register

Offset: 0x24

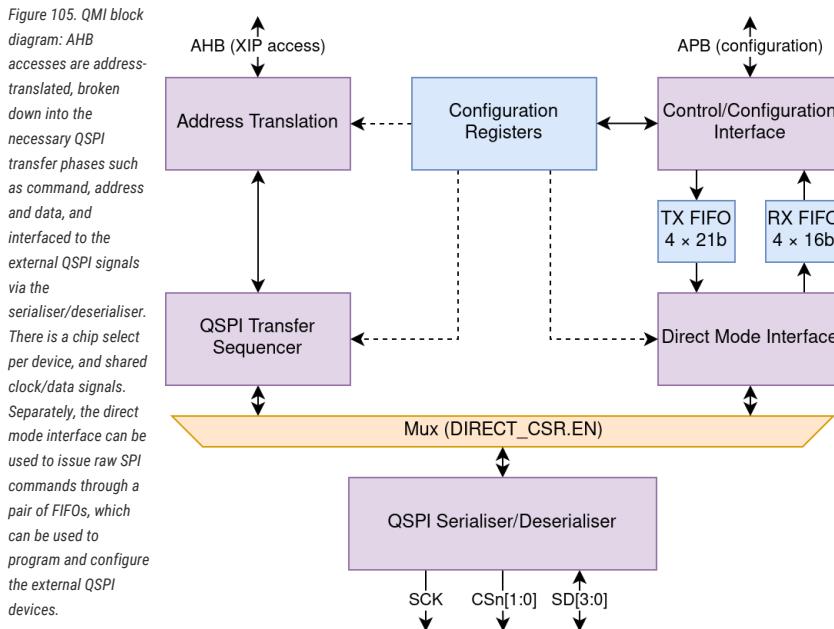
Table 1250. SUM7 Register

Bits	Description	Type	Reset
31:0	256-bit checksum result. Contents are undefined when CSR_SUM_VLD is 0.	RO	0x00000000

12.14. QSPI Memory Interface (QMI)

12.14.1. Overview

The QSPI memory interface (QMI) provides read/write memory-mapped access to two external QSPI memory devices. RP2350 has a single QMI instance, embedded in the XIP subsystem (Section 4.4), which replaces the SSI interface present on RP2040. The QMI supports serial-SPI, dual-SPI, and quad-SPI transfers, with two chip selects and shared clock/data signals.



Each chip select corresponds to a 16MB AHB address window, so a maximum of 32MB of external memory is supported. Chip select 0, which has a dedicated external pin, is mapped to addresses starting from `0x10000000`, and chip select 1, which is available as an alternate GPIO function, starts from `0x11000000`. This mapping is mirrored in the uncached and uncached + untranslated XIP address windows described in [Section 4.4](#).

All timing and SPI command format parameters are configured per chip select, with the correct configuration used automatically based on address decode. For example, `M0_TIMING` configures timing parameters for accesses to chip select 0, and `M1_TIMING` is an identical register for chip select 1.

The serial clock (`SCK`) is any integer division of the system clock in the range 1 to 256. The divisors can be adjusted at any time. Input sample timing can be adjusted in half-system-clock-cycle increments, to compensate for clock-to-data delay at high `SCK` frequencies. Double transfer rate mode (DTR) is implemented by halving the `SCK` frequency whilst maintaining the data transfer rate, which is capped at 4 bits per system clock cycle.

The number of `SCK` cycles issued for each access depends on the access size, which varies between one byte and one cache line. For example, an uncached one-byte read by a processor will fetch exactly one byte of data over the QSPI bus, to avoid wasting time fetching unwanted data. Cache misses are always issued as 64-bit QSPI transfers.

Optionally, the QMI can automatically chain sequentially addressed AHB accesses into a single, long QSPI transfer. This avoids issuing redundant commands and addresses on the QSPI bus, and is particularly beneficial for cold code paths and for streaming in flash data using the XIP streaming hardware ([Section 4.4.3](#)). For PSRAM compatibility, chains can be broken when they exceed a maximum chip select time (`M0_TIMING.MAX_SELECT`) or when they cross certain power-of-two address boundaries (`M0_TIMING.PAGEBREAK`). [Section 12.14.2.1](#) goes into more detail on these features.

The QMI can map addresses with its built-in address translation hardware: each chip select is partitioned into 4×4 MB windows, whose physical base address and aperture size are configured in units of 4 kB (one flash sector). This enables the runtime addresses of flash programs to be independent of where they are stored: for example, a flash-resident bootloader at flash storage address 0 could select one of multiple flash-resident program images, all of them linked to run at address `0x10000000`, and these can be executed in place with no position-independent code required. Address translation is described fully in [Section 12.14.4](#).

Finally, the direct-mode interface is included for cases where software needs to communicate directly with the external QSPI devices, for example to access status registers. This interface also supports serial, dual, and quad interface widths as described in [Section 12.14.5](#).

12.14.2. QSPI Transfers

A QSPI bus connects one host, such as QMI, to multiple devices, such as a serial NOR flash. It consists of:

- One chip select line per device (**CSn**)
- One shared clock line (**SCK**)
- Up to four shared data lines (**SD0** through **SD3**)

No single specification defines the format of QSPI commands. However, certain de facto command sets exist on most QSPI flash/SRAM/PSRAM devices. QMI supports the most common variations of these commands.

QMI is primarily a memory interface, not a general-purpose QSPI peripheral. Although the direct-mode interface (Section 12.14.5) allows arbitrary QSPI accesses by passing raw data through the FIFOs, QMI is optimised for preformatted read/write transfers in response to AHB read/write bus accesses.

All QSPI read/write accesses performed by the QMI use the following five phases:

1. **Prefix:** An optional, constant 8-bit value that indicates the SPI command being performed (referred to as the **command prefix** or **instruction prefix** in SPI device datasheets)
2. **Address:** A 24-bit byte address that specifies the SPI memory location being read/written, corresponding to the lower 24 bits of the AHB address
3. **Suffix:** An optional, constant 8-bit value which follows the address in certain access modes
4. **Dummy:** 0-value (SPI) or high-impedance (dual/quad-SPI) cycles which precede the data, to provide the SPI device adequate time to access the first address
5. **Data:** Transfers memory contents to/from the SPI device at sequential byte addresses from the initial address indicated in the address phase

The chip select for the addressed device is asserted before the prefix phase, and deasserted at the end of the data phase.

Each phase has a length in bits and interface width (single/dual/quad) configured using **M0_RFMT/M1_RFMT** (for reads) and **M0_WFMT/M1_WFMT** for writes. The **M0/M1** versions of each register configure accesses to memory windows 0 and 1 (the two chip selects) respectively. This allows you to address two different QSPI devices with different command formats transparently.

Figure 106. An example serial read. After an 8-bit prefix, the host sends 24 address bits, and the device replies with data starting from the next cycle.

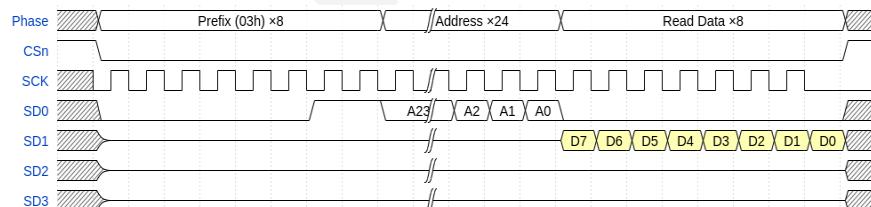


Figure 106 illustrates the **03h** serial read command. This section refers to a handful of common QSPI read/write commands used by QSPI flash/SRAM/PSRAM devices; refer to a QSPI device datasheet for command details. For example, the [W25Q16JV datasheet available from Winbond](#) provides descriptions of all of the read commands mentioned in this section.

Applying the five-phase structure introduced previously, the **03h** QSPI transaction breaks down as follows:

1. 8-bit prefix, at serial width (prefix = **0x03**)
2. 24-bit address, at serial width
3. No suffix (length 0)
4. No dummy bits (length 0)
5. Data bits, at serial width

The number of address bits is fixed at 24 for all QMI accesses. The number of data bits depends on the size of the transfer: this diagram shows 8 bits being transferred, which corresponds to an uncached byte read from the processor.

The **M0_RFMT/M1_RFMT** registers configure all other parameters used for the data phase, such as serial interface width.

The four data lines **SD3** through **SD0** make up the QSPI bus. At serial width, the host drives data out on **SD0**, and the device responds with data travelling in the opposite direction on **SD1**. **SD3** and **SD2** are undriven during serial-SPI and dual-SPI width parts of a transfer, and are usually pulled high. The shaded background behind the **D7** through **D0** data bits indicates that the transfer direction is device-to-host. Higher interface widths use the **SDx** lines bidirectionally.

Figure 107. The **0Bh** read command adds 8 dummy cycles between address and data, to permit higher bus frequencies.

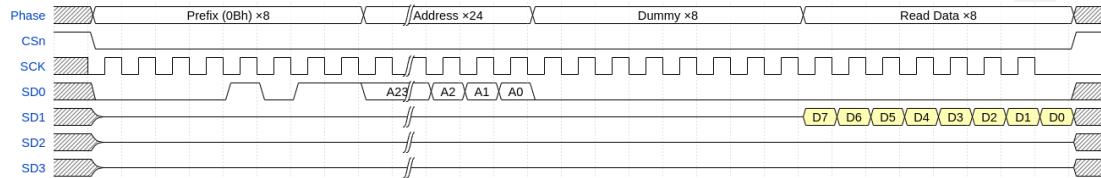


Figure 107 shows the **0Bh** serial read command, a common variation on the **03h**. **0Bh** adds dummy cycles between the address and data phases, which helps hide the initial access latency of the storage array inside of the QSPI device. This allows higher operating frequencies.

Applying the five-phase structure introduced previously, the **0Bh** QSPI transaction breaks down as follows:

1. 8-bit prefix, at serial width (prefix = **0x0b**)
2. 24-bit address, at serial width
3. No suffix (length 0)
4. Eight dummy bits, at serial width
5. Data bits, at serial width

At serial width, the QMI continues to drive the **SD0** line low throughout the dummy phase, as this line is unidirectional at this width. At dual-SPI and quad-SPI width, **SD0** is tristated during the dummy phase along with **SD1** through **SD3**.

QMI idles its clock low between transfers, expecting data to be captured on the leading edge of each clock pulse (i.e. the rising edge). In legacy Motorola SPI terms, the clock polarity is 0 and the clock phase is 0. Other clock polarities and phases are not supported. To ensure data is stable across the rising edge, new data is launched on each *falling* edge.

When transfer chaining is disabled (Section 12.14.2.1), QMI takes advantage of this clock behaviour by suppressing the final clock pulse on reads. This saves energy by avoiding unnecessary **SCK** transitions, and by not inadvertently requesting the data that immediately follows the requested data. QMI still leaves one full **SCK** period where the last data is valid, and still captures at the point the **SCK** rising edge would be launched (Section 12.14.3), but the actual **SCK** clock pulse is suppressed.

Figure 108. An **EBh** quad I/O read command. The command prefix is serial, but address and data are 4 bits per cycle.

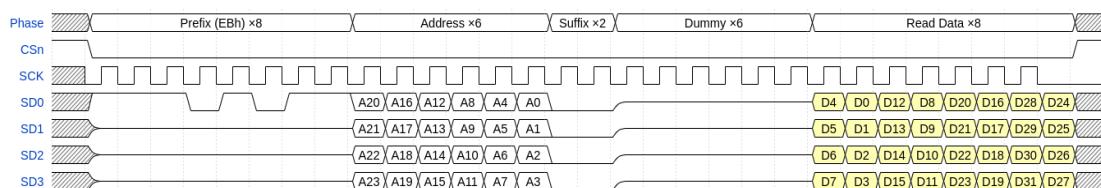


Figure 108 shows a quad-width read transfer. In this example, the command prefix is still transferred at serial width, but the full quad-width is used thereafter, as the prefix identifies the width of the access.

Applying the five-phase structure introduced previously, the QSPI transaction breaks down as follows:

1. 8-bit prefix, at serial width (prefix = **0xeb**)
2. 24-bit address, at quad width
3. 8-bit suffix, at quad width (suffix = **0x00**)
4. 24 dummy bits, at quad width
5. Data bits, at quad width

The suffix is an extension of the command prefix, placed after the address bits to avoid extending the initial access

latency. The bit patterns used for prefixes and suffixes are configured using the [M0_RCMD/M1_RCMD](#) registers (for reads) and [M0_WCMD/M1_WCMD](#) registers (for writes). One common use of the suffix on EBh quad I/O read commands is to enter a so-called continuous read mode, where the prefix of the next command is skipped (assumed to be the same as the current command) to reduce the number of cycles required for the next read access.

Figure 109. An 02h write transfer, shown with the device in QPI mode (4 bits per cycle for all transfers)

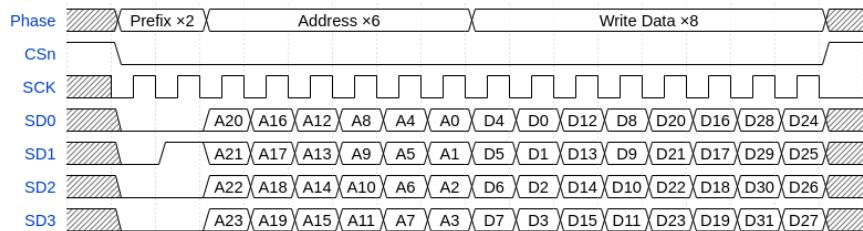


Figure 109 shows a write command at quad width. In this example, the command prefix is also issued in quad mode, which is common for QSPI RAM. Since read and write commands mix freely, dropping the prefix (like flash continuous read mode) is less useful, so QSPI RAM devices often support a **QPI mode** that also issues command prefixes in quad width to reduce per-access cost.

Applying the five-phase structure introduced previously, the QSPI transaction breaks down as follows:

1. 8-bit prefix, at quad width (prefix = `0x02`)
2. 24-bit address, at quad width
3. No suffix (0 bits)
4. No dummy bits
5. Data bits, at quad width

It is worth noting the bit and byte order in this diagram. SPI is conventionally MSB-first within each byte. When multiple bits transfer each cycle (using the `SD0`, `SD1`, `SD2` and `SD3` data lines in parallel), higher-numbered data lines carry more-significant bits. The first cycle of the data transfer in Figure 109 transfers the four most-significant bits of the first byte of data. The most-significant bit (bit 7) transfers on `SD3`, and the least-significant of these bits (bit 4) transfers on `SD0`.

Since RP2350 is a little-endian system, higher byte addresses correspond to higher numerical significance. Figure Figure 109 shows the transfer of a 32-bit value spanning four consecutive byte addresses, starting at the initial address transmitted by the host during the address phase. The first two cycles of the data phase transfer the first byte, containing the 8 least-significant bits of the 32-bit value. The last two cycles of the data phase transfer the last byte, containing the 8 most-significant bits of the 32-bit value (bits 31 through 24, inclusive).

12.14.2.1. Transfer Chaining

Referring back to Figure 108, which shows a 32-bit QSPI read with an EBh serial prefix, it's evident that more time is spent issuing the prefix and address (14 cycles) and waiting for the initial read latency (an additional 8 cycles), than actually transferring the data (8 cycles). This overhead leaves only a small fraction of the theoretical maximum QSPI throughput available for transferring data from flash, which limits the performance of direct code execution.

Figure 110. An EBh read, without the command prefix. The suffix is used to indicate the lack of prefix on the next command.

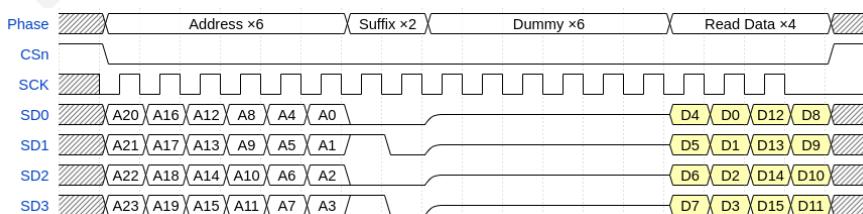
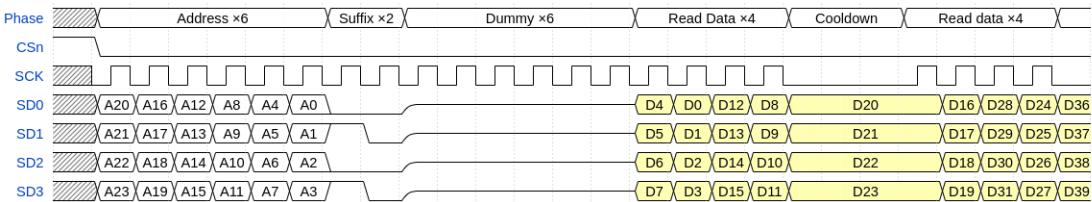


Figure 110 shows how this can be improved by **continuous read mode**, which uses a suffix (here `0xa0`) to indicate the lack of command prefix on the next command. This example only transfers 16 bits of data (e.g. an uncached halfword read by the processor). Suffixes are effectively free to transfer, because they are transferred during the latency wait period between the address being issued and the first data returned from the QSPI device's internal storage. However, this still leaves the majority of the QSPI bus cycles spent issuing addresses and waiting, not transferring data.

Consequently, QSPI memory's random-access performance is much lower than its sequential-access performance.

Figure 111. An EBh read, with a subsequent sequential read chained onto the next transfer



QMI's transfer chaining feature exploits the difference between sequential and non-sequential access speed. Figure 111 shows two sequentially-addressed halfword reads (i.e. the address of the second transfer is two plus the address of the first transfer), with `M0_TIMING.COOLDOWN/M1_TIMING.COOLDOWN` set to a non-zero value.

In Figure 110, QMI suppressed the last clock pulse and immediately released the chip select after the last data transferred. When transfer chaining is enabled, as in Figure 111, QMI does *not* suppress the last clock pulse, instead keeping the chip select asserted. It remains in this state for a certain amount of time, configured by the `COOLDOWN` register field, waiting for another transfer. QMI then executes the next transfer by appending more clocks to the current transfer. The chip select remains asserted throughout instead of releasing and reasserting between commands. To benefit from transfer chaining, the next transfer must meet the following criteria:

- same direction as the previous transfer (read/write)
- address sequential to the previous transfer (equal to previous address plus previous size)
- address in the same window as the previous transfer (same chip select)
- previous transfer did not reach a page break boundary (configured by `M0_TIMING.PAGEBREAK/M1_TIMING.PAGEBREAK`)

This considerably improves throughput for long uncached linear transfers such as using the XIP stream peripheral (Section 4.4.3) or executing cold code sequences which tend to miss the cache many times sequentially.

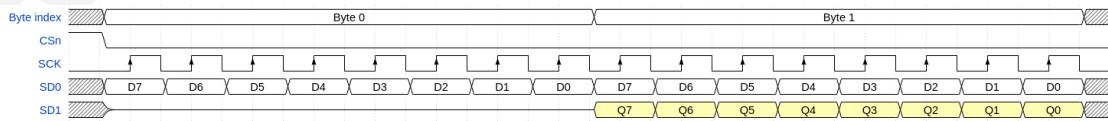
This can continue for arbitrarily many transfers. It is possible to read the entire contents of a typical flash device using transfer chaining from a single address.

Note that the transfer chaining feature can slightly degrade random access performance. If the next transfer is non-sequential, the chip select must be deasserted, possibly dwell high for some minimum period (depending on timing requirements of the QSPI device), and then be reasserted to issue the new address. If transfer chaining were not used, the chip select would have deasserted immediately following the end of the previous transfer, avoiding some of this delay. This can be mitigated by tuning the `COOLDOWN` timer register parameter to avoid leaving the chip select asserted for excessively long periods, since sequential transfers are usually tightly grouped in time.

12.14.3. Timing

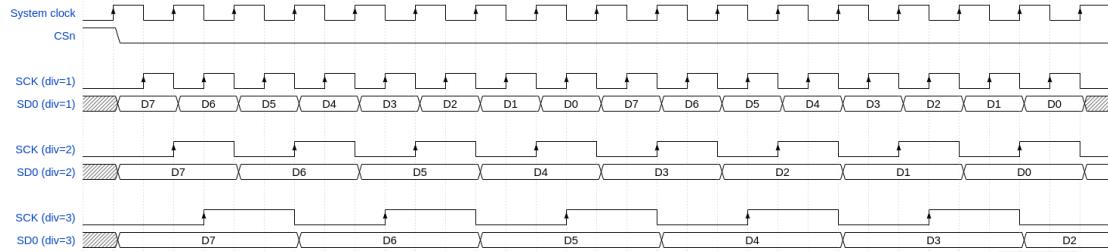
QMI operates in SPI mode 0, capturing data on each rising edge of `SCK`. New data is asserted on each subsequent falling edge. The first output data launches simultaneously with the assertion of the chip select, as illustrated by Figure 112.

Figure 112. A bidirectional SPI transfer, as used by QMI.



QMI timing is relative to the system clock. As this is generally quite fast relative to external signals, the `M0_TIMING.CLKDIV/M1_TIMING.CLKDIV` field can uniformly slow `SCK` and data lines by an integer factor.

Figure 113. The CLKDIV controls set the number of system clock cycles per SCK cycle, for each memory window.



QMI uses DDR input/output registers to enable a resolution of one half system clock cycle for output signal generation and input sampling. This allows QMI to support odd clock divisors, including divide-by-one (`SCK` frequency equal to system clock frequency).

i NOTE

In practice, the maximum `SCK` frequency is constrained by the limits of the attached QSPI device, the signal integrity afforded by the PCB layout, and IO delays in the pads. See [Section 12.14.3.4](#).

12.14.3.1. Input Sampling and RXDELAY

QMI samples input data on the rising edge of `SCK` ([Section 12.14.3](#)). To introduce additional delay to the input delay register (helpful when the round trip delay is longer than half an `SCK` cycle), use `M0_TIMING.RXDELAY`/`M1_TIMING.RXDELAY`. `RXDELAY` counts delay in half system clock cycles, instead of `SCK` cycles.

12.14.3.2. Chip Select Timing

To save power, chip select is deasserted after a transaction completes. To leave chip select asserted after a transaction, use `M0_TIMING.COOLDOWN`/`M1_TIMING.COOLDOWN`. This can reduce latency and increase bus throughput.

Chip select can be asserted one system clock cycle early via `M0_TIMING.SELECT_SETUP`/`M1_TIMING.SELECT_SETUP`. Some flash devices require this setting at very high `SCK` frequencies. Without this setting, QMI asserts chip select one half `SCK` period before the first rising edge of `SCK`. This is simultaneous with the assertion of the first data on `SD0`.

Chip select hold time can also be extended by up to 3 additional system clock cycles via `M0_TIMING.SELECT_HOLD`/`M1_TIMING.SELECT_HOLD`.

To enforce a maximum amount of time that chip select can remain asserted, use `M0_TIMING.MAX_SELECT`/`M1_TIMING.MAX_SELECT`. This is useful for PSRAM devices, which must issue internal DRAM refresh cycles when deselected.

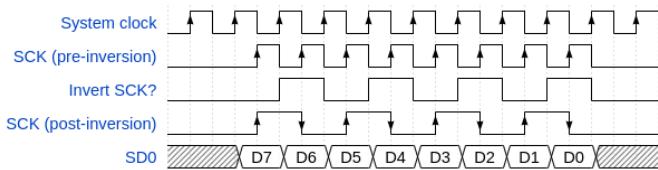
To enforce a minimum amount of time that chip select can remain deasserted, use `M0_TIMING.MIN_DESELECT`/`M1_TIMING.MIN_DESELECT`.

12.14.3.3. Double Transfer Rate (DTR)

Some QSPI memory devices transfer data on both edges of `SCK`. This feature, known as **double transfer rate** (DTR), allows a lower `SCK` frequency for a given data transfer rate, reducing EM emissions and the energy cost of toggling the external clock. To enable DTR mode (per-window and per-direction), set the `M0_RFMT.DTR`/`M1_RFMT.DTR` flag (for reads) or `M0_WFMT.DTR`/`M1_WFMT.DTR` (for writes).

QMI implements DTR by halving the clock frequency whilst maintaining the data rate. To achieve this, QMI *inverts* alternate single transfer rate `SCK` clock periods, transforming a low-high-low-high sequence into a low-high-high-low sequence. When DTR is disabled, the QMI launches data on `SCK` falling edges and captures on rising edges. When DTR is enabled, the QMI launches data at the point half-way in between two `SCK` edges, and captures on each edge, as shown in [Figure 114](#).

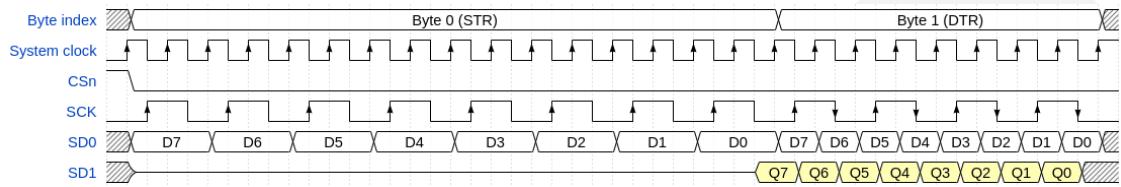
Figure 114. DTR is implemented by halving the SCK frequency whilst maintaining data rate.



Enabling DTR mode does not change the data timing, only the SCK timing. Data is launched at the point where a SCK negative edge would be, had the clock rate not been halved.

When DTR is enabled, the prefix and dummy phase of a transfer remain single transfer rate. In these phases, data bits are doubled to match the half-rate SCK, so that new data is ready in time for each rising edge only. Figure 115 shows the first byte (the command prefix) at single transfer rate and the second byte (address and data) at double transfer rate.

Figure 115. Parts of DTR-enabled transfers are still single transfer rate: effectively each data bit is sent twice.



The arrows on the SCK line in Figure 115 show the active edges of SCK (where data is captured). The single transfer rate portion of the access expects data capture on the rising edge. The double transfer rate portion of the access expects data capture on both edges.

Data travelling from device to host is likewise launched on both edges of SCK. Each time the QMI launches a new clock edge, there is some delay as transitions propagate through the RP2350 pad output delay, QSPI device SCK-to-SD_x delay, and back in through the RP2350 SD_x pad input delays. QMI captures data simultaneously with the launch of the next SCK edge, plus any delay configured by M0_TIMING.RXDELAY/M1_TIMING.RXDELAY. The round-trip delay from SCK output back to SD_x input provides the SD_x input hold time. If the input setup time is not sufficient, you can increase RXDELAY. For more information, see the specific QSPI device datasheet, as well as Section 12.14.3.4.

12.14.3.4. AC Timing Parameters

The QMI interface is timed using the internal system clock. Skew between different QMI pins for inputs or outputs is kept to a minimum. Any additional setup or hold time is supported by using additional clock cycle delays as mentioned in other sections. Skew values vary depending on whether we consider just the dedicated QSPI pins (QSPI_SS, QSPI_SD[3:0], QSPI_SCLK) or include the Bank 0 GPIO XIP special functions (for the additional QMI chip select). Different package options have different skew timing, shown below.

Table 1251. QMI Timing skew

Interface	Typical Skew (ps)	Max Skew (ps)
QSPI input	15	25
QSPI output	100	180
Bank 0 GPIO (QFN-60) output	1080	1725
Bank 0 GPIO (QFN-80) output	1280	2100

It is also useful to know the delay from internal register running on system clock to output pin, and similarly the delay from input pin to the sampling register running on system clock. Table 1252 provides worst case process, voltage, and temperature timings for inputs and outputs on QSPI, and outputs on GPIO. Note that this delay varies based on the VDDIO voltage level as shown in the table.

Table 1252. QMI Timing delay

Path	Max delay (ns) VDDIO=3.3V	Max delay (ns) VDDIO=1.8V
QSPI input to system clock	1.5	1.2
system clock to QSPI output	2.5	3.6

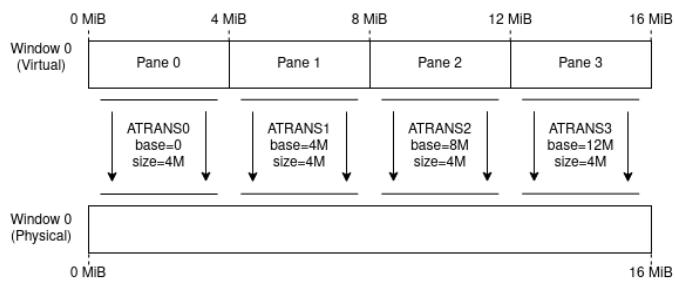
Path	Max delay (ns) $VDDIO=3.3V$	Max delay (ns) $VDDIO=1.8V$
system clock to GPIO (QFN-60) output	3.5	4.9
system clock to GPIO (QFN-80) output	4.1	5.4

12.14.4. Address Translation

QMI applies a configurable mapping from the *virtual* address requested by the processor or DMA to the *physical* address transmitted to the external QSPI device. This is performed separately for each of the 16 MiB chip select windows. You cannot map contents between devices.

Each window is divided into four *panes*, each independently mapped onto the physical address space for that window. The default configuration applied on QMI reset, as shown in [Figure 116](#), is a 1:1 identity mapping between virtual and physical addresses. In this state the address mapping has no effect, and the entire 16 MiB address space of the external QSPI device is mapped directly into the system address space.

[Figure 116](#). By default, each window is set up to map the full 16 MiB virtual address space directly 1:1 with the 16 MiB physical address space.



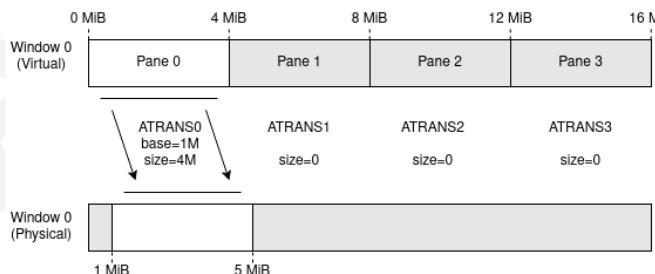
Each pane corresponds to the one of the four `ATRANSx` registers for that window: `ATRANS0` through `ATRANS3` for window 0, and `ATRANS4` through `ATRANS7` for window 1.

The virtual base address of each pane is fixed and assigned in 4 MiB increments. There are two configurable parameters for the mapping of that pane into physical address space:

- **BASE**: defines the physical address corresponding to offset 0 in the virtual address pane. Configured in units of 4 kB (one flash sector), ranging from 0 to (16 MiB minus 4 kB).
- **SIZE**: defines the amount of address space mapped by this pane. Configured in units of 4 kB (one flash sector) ranging from 0 to 4 MiB.

The mapping grows from the start of the pane. A **SIZE** of 1 MiB maps the first 1 MiB of that pane's virtual address range to downstream memory, and the remainder is unmapped. A **SIZE** of 0 means that no address within this virtual address pane is accessible. Accesses beyond the currently configured **SIZE** return a bus error, and do not pass through to the downstream QSPI bus. As a result, they have no effect on the external memory device.

[Figure 117](#). The **BASE** of a pane defines where its physical mapping begins. The **SIZE** defines how far it extends. A **SIZE** of 0 means no addresses are mapped through that pane.



[Figure 117](#) shows an example mapping, where the first 4 MiB of virtual address space for chip select 0 (virtual address offsets `0x000000` through `0x3fffff` inclusive) map to a 4 MiB physical address window starting at a 1 MiB offset (physical address offsets `0x100000` through `0x4fffff` inclusive). This mapping could be used for flash that contains a 1 MiB bootloader application followed by a 4 MiB user application. Ideally, the user application should not be aware of the flash layout defined by the bootloader; that way, the same application can run under different bootloader implementations. The virtual-to-physical mapping solves this problem by making the storage location of the user application (starting at 1 MiB) independent of the address it appears at in the system address space (starting at 0 MiB).

12.14.4.1. Bootrom Support for Address Translation

The bootrom can automatically configure address translation at boot time, so that a binary stored at some arbitrary location in physical flash storage can appear at a runtime flash address of 0.

This is done automatically when the booted image is inside of a flash partition (Section 5.1.2), and can be adjusted manually based on a rolling window delta specified in the [IMAGE_DEF](#) of the launched executable (Section 5.1.1).

The bootrom source code and bootrom documentation often refers to the QMI [ATRANS](#) mapping as "rolling windows", due to the modulo address wrapping on 16 MiB boundaries – see Section 5.1.17.

12.14.4.2. Translation and the XIP Cache

The QMI address translation is performed downstream of the system XIP cache (Section 4.4.1). Therefore, the XIP cache is a *virtual cache* with respect to this translation, because the address translation performed inside QMI is opaque to the XIP cache.

Consequently, changes to the QMI address translation necessitate a flush of the XIP cache. From the cache's point of view, the translation change has moved QMI memory contents around in the cache's downstream address space in a way that is incoherent with the cache contents, so a flush is required to restore coherence. At a minimum, any virtual address whose [ATRANSx](#) register ([ATRANS0](#) through [ATRANS7](#)) has been modified, and which may be allocated in the cache in either the clean or the dirty state, must be flushed. It may be simplest to flush the entire cache.

QMI's address mapping creates another hazard: the same physical address may map to multiple virtual addresses, and therefore may be allocated multiple times in the XIP cache. When you write to a physical address through a cached virtual address alias, the XIP cache does not propagate the change to other aliases. To avoid this issue, do not allow multiple aliases of the same writable physical address at the same instant. Aliasing read-only memory is usually safe. Aliases that exist at different points in time (for example, across an RTOS context switch boundary) can be kept coherent with appropriate cleaning and flushing when the translation is changed.

12.14.5. Direct Mode

In direct mode, the AHB XIP address window is disconnected from the QSPI bus, and the bus is controlled through a Tx/Rx FIFO pair, similar to a normal SPI peripheral. In this state, the XIP window becomes inaccessible. Attempting to access it generates a bus fault. This mode is used for low-level access to the QSPI bus, for example when issuing flash erase/programming commands, or when accessing QSPI device status registers.

All direct-mode operation is controlled through [DIRECT_CSR](#), with data being exchanged through [DIRECT_TX](#) and [DIRECT_RX](#). To enable direct mode, first set [DIRECT_CSR.EN](#), and then poll for [DIRECT_CSR.BUSY](#) to go low to ensure that any in-progress XIP transfer at the point direct mode was enabled has completed.

Direct mode has its own clock divisor and RX sampling delay, configured by [DIRECT_CSR.CLKDIV](#) and [DIRECT_CSR.RXDELAY](#). These are separate from the per-window settings configured in [M0_TIMING/M1_TIMING](#), because serial commands used for control purposes may have different frequency limits than data accesses used for execute-in-place.

For each push to [DIRECT_TX](#), QMI will issue 8 or 16 bits of FIFO data to the QSPI bus. Optionally, the same number of bits are simultaneously sampled and returned in [DIRECT_RX](#). The clock is initially low, and data is always captured on the rising edge of [SCK](#), transitioning on the subsequent falling edge.

After pushing to [DIRECT_TX](#), [DIRECT_CSR.BUSY](#) will go high, and remain high until all direct-mode activity has completed. This works even if no Rx data is returned, so is more reliable than polling the RX FIFO status. The [BUSY](#) flag stays high for half an [SCK](#) period after the transfer finishes, to ensure safe chip select timing when this is used to drive the chip selects – see Section 12.14.5.2.

QMI will never push to a full Rx FIFO, or drop data as a result of the FIFO being full – instead, the interface is paused until the system pops [DIRECT_RX](#). This avoids a common trap of Rx data being lost when the processor is heavily interrupted during direct-mode operation, but software must take care not to poll for [DIRECT_CSR.BUSY](#) low without

also checking the Rx FIFO, as this can cause a deadlock when the FIFO fills.

12.14.5.1. Controls in DIRECT_TX

The Tx FIFO carries control information as well as data, with data in the 16 LSBs, and control information in the immediately more-significant bits:

- `DIRECT_TX.NOPUSH` inhibits the `DIRECT_RX` push which would match this Tx data. This avoids creating garbage when pushing control/address information at the start of a transfer.
- `DIRECT_TX.DWIDTH` is the data width of this FIFO record. 0 means the 8 LSBs contain data, and 1 means the 16 LSBs contain data. This also determines the amount of data returned in the matching `DIRECT_RX` entry.
- `DIRECT_TX.IWIDTH` is the interface width (single/dual/quad) used to clock out this FIFO record. The corresponding RX data is sampled at the same width.
- `DIRECT_TX.OE` controls the pad direction for bidirectional transfers. It is ignored for serial `IWIDTH`, since `SD0` is always an output and `SD1` always an input. At dual/quad width, it must be set in order to enable the output drivers for the duration of this FIFO record. The TX data is don't-care when `IWIDTH` is dual/quad and `OE` is not set.

The default when all control bits are zero is an 8-bit serial transfer, with 8 bits of sampled data returned. Therefore, you can ignore the control bits and treat this as a plain 8-bit data FIFO.

12.14.5.2. Chip Select Control

There are two options for driving the chip selects, both via `DIRECT_CSR`:

- `DIRECT_CSR ASSERT_CS0N` and `DIRECT_CSR ASSERT_CS1N` will *immediately* drive the corresponding chip select low when set
- `DIRECT_CSR AUTO_CS0N` and `DIRECT_CSR AUTO_CS1N` configure the corresponding chip select to be set low whenever the interface is busy, i.e. when the `DIRECT_CSR BUSY` flag is high due to a previous `DIRECT_TX` push

! IMPORTANT

The `ASSERT_CSxN` fields assert the chip select *unconditionally*, including when `DIRECT_CSR EN` is clear. Software must take care not to set these fields when XIP transfers may be active.

12.14.6. List of Registers

The QMI control registers start at address `0x400d0000`, defined as `XIP_QMI_BASE` in the SDK.

Table 1253. List of QMI registers

Offset	Name	Info
0x00	<code>DIRECT_CSR</code>	Control and status for direct serial mode Direct serial mode allows the processor to send and receive raw serial frames, for programming, configuration and control of the external memory devices. Only SPI mode 0 (CPOL=0 CPHA=0) is supported.
0x04	<code>DIRECT_TX</code>	Transmit FIFO for direct mode
0x08	<code>DIRECT_RX</code>	Receive FIFO for direct mode
0x0c	<code>M0_TIMING</code>	Timing configuration register for memory address window 0.
0x10	<code>M0_RFMT</code>	Read transfer format configuration for memory address window 0.

Offset	Name	Info
0x14	M0_RCMD	Command constants used for reads from memory address window 0.
0x18	M0_WFMT	Write transfer format configuration for memory address window 0.
0x1c	M0_WCMD	Command constants used for writes to memory address window 0.
0x20	M1_TIMING	Timing configuration register for memory address window 1.
0x24	M1_RFMT	Read transfer format configuration for memory address window 1.
0x28	M1_RCMD	Command constants used for reads from memory address window 1.
0x2c	M1_WFMT	Write transfer format configuration for memory address window 1.
0x30	M1_WCMD	Command constants used for writes to memory address window 1.
0x34	ATRANS0	Configure address translation for XIP virtual addresses 0x000000 through 0x3fffff (a 4 MiB window starting at +0 MiB).
0x38	ATRANS1	Configure address translation for XIP virtual addresses 0x400000 through 0x7fffff (a 4 MiB window starting at +4 MiB).
0x3c	ATRANS2	Configure address translation for XIP virtual addresses 0x800000 through 0xbfffff (a 4 MiB window starting at +8 MiB).
0x40	ATRANS3	Configure address translation for XIP virtual addresses 0xc00000 through 0xfffffff (a 4 MiB window starting at +12 MiB).
0x44	ATRANS4	Configure address translation for XIP virtual addresses 0x1000000 through 0x13ffff (a 4 MiB window starting at +16 MiB).
0x48	ATRANS5	Configure address translation for XIP virtual addresses 0x1400000 through 0x17ffff (a 4 MiB window starting at +20 MiB).
0x4c	ATRANS6	Configure address translation for XIP virtual addresses 0x1800000 through 0x1bffff (a 4 MiB window starting at +24 MiB).
0x50	ATRANS7	Configure address translation for XIP virtual addresses 0x1c00000 through 0x1fffff (a 4 MiB window starting at +28 MiB).

QMI: DIRECT_CSR Register

Offset: 0x00

Description

Control and status for direct serial mode

Direct serial mode allows the processor to send and receive raw serial frames, for programming, configuration and control of the external memory devices. Only SPI mode 0 (CPOL=0 CPHA=0) is supported.

Table 1254.
DIRECT_CSR Register

Bits	Name	Description	Type	Reset
31:30	RXDELAY	Delay the read data sample timing, in units of one half of a system clock cycle. (Not necessarily half of an SCK cycle.)	RW	0x0
29:22	CLKDIV	<p>Clock divisor for direct serial mode. Divisors of 1..255 are encoded directly, and the maximum divisor of 256 is encoded by a value of CLKDIV=0.</p> <p>The clock divisor can be changed on-the-fly by software, without halting or otherwise coordinating with the serial interface. The serial interface will sample the latest clock divisor each time it begins the transmission of a new byte.</p>	RW	0x06
21	Reserved.	-	-	-
20:18	RXLEVEL	Current level of DIRECT_RX FIFO	RO	0x0
17	RXFULL	When 1, the DIRECT_RX FIFO is currently full. The serial interface will be stalled until data is popped; the interface will not begin a new serial frame when the DIRECT_TX FIFO is empty or the DIRECT_RX FIFO is full.	RO	0x0
16	RXEMPTY	When 1, the DIRECT_RX FIFO is currently empty. If the processor attempts to read more data, the FIFO state is not affected, but the value returned to the processor is undefined.	RO	0x0
15	Reserved.	-	-	-
14:12	TXLEVEL	Current level of DIRECT_TX FIFO	RO	0x0
11	TXEMPTY	When 1, the DIRECT_TX FIFO is currently empty. Unless the processor pushes more data, transmission will stop and BUSY will go low once the current 8-bit serial frame completes.	RO	0x0
10	TXFULL	When 1, the DIRECT_TX FIFO is currently full. If the processor tries to write more data, that data will be ignored.	RO	0x0
9:8	Reserved.	-	-	-
7	AUTO_CS1N	When 1, automatically assert the CS1n chip select line whenever the BUSY flag is set.	RW	0x0
6	AUTO_CS0N	When 1, automatically assert the CS0n chip select line whenever the BUSY flag is set.	RW	0x0
5:4	Reserved.	-	-	-
3	ASSERT_CS1N	<p>When 1, assert (i.e. drive low) the CS1n chip select line.</p> <p>Note that this applies even when DIRECT_CSR_EN is 0.</p>	RW	0x0
2	ASSERT_CS0N	<p>When 1, assert (i.e. drive low) the CS0n chip select line.</p> <p>Note that this applies even when DIRECT_CSR_EN is 0.</p>	RW	0x0

Bits	Name	Description	Type	Reset
1	BUSY	<p>Direct mode busy flag. If 1, data is currently being shifted in/out (or would be if the interface were not stalled on the RX FIFO), and the chip select must not yet be deasserted.</p> <p>The busy flag will also be set to 1 if a memory-mapped transfer is still in progress when direct mode is enabled. Direct mode blocks new memory-mapped transfers, but can't halt a transfer that is already in progress. If there is a chance that memory-mapped transfers may be in progress, the busy flag should be polled for 0 before asserting the chip select.</p> <p>(In practice you will usually discover this timing condition through other means, because any subsequent memory-mapped transfers when direct mode is enabled will return bus errors, which are difficult to ignore.)</p>	RO	0x0
0	EN	<p>Enable direct mode.</p> <p>In direct mode, software controls the chip select lines, and can perform direct SPI transfers by pushing data to the DIRECT_TX FIFO, and popping the same amount of data from the DIRECT_RX FIFO.</p> <p>Memory-mapped accesses will generate bus errors when direct serial mode is enabled.</p>	RW	0x0

QMI: DIRECT_TX Register

Offset: 0x04

Description

Transmit FIFO for direct mode

Table 1255.
DIRECT_TX Register

Bits	Name	Description	Type	Reset
31:21	Reserved.	-	-	-
20	NOPUSH	<p>Inhibit the RX FIFO push that would correspond to this TX FIFO entry.</p> <p>Useful to avoid garbage appearing in the RX FIFO when pushing the command at the beginning of a SPI transfer.</p>	WF	0x0
19	OE	<p>Output enable (active-high). For single width (SPI), this field is ignored, and SD0 is always set to output, with SD1 always set to input.</p> <p>For dual and quad width (DSPI/QSPI), this sets whether the relevant SDx pads are set to output whilst transferring this FIFO record. In this case the command/address should have OE set, and the data transfer should have OE set or clear depending on the direction of the transfer.</p>	WF	0x0

Bits	Name	Description	Type	Reset
18	DWIDTH	Data width. If 0, hardware will transmit the 8 LSBs of the DIRECT_TX DATA field, and return an 8-bit value in the 8 LSBs of DIRECT_RX. If 1, the full 16-bit width is used. 8-bit and 16-bit transfers can be mixed freely.	WF	0x0
17:16	IWIDTH	Configure whether this FIFO record is transferred with single/dual/quad interface width (0/1/2). Different widths can be mixed freely. 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	WF	0x0
15:0	DATA	Data pushed here will be clocked out falling edges of SCK (or before the very first rising edge of SCK, if this is the first pulse). For each byte clocked out, the interface will simultaneously sample one byte, on rising edges of SCK, and push this to the DIRECT_RX FIFO. For 16-bit data, the least-significant byte is transmitted first.	WF	0x0000

QMI: DIRECT_RX Register

Offset: 0x08

Description

Receive FIFO for direct mode

Table 1256.
DIRECT_RX Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	With each byte clocked out on the serial interface, one byte will simultaneously be clocked in, and will appear in this FIFO. The serial interface will stall when this FIFO is full, to avoid dropping data. When 16-bit data is pushed into the TX FIFO, the corresponding RX FIFO push will also contain 16 bits of data. The least-significant byte is the first one received.	RF	0x0000

QMI: M0_TIMING, M1_TIMING Registers

Offsets: 0x0c, 0x20

Description

Timing configuration register for memory address window 0/1.

Table 1257.
M0_TIMING,
M1_TIMING Registers

Bits	Name	Description	Type	Reset
31:30	COOLDOWN	<p>Chip select cooldown period. When a memory transfer finishes, the chip select remains asserted for $64 \times$ COOLDOWN system clock cycles, plus half an SCK clock period (rounded up for odd SCK divisors). After this cooldown expires, the chip select is always deasserted to save power.</p> <p>If the next memory access arrives within the cooldown period, the QMI may be able to append more SCK cycles to the currently ongoing SPI transfer, rather than starting a new transfer. This reduces access latency and increases bus throughput.</p> <p>Specifically, the next access must be in the same direction (read/write), access the same memory window (chip select 0/1), and follow sequentially the address of the last transfer. If any of these are false, the new access will first deassert the chip select, then begin a new transfer.</p> <p>If COOLDOWN is 0, the address alignment configured by PAGEBREAK has been reached, or the total chip select assertion limit MAX_SELECT has been reached, the cooldown period is skipped, and the chip select will always be deasserted one half SCK period after the transfer finishes.</p>	RW	0x1
29:28	PAGEBREAK	<p>When page break is enabled, chip select will automatically deassert when crossing certain power-of-2-aligned address boundaries. The next access will always begin a new read/write SPI burst, even if the address of the next access follows in sequence with the last access before the page boundary.</p> <p>Some flash and PSRAM devices forbid crossing page boundaries with a single read/write transfer, or restrict the operating frequency for transfers that do cross page a boundary. This option allows the QMI to safely support those devices.</p> <p>This field has no effect when COOLDOWN is disabled.</p> <p>0x0 → No page boundary is enforced 0x1 → Break bursts crossing a 256-byte page boundary 0x2 → Break bursts crossing a 1024-byte quad-page boundary 0x3 → Break bursts crossing a 4096-byte sector boundary</p>	RW	0x0
27:26	Reserved.	-	-	-
25	SELECT_SETUP	<p>Add up to one additional system clock cycle of setup between chip select assertion and the first rising edge of SCK.</p> <p>The default setup time is one half SCK period, which is usually sufficient except for very high SCK frequencies with some flash devices.</p>	RW	0x0

Bits	Name	Description	Type	Reset
24:23	SELECT_HOLD	<p>Add up to three additional system clock cycles of active hold between the last falling edge of SCK and the deassertion of this window's chip select.</p> <p>The default hold time is one system clock cycle. Note that flash datasheets usually give chip select active hold time from the last rising edge of SCK, and so even zero hold from the last falling edge would be safe.</p> <p>Note that this is a minimum hold time guaranteed by the QMI: the actual chip select active hold may be slightly longer for read transfers with low clock divisors and/or high sample delays. Specifically, if the point two cycles after the last RX data sample is later than the last SCK falling edge, then the hold time is measured from this point.</p> <p>Note also that, in case the final SCK pulse is masked to save energy (true for non-DTR reads when COOLDOWN is disabled or PAGE_BREAK is reached), all of QMI's timing logic behaves as though the clock pulse were still present. The SELECT_HOLD time is applied from the point where the last SCK falling edge would be if the clock pulse were not masked.</p>	RW	0x0
22:17	MAX_SELECT	<p>Enforce a maximum assertion duration for this window's chip select, in units of 64 system clock cycles. If 0, the QMI is permitted to keep the chip select asserted indefinitely when servicing sequential memory accesses (see COOLDOWN).</p> <p>This feature is required to meet timing constraints of PSRAM devices, which specify a maximum chip select assertion so they can perform DRAM refresh cycles. See also MIN_DESELECT, which can enforce a minimum deselect time.</p> <p>If a memory access is in progress at the time MAX_SELECT is reached, the QMI will wait for the access to complete before deasserting the chip select. This additional time must be accounted for to calculate a safe MAX_SELECT value. In the worst case, this may be a fully-formed serial transfer, including command prefix and address, with a data payload as large as one cache line.</p>	RW	0x00
16:12	MIN_DESELECT	<p>After this window's chip select is deasserted, it remains deasserted for half an SCK cycle (rounded up to an integer number of system clock cycles), plus MIN_DESELECT additional system clock cycles, before the QMI reasserts either chip select pin.</p> <p>Nonzero values may be required for PSRAM devices which enforce a longer minimum CS deselect time, so that they can perform internal DRAM refresh cycles whilst deselected.</p>	RW	0x00

Bits	Name	Description	Type	Reset
11	Reserved.	-	-	-
10:8	RXDELAY	<p>Delay the read data sample timing, in units of one half of a system clock cycle. (Not necessarily half of an SCK cycle.) An RXDELAY of 0 means the sample is captured at the SDI input registers simultaneously with the rising edge of SCK launched from the SCK output register.</p> <p>At higher SCK frequencies, RXDELAY may need to be increased to account for the round trip delay of the pads, and the clock-to-Q delay of the QSPI memory device.</p>	RW	0x0
7:0	CLKDIV	<p>Clock divisor. Odd and even divisors are supported. Defines the SCK clock period in units of 1 system clock cycle. Divisors 1..255 are encoded directly, and a divisor of 256 is encoded with a value of CLKDIV=0.</p> <p>The clock divisor can be changed on-the-fly, even when the QMI is currently accessing memory in this address window. All other parameters must only be changed when the QMI is idle.</p> <p>If software is increasing CLKDIV in anticipation of an increase in the system clock frequency, a dummy access to either memory window (and appropriate processor barriers/fences) must be inserted after the Mx_TIMING write to ensure the SCK divisor change is in effect <i>before</i> the system clock is changed.</p>	RW	0x04

QMI: M0_RFMT, M1_RFMT Registers

Offsets: 0x10, 0x24

Description

Read transfer format configuration for memory address window 0/1.

Configure the bus width of each transfer phase individually, and configure the length or presence of the command prefix, command suffix and dummy/turnaround transfer phases. Only 24-bit addresses are supported.

The reset value of the Mx_RFMT register is configured to support a basic 03h serial read transfer with no additional configuration.

Table 1258.
M0_RFMT, M1_RFMT
Registers

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	DTR	<p>Enable double transfer rate (DTR) for read commands: address, suffix and read data phases are active on both edges of SCK. SDO data is launched centre-aligned on each SCK edge, and SDI data is captured on the SCK edge that follows its launch.</p> <p>DTR is implemented by halving the clock rate; SCK has a period of 2 x CLK_DIV throughout the transfer. The prefix and dummy phases are still single transfer rate.</p> <p>If the suffix is quad-width, it must be 0 or 8 bits in length, to ensure an even number of SCK edges.</p>	RW	0x0

Bits	Name	Description	Type	Reset
27:19	Reserved.	-	-	-
18:16	DUMMY_LEN	Length of dummy phase between command suffix and data phase, in units of 4 bits. (i.e. 1 cycle for quad width, 2 for dual, 4 for single) 0x0 → No dummy phase 0x1 → 4 dummy bits 0x2 → 8 dummy bits 0x3 → 12 dummy bits 0x4 → 16 dummy bits 0x5 → 20 dummy bits 0x6 → 24 dummy bits 0x7 → 28 dummy bits	RW	0x0
15:14	SUFFIX_LEN	Length of post-address command suffix, in units of 4 bits. (i.e. 1 cycle for quad width, 2 for dual, 4 for single) Only values of 0 and 8 bits are supported. 0x0 → No suffix 0x2 → 8-bit suffix	RW	0x0
13	Reserved.	-	-	-
12	PREFIX_LEN	Length of command prefix, in units of 8 bits. (i.e. 2 cycles for quad width, 4 for dual, 8 for single) 0x0 → No prefix 0x1 → 8-bit prefix	RW	0x1
11:10	Reserved.	-	-	-
9:8	DATA_WIDTH	The width used for the data transfer 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0
7:6	DUMMY_WIDTH	The width used for the dummy phase, if any. If width is single, SD0/MOSI is held asserted low during the dummy phase, and SD1...SD3 are tristated. If width is dual/quad, all IOs are tristated during the dummy phase. 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0
5:4	SUFFIX_WIDTH	The width used for the post-address command suffix, if any 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0
3:2	ADDR_WIDTH	The transfer width used for the address. The address phase always transfers 24 bits in total. 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0

Bits	Name	Description	Type	Reset
1:0	PREFIX_WIDTH	The transfer width used for the command prefix, if any 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0

QMI: M0_RCMD, M1_RCMD Registers

Offsets: 0x14, 0x28

Description

Command constants used for reads from memory address window 0/1.

The reset value of the Mx_RCMD register is configured to support a basic 03h serial read transfer with no additional configuration.

Table 1259.
M0_RCMD, M1_RCMD
Registers

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:8	SUFFIX	The command suffix bits following the address, if Mx_RFMT_SUFFIX_LEN is nonzero.	RW	0xa0
7:0	PREFIX	The command prefix bits to prepend on each new transfer, if Mx_RFMT_PREFIX_LEN is nonzero.	RW	0x03

QMI: M0_WFMT, M1_WFMT Registers

Offsets: 0x18, 0x2c

Description

Write transfer format configuration for memory address window 0/1.

Configure the bus width of each transfer phase individually, and configure the length or presence of the command prefix, command suffix and dummy/turnaround transfer phases. Only 24-bit addresses are supported.

The reset value of the Mx_WFMT register is configured to support a basic 02h serial write transfer. However, writes to this window must first be enabled via the XIP_CTRL_WRITABLE_Mx bit for this window, as XIP memory is read-only by default.

Table 1260.
M0_WFMT, M1_WFMT
Registers

Bits	Name	Description	Type	Reset
31:29	Reserved.	-	-	-
28	DTR	Enable double transfer rate (DTR) for write commands: address, suffix and write data phases are active on both edges of SCK. SDO data is launched centre-aligned on each SCK edge, and SDI data is captured on the SCK edge that follows its launch. DTR is implemented by halving the clock rate; SCK has a period of 2 x CLK_DIV throughout the transfer. The prefix and dummy phases are still single transfer rate. If the suffix is quad-width, it must be 0 or 8 bits in length, to ensure an even number of SCK edges.	RW	0x0
27:19	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
18:16	DUMMY_LEN	Length of dummy phase between command suffix and data phase, in units of 4 bits. (i.e. 1 cycle for quad width, 2 for dual, 4 for single) 0x0 → No dummy phase 0x1 → 4 dummy bits 0x2 → 8 dummy bits 0x3 → 12 dummy bits 0x4 → 16 dummy bits 0x5 → 20 dummy bits 0x6 → 24 dummy bits 0x7 → 28 dummy bits	RW	0x0
15:14	SUFFIX_LEN	Length of post-address command suffix, in units of 4 bits. (i.e. 1 cycle for quad width, 2 for dual, 4 for single) Only values of 0 and 8 bits are supported. 0x0 → No suffix 0x2 → 8-bit suffix	RW	0x0
13	Reserved.	-	-	-
12	PREFIX_LEN	Length of command prefix, in units of 8 bits. (i.e. 2 cycles for quad width, 4 for dual, 8 for single) 0x0 → No prefix 0x1 → 8-bit prefix	RW	0x1
11:10	Reserved.	-	-	-
9:8	DATA_WIDTH	The width used for the data transfer 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0
7:6	DUMMY_WIDTH	The width used for the dummy phase, if any. If width is single, SD0/MOSI is held asserted low during the dummy phase, and SD1...SD3 are tristated. If width is dual/quad, all IOs are tristated during the dummy phase. 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0
5:4	SUFFIX_WIDTH	The width used for the post-address command suffix, if any 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0
3:2	ADDR_WIDTH	The transfer width used for the address. The address phase always transfers 24 bits in total. 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0
1:0	PREFIX_WIDTH	The transfer width used for the command prefix, if any 0x0 → Single width 0x1 → Dual width 0x2 → Quad width	RW	0x0

QMI: M0_WCMD, M1_WCMD Registers

Offsets: 0x1c, 0x30

Description

Command constants used for writes to memory address window 0/1.

The reset value of the Mx_WCMD register is configured to support a basic 02h serial write transfer with no additional configuration.

Table 1261.
M0_WCMD,
M1_WCMD Registers

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:8	SUFFIX	The command suffix bits following the address, if Mx_WFMT_SUFFIX_LEN is nonzero.	RW	0xa0
7:0	PREFIX	The command prefix bits to prepend on each new transfer, if Mx_WFMT_PREFIX_LEN is nonzero.	RW	0x02

QMI: ATRANS0, ATRANS4 Registers

Offsets: 0x34, 0x44

Description

Configure address translation for a 4 MiB window of XIP virtual addresses starting at $n \times 4$ MiB.

Address translation allows a program image to be executed in place at multiple physical flash addresses (for example, a double-buffered flash image for over-the-air updates), without the overhead of position-independent code.

At reset, the address translation registers are initialised to an identity mapping, so that they can be ignored if address translation is not required.

Note that the XIP cache is fully virtually addressed, so a cache flush is required after changing the address translation.

Table 1262. ATRANS0,
ATRANS4 Registers

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26:16	SIZE	Translation aperture size for this virtual address range, in units of 4 kiB (one flash sector). Bits 21:12 of the virtual address are compared to SIZE. Offsets greater than SIZE return a bus error, and do not cause a QSPI access.	RW	0x400
15:12	Reserved.	-	-	-
11:0	BASE	Physical address base for this virtual address range, in units of 4 kiB (one flash sector). Taking a 24-bit virtual address, firstly bits 23:22 (the two MSBs) are masked to zero, and then BASE is added to bits 23:12 (the upper 12 bits) to form the physical address. Translation wraps on a 16 MiB boundary.	RW	0x000

QMI: ATRANS1, ATRANS5 Registers

Offsets: 0x38, 0x48

Description

Configure address translation for XIP virtual addresses 0x400000 through 0x7fffff (a 4 MiB window starting at +4 MiB).

Address translation allows a program image to be executed in place at multiple physical flash addresses (for example, a double-buffered flash image for over-the-air updates), without the overhead of position-independent code.

At reset, the address translation registers are initialised to an identity mapping, so that they can be ignored if address translation is not required.

Note that the XIP cache is fully virtually addressed, so a cache flush is required after changing the address translation.

Table 1263. ATRANS1,
ATRANS5 Registers

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26:16	SIZE	Translation aperture size for this virtual address range, in units of 4 kiB (one flash sector). Bits 21:12 of the virtual address are compared to SIZE. Offsets greater than SIZE return a bus error, and do not cause a QSPI access.	RW	0x400
15:12	Reserved.	-	-	-
11:0	BASE	Physical address base for this virtual address range, in units of 4 kiB (one flash sector). Taking a 24-bit virtual address, firstly bits 23:22 (the two MSBs) are masked to zero, and then BASE is added to bits 23:12 (the upper 12 bits) to form the physical address. Translation wraps on a 16 MiB boundary.	RW	0x400

QMI: ATRANS2, ATRANS6 Registers

Offsets: 0x3c, 0x4c

Description

Configure address translation for XIP virtual addresses 0x800000 through 0xbffff (a 4 MiB window starting at +8 MiB).

Address translation allows a program image to be executed in place at multiple physical flash addresses (for example, a double-buffered flash image for over-the-air updates), without the overhead of position-independent code.

At reset, the address translation registers are initialised to an identity mapping, so that they can be ignored if address translation is not required.

Note that the XIP cache is fully virtually addressed, so a cache flush is required after changing the address translation.

Table 1264. ATRANS2,
ATRANS6 Registers

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26:16	SIZE	Translation aperture size for this virtual address range, in units of 4 kiB (one flash sector). Bits 21:12 of the virtual address are compared to SIZE. Offsets greater than SIZE return a bus error, and do not cause a QSPI access.	RW	0x400
15:12	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
11:0	BASE	<p>Physical address base for this virtual address range, in units of 4 kiB (one flash sector).</p> <p>Taking a 24-bit virtual address, firstly bits 23:22 (the two MSBs) are masked to zero, and then BASE is added to bits 23:12 (the upper 12 bits) to form the physical address. Translation wraps on a 16 MiB boundary.</p>	RW	0x800

QMI: ATRANS3, ATRANS7 Registers

Offsets: 0x40, 0x50

Description

Configure address translation for XIP virtual addresses 0xc00000 through 0xfffffff (a 4 MiB window starting at +12 MiB).

Address translation allows a program image to be executed in place at multiple physical flash addresses (for example, a double-buffered flash image for over-the-air updates), without the overhead of position-independent code.

At reset, the address translation registers are initialised to an identity mapping, so that they can be ignored if address translation is not required.

Note that the XIP cache is fully virtually addressed, so a cache flush is required after changing the address translation.

Table 1265. ATRANS3, ATRANS7 Registers

Bits	Name	Description	Type	Reset
31:27	Reserved.	-	-	-
26:16	SIZE	<p>Translation aperture size for this virtual address range, in units of 4 kiB (one flash sector).</p> <p>Bits 21:12 of the virtual address are compared to SIZE. Offsets greater than SIZE return a bus error, and do not cause a QSPI access.</p>	RW	0x400
15:12	Reserved.	-	-	-
11:0	BASE	<p>Physical address base for this virtual address range, in units of 4 kiB (one flash sector).</p> <p>Taking a 24-bit virtual address, firstly bits 23:22 (the two MSBs) are masked to zero, and then BASE is added to bits 23:12 (the upper 12 bits) to form the physical address. Translation wraps on a 16 MiB boundary.</p>	RW	0xc00

12.15. System Control Registers

These registers are not associated with any particular peripheral. They control, or provide information about, system-level hardware such as the bus fabric. This is also where chip identification information such as the JEDEC IDCODE are provided in a software-accessible manner.

12.15.1. SYSINFO

12.15.1.1. Overview

The sysinfo block contains system information. The first register contains the Chip ID, which allows the programmer to know which version of the chip software is running on. The second register indicates which package configuration is used (QFN-60 or QFN-80). The third register will always read as 1.

12.15.1.2. List of Registers

The sysinfo registers start at a base address of `0x40000000` (defined as `SYSINFO_BASE` in SDK).

Table 1266. List of SYSINFO registers

Offset	Name	Info
0x00	CHIP_ID	JEDEC JEP-106 compliant chip identifier.
0x04	PACKAGE_SEL	Package selection indicator, 0 = QFN80, 1 = QFN60
0x08	PLATFORM	Platform register. Allows software to know what environment it is running in during pre-production development. Post-production, the PLATFORM is always ASIC, non-SIM.
0x14	GITREF_RP2350	Git hash of the chip source. Used to identify chip version.

SYSINFO: CHIP_ID Register

Offset: 0x00

Description

JEDEC JEP-106 compliant chip identifier.

Table 1267. CHIP_ID Register

Bits	Name	Description	Type	Reset
31:28	REVISION		RO	-
27:12	PART		RO	-
11:1	MANUFACTURER		RO	-
0	STOP_BIT		RO	0x1

SYSINFO: PACKAGE_SEL Register

Offset: 0x04

Table 1268. PACKAGE_SEL Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Package selection indicator, 0 = QFN80, 1 = QFN60	RO	0x0

SYSINFO: PLATFORM Register

Offset: 0x08

Description

Platform register. Allows software to know what environment it is running in during pre-production development. Post-production, the PLATFORM is always ASIC, non-SIM.

Table 1269. PLATFORM Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	GATESIM		RO	-

Bits	Name	Description	Type	Reset
3	BATCHSIM		RO	-
2	HDLSIM		RO	-
1	ASIC		RO	-
0	FPGA		RO	-

SYSINFO: GITREF_RP2350 Register

Offset: 0x14

Table 1270.
GITREF_RP2350
Register

Bits	Description	Type	Reset
31:0	Git hash of the chip source. Used to identify chip version.	RO	-

12.15.2. SYSCFG

12.15.2.1. Overview

The system config block controls miscellaneous chip settings, including:

- Check debug halt status of both cores
- Processor GPIO input synchroniser control (set to 1 to allow input synchroniser bypassing to reduce latency for synchronous clocks)
- SWD interface control from inside the chip (allows one core to debug another, which may make debug connectivity easier)
- State-retaining memory power down (SRAM periphery can be powered down when not in use to save a small amount of power)
 - when enabled, power is still applied to the SRAM storage array; use the Power Manager (Chapter 6) to completely remove power
- Spare registers
 - 0 - Force POWMAN `wdreset` (for use when the user wants the watchdog to restart the PSM from any of the early stages)
 - 1 - Disable TRNG LPOS C entropy source

12.15.2.2. Changes from RP2040

- Moved the NMI mask to per-core registers in the EPPB (Section 3.7.5.1). The new registers reset on a processor warm reset, which avoids issues with NMIs asserting during the bootrom early boot process.
- Expanded `MEMPOWERDOWN` to cover new memory banks
- Removed controls from `DBGFORCE` to account for the new single-DP debug topology

12.15.2.3. List of Registers

The system config registers start at a base address of `0x40008000` (defined as `SYSCFG_BASE` in SDK).

Table 1271. List of
SYSCFG registers

Offset	Name	Info
0x00	PROC_CONFIG	Configuration for processors
0x04	PROC_IN_SYNC_BYPASS	For each bit, if 1, bypass the input synchronizer between that GPIO and the GPIO input register in the SIO. The input synchronizers should generally be unbypassed, to avoid injecting metastabilities into processors. If you're feeling brave, you can bypass to save two cycles of input latency. This register applies to GPIO 0...31.
0x08	PROC_IN_SYNC_BYPASS_HI	For each bit, if 1, bypass the input synchronizer between that GPIO and the GPIO input register in the SIO. The input synchronizers should generally be unbypassed, to avoid injecting metastabilities into processors. If you're feeling brave, you can bypass to save two cycles of input latency. This register applies to GPIO 32...47. USB GPIO 56..57 QSPI GPIO 58..63
0x0c	DBGFORCE	Directly control the chip SWD debug port
0x10	MEMPOWERDOWN	Control PD pins to memories. Set high to put memories to a low power state. In this state the memories will retain contents but not be accessible Use with caution
0x14	SPARE	Spare registers

SYSCFG: PROC_CONFIG Register

Offset: 0x00

Description

Configuration for processors

Table 1272.
PROC_CONFIG
Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	PROC1_HALTED	Indication that proc1 has halted	RO	0x0
0	PROC0_HALTED	Indication that proc0 has halted	RO	0x0

SYSCFG: PROC_IN_SYNC_BYPASS Register

Offset: 0x04

Description

For each bit, if 1, bypass the input synchronizer between that GPIO and the GPIO input register in the SIO. The input synchronizers should generally be unbypassed, to avoid injecting metastabilities into processors. If you're feeling brave, you can bypass to save two cycles of input latency. This register applies to GPIO 0...31.

Table 1273.
PROC_IN_SYNC_BYPA
SS Register

Bits	Name	Description	Type	Reset
31:0	GPIO		RW	0x00000000

SYSCFG: PROC_IN_SYNC_BYPASS_HI Register

Offset: 0x08

Description

For each bit, if 1, bypass the input synchronizer between that GPIO and the GPIO input register in the SIO. The input synchronizers should generally be unbypassed, to avoid injecting metastabilities into processors. If you're feeling brave, you can bypass to save two cycles of input latency. This register applies to GPIO 32...47. USB GPIO 56..57 QSPI GPIO 58..63

Table 1274.
PROC_IN_SYNC_BYPA
SS_HI Register

Bits	Name	Description	Type	Reset
31:28	QSPI_SD		RW	0x0
27	QSPI_CSN		RW	0x0
26	QSPI_SCK		RW	0x0
25	USB_DM		RW	0x0
24	USB_DP		RW	0x0
23:16	Reserved.	-	-	-
15:0	GPIO		RW	0x0000

SYSCFG: DBGFORCE Register

Offset: 0x0c

Description

Directly control the chip SWD debug port

Table 1275.
DBGFORCE Register

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3	ATTACH	Attach chip debug port to syscfg controls, and disconnect it from external SWD pads.	RW	0x0
2	SWCLK	Directly drive SWCLK, if ATTACH is set	RW	0x1
1	SWDI	Directly drive SWDIO input, if ATTACH is set	RW	0x1
0	SWDO	Observe the value of SWDIO output.	RO	-

SYSCFG: MEMPOWERDOWN Register

Offset: 0x10

Description

Control PD pins to memories.

Set high to put memories to a low power state. In this state the memories will retain contents but not be accessible. Use with caution

Table 1276.
MEMPOWERDOWN
Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	BOOTRAM		RW	0x0

Bits	Name	Description	Type	Reset
11	ROM		RW	0x0
10	USB		RW	0x0
9	SRAM9		RW	0x0
8	SRAM8		RW	0x0
7	SRAM7		RW	0x0
6	SRAM6		RW	0x0
5	SRAM5		RW	0x0
4	SRAM4		RW	0x0
3	SRAM3		RW	0x0
2	SRAM2		RW	0x0
1	SRAM1		RW	0x0
0	SRAM0		RW	0x0

SYSCFG: SPARE Register

Offset: 0x14

Description

Spare registers

Table 1277. SPARE Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	bit 0 - Force POWMAN to switch to Iposc clock (used for a safe software initiated reset from early stages of RSM) bit 1 - disable trng entropy from Iposc. No need to set this high. bit 2-7 are not used and have no effect on the chip	RW	0x00

12.15.3. TBMAN

TBMAN refers to the testbench manager, used during chip development simulations to verify the design. During these simulations TBMAN allows software running on RP2350 to control the testbench and simulation environment. On the real chip, it has no effect other than providing a single **PLATFORM** register that indicates that this is the real chip. This **PLATFORM** functionality is duplicated in the sysinfo (Section 12.15.1) registers.

12.15.3.1. List of Registers

The TBMAN registers start at a base address of **0x40160000** (defined as **TBMAN_BASE** in SDK).

Table 1278. List of TBMAN registers

Offset	Name	Info
0x0	PLATFORM	Indicates the type of platform in use

TBMAN: PLATFORM Register

Offset: 0x0

Description

Indicates the type of platform in use

Table 1279.
PLATFORM Register

Bits	Name	Description	Type	Reset
31:3	Reserved.	-	-	-
2	HDLSIM	Indicates the platform is a simulation	RO	0x0
1	FPGA	Indicates the platform is an FPGA	RO	0x0
0	ASIC	Indicates the platform is an ASIC	RO	0x1

12.15.4. BUSCTRL

This block provides basic controls and monitoring for the system bus fabric.

PERFSEL x	Event	Description
0	APB access, contested	Completion of an access to the APB arbiter (which is upstream of all APB peripherals), which was previously delayed due to an access by another master.
1	APB access	Completion of an access to the APB arbiter
2	FASTPERI access, contested	Completion of an access to the FASTPERI arbiter (which is upstream of PIOs, DMA config port, USB, XIP aux FIFO port), which was previously delayed due to an access by another master.
3	FASTPERI access	Completion of an access to the FASTPERI arbiter
4	SRAM5 access, contested	Completion of an access to the SRAM5 arbiter, which was previously delayed due to an access by another master.
5	SRAM5 access	Completion of an access to the SRAM5 arbiter
6	SRAM4 access, contested	Completion of an access to the SRAM4 arbiter, which was previously delayed due to an access by another master.
7	SRAM4 access	Completion of an access to the SRAM4 arbiter
8	SRAM3 access, contested	Completion of an access to the SRAM3 arbiter, which was previously delayed due to an access by another master.
9	SRAM3 access	Completion of an access to the SRAM3 arbiter
10	SRAM2 access, contested	Completion of an access to the SRAM2 arbiter, which was previously delayed due to an access by another master.
11	SRAM2 access	Completion of an access to the SRAM2 arbiter
12	SRAM1 access, contested	Completion of an access to the SRAM1 arbiter, which was previously delayed due to an access by another master.
13	SRAM1 access	Completion of an access to the SRAM1 arbiter
14	SRAM0 access, contested	Completion of an access to the SRAM0 arbiter, which was previously delayed due to an access by another master.
15	SRAM0 access	Completion of an access to the SRAM0 arbiter
16	XIP_MAIN access, contested	Completion of an access to the XIP_MAIN arbiter, which was previously delayed due to an access by another master.
17	XIP_MAIN access	Completion of an access to the XIP_MAIN arbiter

PERFSELx	Event	Description
18	ROM access, contested	Completion of an access to the ROM arbiter, which was previously delayed due to an access by another master.
19	ROM access	Completion of an access to the ROM arbiter

12.15.4.1. List of Registers

The Bus Fabric registers start at a base address of `0x40068000` (defined as `BUSCTRL_BASE` in SDK).

Table 1280. List of `BUSCTRL` registers

Offset	Name	Info
0x00	<code>BUS_PRIORITY</code>	Set the priority of each master for bus arbitration.
0x04	<code>BUS_PRIORITY_ACK</code>	Bus priority acknowledge
0x08	<code>PERFCTR_EN</code>	Enable the performance counters. If 0, the performance counters do not increment. This can be used to precisely start/stop event sampling around the profiled section of code. The performance counters are initially disabled, to save energy.
0x0c	<code>PERFCTR0</code>	Bus fabric performance counter 0
0x10	<code>PERFSEL0</code>	Bus fabric performance event select for <code>PERFCTR0</code>
0x14	<code>PERFCTR1</code>	Bus fabric performance counter 1
0x18	<code>PERFSEL1</code>	Bus fabric performance event select for <code>PERFCTR1</code>
0x1c	<code>PERFCTR2</code>	Bus fabric performance counter 2
0x20	<code>PERFSEL2</code>	Bus fabric performance event select for <code>PERFCTR2</code>
0x24	<code>PERFCTR3</code>	Bus fabric performance counter 3
0x28	<code>PERFSEL3</code>	Bus fabric performance event select for <code>PERFCTR3</code>

BUSCTRL: BUS_PRIORITY Register

Offset: 0x00

Description

Set the priority of each master for bus arbitration.

Table 1281.
`BUS_PRIORITY`
Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	DMA_W	0 - low priority, 1 - high priority	RW	0x0
11:9	Reserved.	-	-	-
8	DMA_R	0 - low priority, 1 - high priority	RW	0x0
7:5	Reserved.	-	-	-
4	PROC1	0 - low priority, 1 - high priority	RW	0x0
3:1	Reserved.	-	-	-
0	PROC0	0 - low priority, 1 - high priority	RW	0x0

BUSCTRL: BUS_PRIORITY_ACK Register

Offset: 0x04

Description

Bus priority acknowledge

Table 1282.
BUS_PRIORITY_ACK Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Goes to 1 once all arbiters have registered the new global priority levels. Arbiters update their local priority when servicing a new nonsequential access. In normal circumstances this will happen almost immediately.	RO	0x0

BUSCTRL: PERFCTR_EN Register

Offset: 0x08

Table 1283.
PERFCTR_EN Register

Bits	Description	Type	Reset
31:1	Reserved.	-	-
0	Enable the performance counters. If 0, the performance counters do not increment. This can be used to precisely start/stop event sampling around the profiled section of code. The performance counters are initially disabled, to save energy.	RW	0x0

BUSCTRL: PERFCTR0 Register

Offset: 0x0c

Description

Bus fabric performance counter 0

Table 1284.
PERFCTR0 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 0 Count some event signal from the busfabric arbiters, if PERFCTR_EN is set. Write any value to clear. Select an event to count using PERFSEL0	WC	0x000000

BUSCTRL: PERFSEL0 Register

Offset: 0x10

Description

Bus fabric performance event select for PERFCTR0

Table 1285. *PERFSEL0*
Register

Bits	Description	Type	Reset
31:7	Reserved.	-	-

Bits	Description	Type	Reset
6:0	<p>Select an event for PERFCTR0. For each downstream port of the main crossbar, four events are available: ACCESS, an access took place; ACCESS_CONTESTED, an access took place that previously stalled due to contention from other masters; STALL_DOWNSTREAM, count cycles where any master stalled due to a stall on the downstream bus; STALL_UPSTREAM, count cycles where any master stalled for any reason, including contention from other masters.</p> <p>0x00 → siob_proc1_stall_upstream 0x01 → siob_proc1_stall_downstream 0x02 → siob_proc1_access_contested 0x03 → siob_proc1_access 0x04 → siob_proc0_stall_upstream 0x05 → siob_proc0_stall_downstream 0x06 → siob_proc0_access_contested 0x07 → siob_proc0_access 0x08 → apb_stall_upstream 0x09 → apb_stall_downstream 0x0a → apb_access_contested 0x0b → apb_access 0x0c → fastperi_stall_upstream 0x0d → fastperi_stall_downstream 0x0e → fastperi_access_contested 0x0f → fastperi_access 0x10 → sram9_stall_upstream 0x11 → sram9_stall_downstream 0x12 → sram9_access_contested 0x13 → sram9_access 0x14 → sram8_stall_upstream 0x15 → sram8_stall_downstream 0x16 → sram8_access_contested 0x17 → sram8_access 0x18 → sram7_stall_upstream 0x19 → sram7_stall_downstream 0x1a → sram7_access_contested 0x1b → sram7_access 0x1c → sram6_stall_upstream 0x1d → sram6_stall_downstream 0x1e → sram6_access_contested 0x1f → sram6_access 0x20 → sram5_stall_upstream 0x21 → sram5_stall_downstream 0x22 → sram5_access_contested 0x23 → sram5_access 0x24 → sram4_stall_upstream 0x25 → sram4_stall_downstream 0x26 → sram4_access_contested 0x27 → sram4_access 0x28 → sram3_stall_upstream 0x29 → sram3_stall_downstream 0x2a → sram3_access_contested 0x2b → sram3_access 0x2c → sram2_stall_upstream 0x2d → sram2_stall_downstream 0x2e → sram2_access_contested 0x2f → sram2_access 0x30 → sram1_stall_upstream</p>	RW	0x1f

BUSCTRL: PERFCTR1 Register

Offset: 0x14

Description

Bus fabric performance counter 1

Table 1286.
PERFCTR1 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 1 Count some event signal from the busfabric arbiters, if PERFCTR_EN is set. Write any value to clear. Select an event to count using PERFSEL1	WC	0x000000

BUSCTRL: PERFSEL1 Register

Offset: 0x18

Description

Bus fabric performance event select for PERFCTR1

Table 1287. *PERFSEL1*
Register

Bits	Description	Type	Reset
31:7	Reserved.	-	-

Bits	Description	Type	Reset
6:0	<p>Select an event for PERFCTR1. For each downstream port of the main crossbar, four events are available: ACCESS, an access took place; ACCESS_CONTESTED, an access took place that previously stalled due to contention from other masters; STALL_DOWNSTREAM, count cycles where any master stalled due to a stall on the downstream bus; STALL_UPSTREAM, count cycles where any master stalled for any reason, including contention from other masters.</p> <p>0x00 → siob_proc1_stall_upstream 0x01 → siob_proc1_stall_downstream 0x02 → siob_proc1_access_contested 0x03 → siob_proc1_access 0x04 → siob_proc0_stall_upstream 0x05 → siob_proc0_stall_downstream 0x06 → siob_proc0_access_contested 0x07 → siob_proc0_access 0x08 → apb_stall_upstream 0x09 → apb_stall_downstream 0x0a → apb_access_contested 0x0b → apb_access 0x0c → fastperi_stall_upstream 0x0d → fastperi_stall_downstream 0x0e → fastperi_access_contested 0x0f → fastperi_access 0x10 → sram9_stall_upstream 0x11 → sram9_stall_downstream 0x12 → sram9_access_contested 0x13 → sram9_access 0x14 → sram8_stall_upstream 0x15 → sram8_stall_downstream 0x16 → sram8_access_contested 0x17 → sram8_access 0x18 → sram7_stall_upstream 0x19 → sram7_stall_downstream 0x1a → sram7_access_contested 0x1b → sram7_access 0x1c → sram6_stall_upstream 0x1d → sram6_stall_downstream 0x1e → sram6_access_contested 0x1f → sram6_access 0x20 → sram5_stall_upstream 0x21 → sram5_stall_downstream 0x22 → sram5_access_contested 0x23 → sram5_access 0x24 → sram4_stall_upstream 0x25 → sram4_stall_downstream 0x26 → sram4_access_contested 0x27 → sram4_access 0x28 → sram3_stall_upstream 0x29 → sram3_stall_downstream 0x2a → sram3_access_contested 0x2b → sram3_access 0x2c → sram2_stall_upstream 0x2d → sram2_stall_downstream 0x2e → sram2_access_contested 0x2f → sram2_access 0x30 → sram1_stall_upstream</p>	RW	0x1f

BUSCTRL: PERFCTR2 Register

Offset: 0x1c

Description

Bus fabric performance counter 2

Table 1288.
PERFCTR2 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 2 Count some event signal from the busfabric arbiters, if PERFCTR_EN is set. Write any value to clear. Select an event to count using PERFSEL2	WC	0x000000

BUSCTRL: PERFSEL2 Register

Offset: 0x20

Description

Bus fabric performance event select for PERFCTR2

Table 1289. *PERFSEL2*
Register

Bits	Description	Type	Reset
31:7	Reserved.	-	-

Bits	Description	Type	Reset
6:0	<p>Select an event for PERFCTR2. For each downstream port of the main crossbar, four events are available: ACCESS, an access took place; ACCESS_CONTESTED, an access took place that previously stalled due to contention from other masters; STALL_DOWNSTREAM, count cycles where any master stalled due to a stall on the downstream bus; STALL_UPSTREAM, count cycles where any master stalled for any reason, including contention from other masters.</p> <p>0x00 → siob_proc1_stall_upstream 0x01 → siob_proc1_stall_downstream 0x02 → siob_proc1_access_contested 0x03 → siob_proc1_access 0x04 → siob_proc0_stall_upstream 0x05 → siob_proc0_stall_downstream 0x06 → siob_proc0_access_contested 0x07 → siob_proc0_access 0x08 → apb_stall_upstream 0x09 → apb_stall_downstream 0x0a → apb_access_contested 0x0b → apb_access 0x0c → fastperi_stall_upstream 0x0d → fastperi_stall_downstream 0x0e → fastperi_access_contested 0x0f → fastperi_access 0x10 → sram9_stall_upstream 0x11 → sram9_stall_downstream 0x12 → sram9_access_contested 0x13 → sram9_access 0x14 → sram8_stall_upstream 0x15 → sram8_stall_downstream 0x16 → sram8_access_contested 0x17 → sram8_access 0x18 → sram7_stall_upstream 0x19 → sram7_stall_downstream 0x1a → sram7_access_contested 0x1b → sram7_access 0x1c → sram6_stall_upstream 0x1d → sram6_stall_downstream 0x1e → sram6_access_contested 0x1f → sram6_access 0x20 → sram5_stall_upstream 0x21 → sram5_stall_downstream 0x22 → sram5_access_contested 0x23 → sram5_access 0x24 → sram4_stall_upstream 0x25 → sram4_stall_downstream 0x26 → sram4_access_contested 0x27 → sram4_access 0x28 → sram3_stall_upstream 0x29 → sram3_stall_downstream 0x2a → sram3_access_contested 0x2b → sram3_access 0x2c → sram2_stall_upstream 0x2d → sram2_stall_downstream 0x2e → sram2_access_contested 0x2f → sram2_access 0x30 → sram1_stall_upstream</p>	RW	0x1f

BUSCTRL: PERFCTR3 Register

Offset: 0x24

Description

Bus fabric performance counter 3

Table 1290.
PERFCTR3 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Busfabric saturating performance counter 3 Count some event signal from the busfabric arbiters, if PERFCTR_EN is set. Write any value to clear. Select an event to count using PERFSEL3	WC	0x000000

BUSCTRL: PERFSEL3 Register

Offset: 0x28

Description

Bus fabric performance event select for PERFCTR3

Table 1291. *PERFSEL3*
Register

Bits	Description	Type	Reset
31:7	Reserved.	-	-

Bits	Description	Type	Reset
6:0	<p>Select an event for PERFCTR3. For each downstream port of the main crossbar, four events are available: ACCESS, an access took place; ACCESS_CONTESTED, an access took place that previously stalled due to contention from other masters; STALL_DOWNSTREAM, count cycles where any master stalled due to a stall on the downstream bus; STALL_UPSTREAM, count cycles where any master stalled for any reason, including contention from other masters.</p> <p>0x00 → siob_proc1_stall_upstream 0x01 → siob_proc1_stall_downstream 0x02 → siob_proc1_access_contested 0x03 → siob_proc1_access 0x04 → siob_proc0_stall_upstream 0x05 → siob_proc0_stall_downstream 0x06 → siob_proc0_access_contested 0x07 → siob_proc0_access 0x08 → apb_stall_upstream 0x09 → apb_stall_downstream 0x0a → apb_access_contested 0x0b → apb_access 0x0c → fastperi_stall_upstream 0x0d → fastperi_stall_downstream 0x0e → fastperi_access_contested 0x0f → fastperi_access 0x10 → sram9_stall_upstream 0x11 → sram9_stall_downstream 0x12 → sram9_access_contested 0x13 → sram9_access 0x14 → sram8_stall_upstream 0x15 → sram8_stall_downstream 0x16 → sram8_access_contested 0x17 → sram8_access 0x18 → sram7_stall_upstream 0x19 → sram7_stall_downstream 0x1a → sram7_access_contested 0x1b → sram7_access 0x1c → sram6_stall_upstream 0x1d → sram6_stall_downstream 0x1e → sram6_access_contested 0x1f → sram6_access 0x20 → sram5_stall_upstream 0x21 → sram5_stall_downstream 0x22 → sram5_access_contested 0x23 → sram5_access 0x24 → sram4_stall_upstream 0x25 → sram4_stall_downstream 0x26 → sram4_access_contested 0x27 → sram4_access 0x28 → sram3_stall_upstream 0x29 → sram3_stall_downstream 0x2a → sram3_access_contested 0x2b → sram3_access 0x2c → sram2_stall_upstream 0x2d → sram2_stall_downstream 0x2e → sram2_access_contested 0x2f → sram2_access 0x30 → sram1_stall_upstream</p>	RW	0x1f

Chapter 13. OTP

RP2350 provides 8kB of one-time programmable storage (OTP), which holds:

- Preprogrammed per-device information, such as unique device identifier and oscillator trim values
- Security configuration such as debug disable and secure boot enable
- Public key fingerprints for secure boot
- Symmetric keys for decryption of flash contents into SRAM
- Configuration for the USB bootloader, such as customising VID/PID and descriptors
- Bootable software images, for low-cost flashless applications or custom bootloaders
- Any other user-defined data, such as per-device personalisation values

For the full listing of predefined OTP contents, see [Section 13.9](#).

OTP is physically an array of 4096 rows of 24 bits each. You can directly access these 24-bit values, but there is also hardware support for storing 16 bits of data in each row, with 6 bits of Hamming ECC protection and 2 bits of bit polarity reversal protection, yielding an ECC data capacity of 8192 bytes.

On a blank device, the OTP contents is all zeroes, except for some basic device information pre-programmed during manufacturing test. Each bit can be *irreversibly* programmed from zero to one. To program the OTP contents:

- directly access the registers using the SBPI bridge
- call the bootrom `otp_access` API ([Section 5.4.11.4](#))
- use the PICOBLOCK interface of the USB bootloader ([Section 5.6](#))

RP2350 enforces page-based permissions on OTP to partition Secure from Non-secure data and to ensure that contents that should not change *do not* change. The OTP address space is logically partitioned into 64 pages, each 64 rows in size, for a total of 128 bytes of ECC data per page. Pages initially have full read-write permissions, but can be restricted to read-only or inaccessible for each of Secure, Non-secure and bootloader access.

The page permissions themselves are stored in OTP. Locking pages in this way is an irreversible operation, referred to as *hard locking*. The hardware also supports *soft locking*, where a page's permissions are further restricted by writing to the relevant register in `SW_LOCK0` through `SW_LOCK63`; this restriction remains in effect until the next OTP reset. Resetting the OTP block also resets the processors, so soft locking can be used to restrict the availability of sensitive content like decryption keys to early boot stages.

OTP access keys ([Section 13.5.2](#)) provide an additional layer of protection. A fixed challenge is written to a write-only OTP area. Pages registered to that key require the key to be entered to a write-only register in order to open read or write access. This supports configuration data that can be accessed or edited by the board manufacturer, but not by general firmware running on the device.

13.1. OTP Address Map

The OTP hardware resides in a 128kB region starting at `0x40120000` (OTP_BASE in the SDK). Bit 16 of the address is used to select either the OTP control registers, in the lower 64kB, or one of the OTP read data aliases, in the upper 64 kB of this space.

The OTP control registers ([Section 13.8](#)) are aliased at 4kB intervals to implement the usual set, clear, and XOR atomic write aliases described in [Section 2.1.3](#).

The read data region starting at `0x40130000` divides further into four aliases:

- `0x40130000, OTP_DATA_BASE`: ECC read alias. A 32-bit read returns the ECC-corrected data for two neighbouring rows, or all-ones on permission failure. Only the first 8kB is populated.

- **0x40138000, OTP_DATA_GUARDED_BASE**: ECC guarded read alias. Successful reads return the same data as **OTP_DATA_BASE**. Only the first 8kB is populated.
- **0x40134000, OTP_DATA_RAW_BASE**: raw read alias. A 32-bit read directly returns the 24-bit contents of a single row, with zeroes in the eight MSBs, or returns all-ones on permission failure.
- **0x4013c000, OTP_DATA_RAW_GUARDED_BASE**: raw, guarded read alias. Successful reads return the same data as **OTP_DATA_RAW_BASE**.

Bit 15 of the address selects ECC (0) vs raw (1). Bit 14 of the address selects unguarded (0) vs guarded (1) access. Guarded reads return the same data as unguarded reads, but perform additional hardware consistency checks and return bus faults on permission failure. For more information, see [Section 13.1.1](#).

❗ **IMPORTANT**

The read data regions starting at **0x40130000** are accessible only when **USR.DCTRL** is set, otherwise all reads return a bus error response. This bit is clear when the OTP is being programmed via the SBPI bridge.

Writing to the read data aliases is not a valid operation, and will always return a bus fault. The OTP is programmed by the SBPI bridge, which is used internally by the bootrom **otp_access** API, [Section 5.4.11.4](#).

13.1.1. Guarded Reads

Reads through the guarded aliases differ from unguarded reads in the following ways:

- Permission failures return bus faults rather than a bit pattern of all-ones.
- Uncorrectable ECC errors return a bus fault if detected.
- Guarded reads perform an additional hardware consistency check to detect power transients. If this check fails, the read returns a bus fault.

These checks help to make the OTP fail-safe in contexts where deliberate fault injection is a possibility. For example, the RP2350 bootrom uses guarded reads to check boot configuration flags.

The data returned from a successful guarded read is the same as the data returned by a successful read from the corresponding unguarded alias.

13.2. Background: OTP IP Details

The RP2350 OTP subsystem uses the Synopsys NVM OTP IP, which comes in 3 parts:

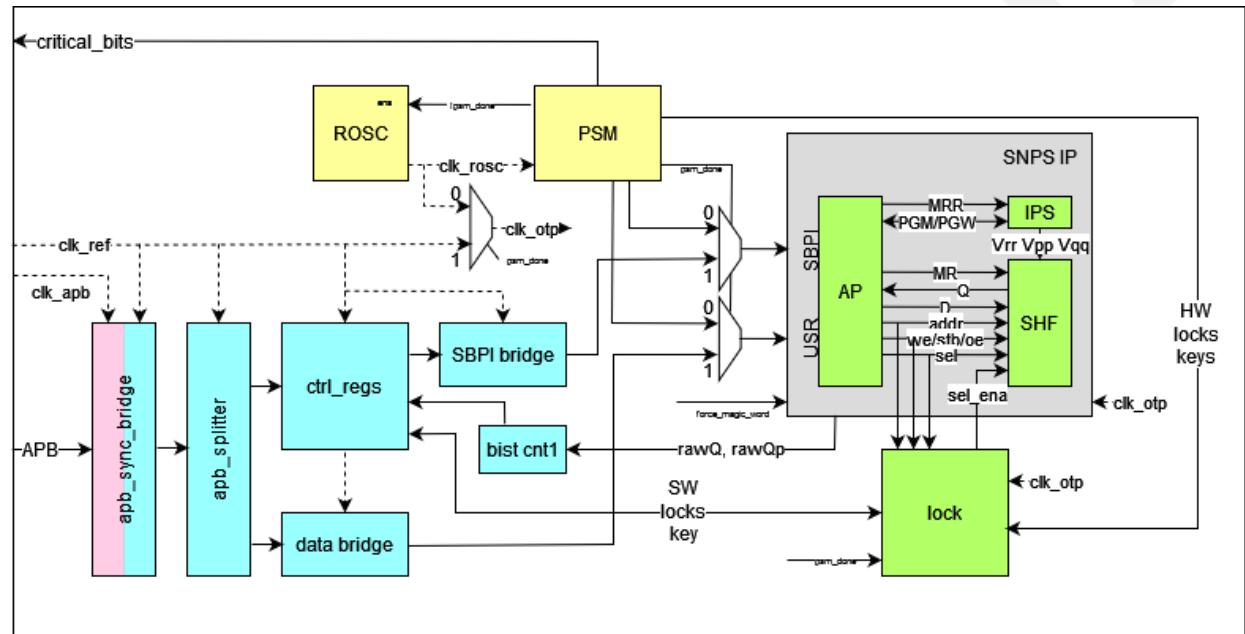
- Integrated Power Supply (IPS), including:
 - Charge Pump (for programming)
 - Regulator (for reading)
- OTP Macro (SHF, Fuse)
 - 4096 × 24 (8kB with ECC, 16-bit ECC write granularity)
- Access port (AP), providing:
 - Basic read access
 - Programming access
 - ECC and bit redundancy
 - BOOT function, which polls for stable OTP power supply at start-of-day

13.3. Background: OTP Hardware Architecture

This diagram shows the integration of the three Synopsys IP components, and the Raspberry Pi hardware added to make this all function in the context of RP2350's system and security architecture. More specifically:

- APB interface(s) to connect to the SoC
 - Internal ring oscillator with clock edge randomisation
 - Power-up state machine, running off the ring oscillator
 - Lock shim, sitting between the SNPS RTL and the memory core (fuse)

Figure 118. OTP architecture



The OTP subsystem clock is initially provided by the OTP boot oscillator (Section 13.3.3) during hardware startup, but switches to `clk_ref` before any software runs on the processors. The frequency of `clk_ref` must not exceed 25 MHz when accessing the OTP.

13.3.1. Lock Shim

The lock shim is inserted between the Synopsys AP block and the SHF block, and is used to enforce read/write page locks, based on:

- The OTP address presented on the AP → SHF bus
 - The read/write strobe on the AP → SHF bus
 - The security attribute of the upstream bus access which caused this SHF access (assumed to be Secure if SBPI is currently enabled via USR.DCTRL)

Because the Synopsys AP performs both reads and writes in the course of programming an OTP row, it is impossible to disable reads to an address without also disabling writes. Three lock states are supported:

- Read/Write
 - Read-only
 - Inaccessible

The full locking scheme is described in [Section 13.5](#), but to summarise:

- The lock state of each OTP page is read from OTP at boot time.

- There is a separate copy of the lock state for Secure/Non-secure accesses. The lock shim applies the Secure read permissions to Secure reads, and the Non-secure read permissions to Non-secure reads. There is no such rule for writes, because Non-secure code is not capable of accessing the programming hardware.
- The lock encoding in OTP storage is such that a page can always be locked down to a less permissive state (in the order RW → RO → Inaccessible) but can never return to a more permissive state.
- Software can advance the state of each individual lock at runtime without programming OTP, and this lasts until the OTP PSM is re-run.
- Software locks also obey the lock progression order (RW → RO → Inaccessible) and can not be regressed.

The full locking scheme is described in [Section 13.5](#).

13.3.2. External Interfaces

The OTP integration has one upstream APB interface, which splits internally onto two separate interfaces. This guarantees the hardware only serves a single upstream APB access at a time, with a single PPROT security level.

The first APB interface is the *data* interface (or data bridge) (OTPD). It has the following characteristics:

- Read-only
- Connects to the Synopsys device access port (DAP)
- Data interface reads always return 32 or 24 bits of valid data
- The data interface address is rounded down to a multiple of 32 bits, so that narrow reads return the correct byte lanes
- There is an 8kB window which supports 32-bit ECC reads
 - Each upstream bus read is split into two OTP accesses, each of which returns 16 bits of error-corrected data from the OTP
- There is an 8kB window which supports guarded 32-bit ECC reads, and returns a bus error if the guarding read fails.
 - Functions the same as the ECC read window, but reads the Synopsys boot word before accessing the OTP array, and return a bus error if the first read does not match the expected constant
 - Used to increase confidence in software OTP reads in the bootrom
- There is a 16kB window which supports 24-bit raw reads
 - Each access returns a single raw 24-bit OTP row, bypassing error correction
 - Software must provide its own redundancy (e.g. triple majority vote)
 - Allows bit-mutable data structures, such as boot flags, or thermometer counters

The second APB interface is the *command* interface. This provides two main functions:

- Provides a bridge to the SBPI interface (Synopsys proprietary Serial and Byte-Parallel Interface bus)
 - SBPI connects to the Programmable Master Controller (PMC), with access to the DAP, DATAPATH, charge pump (IPS), and fuse memory (SHF)
 - Allows arbitrary OTP operations, including programming
 - Only accessible to Secure reads and writes
- Provides control registers for Raspberry Pi hardware
 - Registers have different accessibility according to Secure/Non-secure and read/write
 - Software lock registers are always readable by both security domains

Hardware configuration data read from OTP during the power-up sequence drives system-level control signals, e.g.

disabling CoreSight APs. This is described in more detail in (Section 13.3.4).

A single system-level interrupt output (IRQ) generates interrupts for the following sources:

- Secure read failed due to locks
- Non-secure read failed due to locks
- Write failed due to locks
- SBPI FLAG, used by the PMC to signal completion
- Data port access when DCTRL is set error
 - `USR.DCTRL` tells the SNPS AP whether the SBPI bridge or data bridge can access the memory array; this helps debugging SW if a data access is attempted whilst the DAP is inaccessible

Any failed access also returns a bus fault (`PSLVERR`). To determine whether an OTP address is accessible, query the lock tables.

Non-secure code cannot access the interrupt status registers.

13.3.3. OTP Boot Oscillator

The OTP startup sequence (Section 13.3.4) runs from a local ring oscillator, dedicated to the OTP subsystem. This is separate from the system ring oscillator (the ROSC) which provides the system clock to run the processors during boot.

- The OTP boot oscillator is the only clock used by the OTP power-up state machine
- The OTP boot oscillator dynamically randomises its own frequency controls, to deliberately add jitter to the clock
- The OTP boot oscillator stops when the PSM completes, and does not start again until the OTP resets
- The OTP clock automatically switches to `clk_ref` when the OTP boot oscillator stops

The boot oscillator has a nominal frequency of 12MHz. It provides the clock for reading out hardware configuration from OTP, including the critical flags (Section 13.4) which configure hardware security features such as debug disable and the glitch detectors.

Keeping this oscillator local to the OTP hardware subsystem reduces the power signature of the clock itself, due to the lower switched clock capacitance. Along with the random jitter of the frequency controls, this helps frustrate attempts to recover OTP access keys and debug keys via power signature analysis attacks, or to disable security features by timing fault injection against the OTP clock.

Only the OTP boot oscillator enables the ROSC frequency randomisation feature by default: for later operations using the system ROSC (Section 8.3), you must explicitly enable this feature on that oscillator, by programming the ROSC control registers. The crystal oscillator (XOSC) does not support frequency randomisation.

13.3.4. Power-up State Machine

The OTP is the second item in the switched core domain's Power-On State Machine (Section 7.4), after the processor cold reset. OTP does not release its `rst_done`, or enable any debug interface (including the factory test JTAG described in Section 10.8), until the OTP PSM reads out OTP-resident hardware configuration. The `rst_done` output to the system PSM holds the rest of the system in reset until the OTP PSM completes, so that no software runs until the OTP's contents are known.

The OTP boot sequence runs from a local ring oscillator. This oscillator is dedicated to the OTP subsystem, and is separate from the main system ROSC used by the processors at boot. The sequence is:

1. First, the PSM runs the Synopsys boot instruction. This has the following steps:
 - a. Wait for the power supply to return a 'good' value.

- b. Read consistency check location until hardware sees the correct value for 16 successive reads.

NOTE

Consistency checks use predefined words stored in mask ROM cells with similar analogue properties to OTP cells.

2. Read critical flags (non-ECC): each critical bit is redundant across 8 OTP rows, with three-of-eight vote for each flag.
3. Read hardware access keys via ECC read interface.
4. Read valid bits for hardware access keys, including the debug keys ([Section 3.5.9.2](#))
5. Initialise page lock registers from the lock page via raw read interface.
6. Assert `rst_done` signal to the system power-on state machine
7. The system reset sequence continues, starting with the system ROSC

13.4. Critical Flags

Critical flags enable hardware security features which are fundamental to RP2350's secure boot implementation. The OTP power-up state machine reads critical flags very early in the system reset sequence, before any code runs on the processors.

Most critical flags are in the main Boot Configuration page, page 1. These are listed under **CRIT1** in the OTP data listing. The exceptions are the Arm/RISC-V disable flags, which are in the Chip Info page, page 0. This page is made read-only during factory programming, so users can not write to the **CRIT0** flags.

Critical flags define **0** as the unprogrammed value, and **1** as the programmed value. On a blank device, all of the **CRIT1** flags are **0**. The reset value specified below is the value assigned to the internal logic net between the OTP reset being applied and the OTP PSM completing. For example, the reset value of **1** for the debug disable flags implies that debug is not accessible whilst the OTP PSM is running, but may be available afterward, depending on the value read from OTP storage.

- **ARM_DISABLE** (reset: **0**): Force the **ARCHSEL** register to RISC-V, at higher priority than RISC-V disable flag, secure boot enable flag, or default boot architecture flag.
- **RISCV_DISABLE** (reset: **0**): Force the **ARCHSEL** register to Arm, at higher priority than the default boot architecture flag.
- **SECURE_BOOT_ENABLE** (reset: **1**): Enable boot signature checking in bootrom, disable factory JTAG, and force the **ARCHSEL** register to Arm, at higher priority than the default boot architecture flag.
- **SECURE_DEBUG_DISABLE** (reset: **1**): Disable factory JTAG, block Secure accesses from Mem-APs, and block halt requests to Secure processors.
 - Prevents secure AP accesses by masking their `ap_secure_en` signals.
 - Prevents secure processor halting by masking the M33's `spiden` and `nspiden` signals.
 - Secure debug can be re-enabled by a Secure register in the OTP block.
 - Re-enable of Secure debug can be disabled by a Secure write-1-only lock register, also in the OTP block.
- **DEBUG_DISABLE** (reset: **1**): Completely disable the Mem-APs, in addition to disabling everything disabled by the secure debug disable flag.
- **BOOT_ARCH** (reset: **0**): set the reset value of the **ARCHSEL** register (**0** → Arm, **1** → RISC-V) if it has not been forced by other critical flags.
 - Not critical, but hardware-read.
- **GLITCH_DETECTOR_ENABLE** (reset: **0**): pass an enable signal to the glitch detectors so that they can be armed before any software runs.

- **GLITCH_DETECTOR_SENS**(reset: 0): configure the initial sensitivity of the glitch detector circuits.

Critical flags are encoded with a three-of-eight vote across eight consecutive OTP rows. Each flag is redundantly programmed to the same bit position in eight consecutive rows. Hardware considers the flag to be set if the bit reads as 1 in at least three of these eight rows. The flag is considered clear if no more than two bits are observed to be set.

JTAG disable is ignored only if the customer RMA flag ([Section 13.7](#)) is set.

For further discussion of the effects of the critical flags, see:

- [Section 3.5.9.1](#) for the effects of the debug disable flags
- [Section 3.9](#) for the effects of the Arm/RISC-V architecture select flags
- [Section 10.7](#) for the effects of the glitch detector configuration flags
- [Section 10.1.1](#) for discussion of the bootrom secure boot support enabled by the **SECURE_BOOT_ENABLE** flag

13.5. Page Locks

The OTP protection hardware logically segments OTP into 64 pages (0 through 63), each 128 bytes in size, or equivalently 64 OTP rows.

Each page has a set of lock registers which determine read and write access for that page from Secure and Non-secure code. The lock registers are preloaded from OTP at reset, and can then be advanced (i.e. made less permissive) by software. Lock registers themselves are always world-readable.

Pages 61 through 63 are not so neatly described by a single set of lock registers. These pages store lock initialisation metadata. For more details, see [Section 13.5.4](#). This section describes the more common case of a page protected by a set of page locks.

13.5.1. Lock Progression

Due to hardware constraints ([Section 13.3.1](#)), read and write restrictions are not orthogonal: it's impossible to disallow reads to an address without also disallowing writes. So, the progression of locking for a given page is:

0. Read/Write
1. Read-only
2. Inaccessible

Lock state *only increases*. This is enforced in two ways:

- Due to the nature of OTP and the choice of encoding, you cannot lower the OTP values preloaded to the lock registers during boot.
- The lock registers ignore writes of lower-than-current values.

Secure and Non-secure use separate lock values, which can advance independently of one another. There is no hardware distinction between Non-secure Read/Write and Non-secure Read-only, since Non-secure can not directly write to the OTP anyway. It is still worth encoding, because Secure software performing a write on Non-secure software's behalf can check and enforce the Non-secure write lock.

You can reprogram bits from any state to any higher state. Locks use a 2-bit thermometer code: the initial all-zeroes state is read-write, and locks are advanced by programming first bit 0, then bit 1.

Lock bits in OTP are triple-redundant with a majority vote. They can't be ECC-protected, because they may be mutated bit-by-bit over multiple programming operations.

The OTP-resident lock bits are write-protected by their own Secure lock level. The lock pages are always world-readable.

The Secure lock registers can be advanced by Secure code, and are world-readable.

The Non-secure lock registers can be advanced by Secure or Non-secure code, and are world-readable.

13.5.2. OTP Access Keys

Page 61 contains 128-bit keys. Each key has a **valid bit**: when set, the key becomes completely inaccessible to software. The keys are always read out into hidden registers by hardware during startup so that hardware can perform key comparisons without exposing the keys to software.

Pages can require specific keys for some page permissions. To unlock the page, the user writes their key to a write-only register in the OTP block. The page remains unlocked for as long as the correct key is present in this register. To re-lock the page, erase the active key by writing zeroes to the key register.

The per-page lock config specifies the following:

- a **read key** index 1-7, or 0 if there is no read key
- a **write key** index 1-7 or 0 if there is no write key
- the **no-key state**: either Read-only or Inaccessible, state of the page when no registered key has been entered by software into the key register

The no-key state is encoded as follows:

- 0 for Read-only (lock level 1)
- 1 for Inaccessible (lock level 2).

 **TIP**

Key index 7 does not exist in the configuration. If you specify key index 7, it is guaranteed to never match.

The hardware determines the **key lock level** by comparing the entered key to the key config of the current page, as follows:

1. If no keys are registered, the key lock level is 0
2. Else if keys are registered and no matching key is entered, the key lock level is 2 or 1 depending on the "no-key state" config
3. Else if a write key is registered and present, the key lock level is 0
4. Else if a read key is registered and present, the key lock level is 1

Hardware compares the key lock level to the page's lock level for the current security domain (Secure/Non-secure) and takes **whichever is higher**. For example, if a page has been made Non-secure read-only, there is nothing a key can do to make it Non-secure writable.

There are six 128-bit access keys stored in the OTP. Keys 5 and 6 also function as the Secure debug access key and Non-secure debug access key, respectively. See [Section 3.5.9.2](#) for information on how the debug keys affect external debug access.

You might use OTP access keys if a bootloader contains OTP configuration that needs to be Secure-writable *only to the board owner*, not to general Secure software on the device.

13.5.3. Lock Encoding in OTP

Page locks are encoded as a 16-bit value. This value is stored as a pair of triple-redundant bytes, each byte occupying a 24-bit OTP row.

The lock halfword is encoded as follows:

Bits	Purpose
2:0	Write key index, or 0 if no write key
5:3	Read key index, or 0 if no read key
6	No-key state, 0=Read-only 1=Inaccessible
7	Reserved
9:8	Secure lock state (thermometer code 0 → 2)
11:10	Non-secure lock state (thermometer code 0 → 2)
13:12	PicoBoot lock state (thermometer code 0 → 2) or software-defined use if PicoBoot OTP is disabled
15:12	Reserved

13.5.4. Special Pages

The following pages require special case handling in their lock checks:

- The lock word region itself (pages 62 and 63)
 - Lock words are always world-readable
 - Lock words are writable by Secure code if the lock word itself permits Secure writes
 - Consequently, lock words 62 and 63 are considered "spare", since they do not protect pages 62 and 63; the page 63 lock word is repurposed for the RMA flag
- The hardware access key page (page 61)
 - Contains OTP access keys and debug access keys
 - Each key also has a valid bit (`rbit`)
 - Page 61 (key page) has all of the usual protections from the page 61 lock word
 - If a key's valid bit is set, that key is inaccessible; the converse is not necessarily true

Page 0, known as the **chip info page**, is *not* a special page. Raspberry Pi sets page 0 to read-only during factory test, after writing chip identification and calibration values.

13.5.5. Permissions of Blank Devices

Each RP2350 device has some information programmed during manufacturing test. At this time, a small number of hard page lock bits are also programmed:

- Page 0, which contains chip information, is read-only for all accesses.
- Pages 1 and 2, which contain boot config and boot key fingerprints, are read-only for Non-secure access, read-write for Secure access, and read-write for bootloader access.
- Page 63, which contains the RMA flag, is read-only for Non-secure and bootloader access, and read-write for Secure access.

This minimal set of default permissions on blank devices avoids certain classes of security model violation, like Non-secure code being able to brick the chip by overwriting the boot key fingerprints with invalid data. In this context, the term *blank device* refers to a device that has gone through manufacturing test programming, but has not had any other OTP bits programmed by the user.

You can add additional soft or hard locks to these default permissions, with the exception of page 0. Page 0 cannot be hard-locked, since the secure read-only permission prevents a user from altering its lock word.

Lock words 2 through 61, covering all pages with user-defined contents, are left unprogrammed. On a blank device, these pages are fully accessible from all domains. Before launching any Non-secure application, you should apply at least a soft read-only lock to all pages that are not explicitly allocated for Non-secure use. To do this, write to [SW_LOCK2](#) through [SW_LOCK61](#). For devices that you don't expect to RMA, such as those that have passed board-level manufacturing tests, you should lock secure writes to the RMA flag.

13.6. Error Correction Code (ECC)

ECC-protected rows store data in the following structure, accessible through a raw alias:

- Bits [23:22](#): bit repair by polarity (BRP) flag
- Bits [21:16](#): modified Hamming ECC code
- Bits [15:0](#) (the 16 LSBs): data

RP2350 stores the following error correction data in the 8 MSBs of each 24-bit row:

- a 6-bit modified Hamming code ECC, providing single-error-correct and double-error-detect capabilities
- 2 bits of bit repair by polarity (BRP), which supports inverting the entire row at programming time to repair a single set bit that should be clear

Writes first encode ECC, then BRP. Reads first decode BRP, then ECC. When reading through an ECC data alias ([Section 13.1](#)), hardware performs correction transparently. ECC programming operations (writes) automatically generate ECC bits when you use the bootrom [otp_access](#) API ([Section 5.4.11.4](#)).

ECC is not suitable for data that mutates one bit at a time, since the ECC value is derived from the entire 16-bit data value. When storing data without ECC, use another form of redundancy, such as 3-way majority vote.

13.6.1. Bit repair by polarity (BRP)

Bit repair by polarity (BRP) compensates for a single bit present at time of programming.

When programming a row, hardware or software first calculates a 24-bit target value consisting of:

- Two zeroes in bits [23:22](#)
- 6-bit Hamming ECC in bits [21:16](#)
- 16-bit data value in bits [15:0](#)

Before programming, an OTP row should contain all zeros. However, sometimes OTP rows contain a single bit that is already set to [1](#), either due to manufacturing flaws or previous programming. If a bit is already set ([1](#)) in an OTP row before programming, BRP checks the status of the corresponding bit in the target value. BRP compensates for this single set bit in one of two ways, depending on the corresponding value in the target value:

- If the bit is clear ([0](#)), BRP inverts the target row and writes two ones in bits [23:22](#).
- If the bit is set ([1](#)), BRP does not invert the target row, leaving two zeroes in bits [23:22](#).

When you read an OTP value through an ECC alias ([Section 13.1](#)), BRP checks for two ones in bits [23:22](#). When both bits 23 and 22 are set, BRP inverts the entire row before passing it to the modified Hamming code stage.

BRP makes it possible to store any 22-bit value in a row that initially has at most one bit set, preserving the correction margin of the modified Hamming code. During manufacturing test, hardware scans the entire OTP array to ensure no rows contain more than one pre-set bit.

13.6.2. Modified Hamming ECC

ECC generates six **parity bits** based on the data value stored in bits 15:0 of an OTP row. When programming a row, ECC generates those six parity bits and includes them in the target value as bits 21:16. This code consists of:

- A 5-bit Hamming code that identifies single-bit errors
- An even parity bit which allows two-bit errors to be detected in the Hamming code and the original 16-bit data

When you read an OTP value through an ECC alias (Section 13.1), ECC recalculates the six parity bits based on the value read from the OTP row. Then, ECC XORs the original six parity bits with the newly-calculated parity bits. This generates 6 new bits:

- the 5 LSBs are the *syndrome*, a unique bit pattern that corresponds to each possible bit flip in the data value
- the MSB distinguishes between odd and even numbers of bit flips

If all 6 bits in this value are zero, ECC did not detect an error. If the MSB is 1, the syndrome should indicate a single-bit error. ECC flips the corresponding data bit to recover from the error. If the MSB is 0, but the syndrome contains a value other than 0, the ECC detected an unrecoverable multi-bit error.

You can calculate 5-bit Hamming codes and parity bits with the following C code (adapted from the RP2350 bootrom source):

```
uint32_t evenParity(uint32_t input) {
    uint32_t rc = 0;
    while (input) {
        rc ^= input & 1;
        input >>= 1;
    }
    return rc;
}

const uint32_t otpEccParityTable[6] = {
    0b000001010110101011011,
    0b000000011011001101101,
    0b0000001100011110001110,
    0b000000000011111110000,
    0b000000111110000000000,
    0b01111111111111111111111
};

uint32_t sOtpCalculateEcc(uint16_t x) {
    uint32_t p = x;
    for (uint i = 0; i < 6; ++i) {
        p |= evenParity(p & otpEccParityTable[i]) << (16 + i);
    }
    return p;
}
```

13.7. Device Decommissioning (RMA)

Decommissioning refers to destroying a device's sensitive contents and restoring some test or debug functionality when a device reaches the end of its security lifecycle. The OTP hardware can't actually destroy user data without circumventing write protection in some way. Instead, decommissioning is implemented with the **RMA flag**, which modifies devices in the following ways:

- re-enables factory test JTAG which is otherwise disabled by the secure boot critical flag

- makes pages 3 through 61 inaccessible

The RMA flag doesn't change permissions for page 0 (manufacturing data), pages 1 and 2 (boot configuration), page 61 (OTP access keys), or pages 62 and 63 (locks).

The RMA flag is encoded in a spare bit of the page 63 lock word. This lock word would otherwise be unused, since page 63 is one of the lock pages; consequently, it is not protected by a lock word. Instead, each lock word protects itself.

Like all other lock words, the page 63 lock word is protected by its own locks, which means it can be hard- and soft-locked to prevent the RMA flag being set. Locking the RMA flag makes it impossible to re-enable the factory JTAG interface if any of [CRIT1.SECURE_BOOT_ENABLE](#), [CRIT1.DEBUG_DISABLE](#) or [CRIT1.SECURE_DEBUG_DISABLE](#) is set. This makes it impossible for Raspberry Pi to re-test such devices if they are returned for fault analysis.

❗ IMPORTANT

Setting the RMA flag does not destroy OTP contents, it merely renders it inaccessible. The design intent is for this to be irreversible, but hardware is never perfect. This is something the user's threat model must account for when programming the RMA flag on devices with sensitive OTP contents – for example, by personalising per-device OTP secrets to avoid class breaks if an attacker is able to retrieve the keys.

13.8. List of Registers

The OTP control registers start at a base address of [0x40120000](#) (defined as [OTP_BASE](#) in the SDK).

Table 1292. List of OTP registers

Offset	Name	Info
0x000	SW_LOCK0	Software lock register for page 0.
0x004	SW_LOCK1	Software lock register for page 1.
0x008	SW_LOCK2	Software lock register for page 2.
0x00c	SW_LOCK3	Software lock register for page 3.
0x010	SW_LOCK4	Software lock register for page 4.
0x014	SW_LOCK5	Software lock register for page 5.
0x018	SW_LOCK6	Software lock register for page 6.
0x01c	SW_LOCK7	Software lock register for page 7.
0x020	SW_LOCK8	Software lock register for page 8.
0x024	SW_LOCK9	Software lock register for page 9.
0x028	SW_LOCK10	Software lock register for page 10.
0x02c	SW_LOCK11	Software lock register for page 11.
0x030	SW_LOCK12	Software lock register for page 12.
0x034	SW_LOCK13	Software lock register for page 13.
0x038	SW_LOCK14	Software lock register for page 14.
0x03c	SW_LOCK15	Software lock register for page 15.
0x040	SW_LOCK16	Software lock register for page 16.
0x044	SW_LOCK17	Software lock register for page 17.
0x048	SW_LOCK18	Software lock register for page 18.
0x04c	SW_LOCK19	Software lock register for page 19.

Offset	Name	Info
0x050	SW_LOCK20	Software lock register for page 20.
0x054	SW_LOCK21	Software lock register for page 21.
0x058	SW_LOCK22	Software lock register for page 22.
0x05c	SW_LOCK23	Software lock register for page 23.
0x060	SW_LOCK24	Software lock register for page 24.
0x064	SW_LOCK25	Software lock register for page 25.
0x068	SW_LOCK26	Software lock register for page 26.
0x06c	SW_LOCK27	Software lock register for page 27.
0x070	SW_LOCK28	Software lock register for page 28.
0x074	SW_LOCK29	Software lock register for page 29.
0x078	SW_LOCK30	Software lock register for page 30.
0x07c	SW_LOCK31	Software lock register for page 31.
0x080	SW_LOCK32	Software lock register for page 32.
0x084	SW_LOCK33	Software lock register for page 33.
0x088	SW_LOCK34	Software lock register for page 34.
0x08c	SW_LOCK35	Software lock register for page 35.
0x090	SW_LOCK36	Software lock register for page 36.
0x094	SW_LOCK37	Software lock register for page 37.
0x098	SW_LOCK38	Software lock register for page 38.
0x09c	SW_LOCK39	Software lock register for page 39.
0x0a0	SW_LOCK40	Software lock register for page 40.
0x0a4	SW_LOCK41	Software lock register for page 41.
0x0a8	SW_LOCK42	Software lock register for page 42.
0x0ac	SW_LOCK43	Software lock register for page 43.
0x0b0	SW_LOCK44	Software lock register for page 44.
0x0b4	SW_LOCK45	Software lock register for page 45.
0x0b8	SW_LOCK46	Software lock register for page 46.
0x0bc	SW_LOCK47	Software lock register for page 47.
0x0c0	SW_LOCK48	Software lock register for page 48.
0x0c4	SW_LOCK49	Software lock register for page 49.
0x0c8	SW_LOCK50	Software lock register for page 50.
0x0cc	SW_LOCK51	Software lock register for page 51.
0x0d0	SW_LOCK52	Software lock register for page 52.
0x0d4	SW_LOCK53	Software lock register for page 53.
0x0d8	SW_LOCK54	Software lock register for page 54.
0x0dc	SW_LOCK55	Software lock register for page 55.

Offset	Name	Info
0x0e0	SW_LOCK56	Software lock register for page 56.
0x0e4	SW_LOCK57	Software lock register for page 57.
0x0e8	SW_LOCK58	Software lock register for page 58.
0x0ec	SW_LOCK59	Software lock register for page 59.
0x0f0	SW_LOCK60	Software lock register for page 60.
0x0f4	SW_LOCK61	Software lock register for page 61.
0x0f8	SW_LOCK62	Software lock register for page 62.
0x0fc	SW_LOCK63	Software lock register for page 63.
0x100	SBPI_INSTR	Dispatch instructions to the SBPI interface, used for programming the OTP fuses.
0x104	SBPI_WDATA_0	SBPI write payload bytes 3..0
0x108	SBPI_WDATA_1	SBPI write payload bytes 7..4
0x10c	SBPI_WDATA_2	SBPI write payload bytes 11..8
0x110	SBPI_WDATA_3	SBPI write payload bytes 15..12
0x114	SBPI_RDATA_0	Read payload bytes 3..0. Once read, the data in the register will automatically clear to 0.
0x118	SBPI_RDATA_1	Read payload bytes 7..4. Once read, the data in the register will automatically clear to 0.
0x11c	SBPI_RDATA_2	Read payload bytes 11..8. Once read, the data in the register will automatically clear to 0.
0x120	SBPI_RDATA_3	Read payload bytes 15..12. Once read, the data in the register will automatically clear to 0.
0x124	SBPI_STATUS	
0x128	USR	Controls for APB data read interface (USER interface)
0x12c	DBG	Debug for OTP power-on state machine
0x134	BIST	During BIST, count address locations that have at least one leaky bit
0x138	CRT_KEY_W0	Word 0 (bits 31..0) of the key. Write only, read returns 0x0
0x13c	CRT_KEY_W1	Word 1 (bits 63..32) of the key. Write only, read returns 0x0
0x140	CRT_KEY_W2	Word 2 (bits 95..64) of the key. Write only, read returns 0x0
0x144	CRT_KEY_W3	Word 3 (bits 127..96) of the key. Write only, read returns 0x0
0x148	CRITICAL	Quickly check values of critical flags read during boot up
0x14c	KEY_VALID	Which keys were valid (enrolled) at boot time
0x150	DEBUGEN	Enable a debug feature that has been disabled. Debug features are disabled if one of the relevant critical boot flags is set in OTP (DEBUG_DISABLE or SECURE_DEBUG_DISABLE), OR if a debug key is marked valid in OTP, and the matching key value has not been supplied over SWD.

Offset	Name	Info
0x154	DEBUGEN_LOCK	Write 1s to lock corresponding bits in DEBUGEN. This register is reset by the processor cold reset.
0x158	ARCHSEL	Architecture select (Arm/RISC-V), applied on next processor reset. The default and allowable values of this register are constrained by the critical boot flags.
0x15c	ARCHSEL_STATUS	Get the current architecture select state of each core. Cores sample the current value of the ARCHSEL register when their warm reset is released, at which point the corresponding bit in this register will also update.
0x160	BOOTDIS	Tell the bootrom to ignore scratch register boot vectors (both power manager and watchdog) on the next power up.
0x164	INTR	Raw Interrupts
0x168	INTE	Interrupt Enable
0x16c	INTF	Interrupt Force
0x170	INTS	Interrupt status after masking & forcing

OTP: SW_LOCK0, SW_LOCK1, ..., SW_LOCK62, SW_LOCK63 Registers

Offsets: 0x000, 0x004, ..., 0x0f8, 0x0fc

Description

Software lock register for page N .

Locks are initialised from the OTP lock pages at reset. This register can be written to further advance the lock state of each page (until next reset), and read to check the current lock state of a page.

Table 1293.
SW_LOCK0,
SW_LOCK1, ...,
SW_LOCK62,
SW_LOCK63 Registers

Bits	Name	Description	Type	Reset
31:4	Reserved.	-	-	-
3:2	NSEC	Non-secure lock status. Writes are OR'd with the current value. 0x0 → read_write 0x1 → read_only 0x3 → inaccessible	RW	-
1:0	SEC	Secure lock status. Writes are OR'd with the current value. This field is read-only to Non-secure code. 0x0 → read_write 0x1 → read_only 0x3 → inaccessible	RW	-

OTP: SBPI_INSTR Register

Offset: 0x100

Description

Dispatch instructions to the SBPI interface, used for programming the OTP fuses.

Table 1294.
SBPI_INSTR Register

Bits	Name	Description	Type	Reset
31	Reserved.	-	-	-
30	EXEC	Execute instruction	SC	0x0

Bits	Name	Description	Type	Reset
29	IS_WR	Payload type is write	RW	0x0
28	HAS_PAYLOAD	Instruction has payload (data to be written or to be read)	RW	0x0
27:24	PAYLOAD_SIZE_M1	Instruction payload size in bytes minus 1	RW	0x0
23:16	TARGET	Instruction target, it can be PMC (0x3a) or DAP (0x02)	RW	0x00
15:8	CMD		RW	0x00
7:0	SHORT_WDATA	wdata to be used only when payload_size_m1=0	RW	0x00

OTP: SBPI_WDATA_0 Register

Offset: 0x104

Table 1295.
SBPI_WDATA_0
Register

Bits	Description	Type	Reset
31:0	SBPI write payload bytes 3..0	RW	0x00000000

OTP: SBPI_WDATA_1 Register

Offset: 0x108

Table 1296.
SBPI_WDATA_1
Register

Bits	Description	Type	Reset
31:0	SBPI write payload bytes 7..4	RW	0x00000000

OTP: SBPI_WDATA_2 Register

Offset: 0x10c

Table 1297.
SBPI_WDATA_2
Register

Bits	Description	Type	Reset
31:0	SBPI write payload bytes 11..8	RW	0x00000000

OTP: SBPI_WDATA_3 Register

Offset: 0x110

Table 1298.
SBPI_WDATA_3
Register

Bits	Description	Type	Reset
31:0	SBPI write payload bytes 15..12	RW	0x00000000

OTP: SBPI_RDATA_0 Register

Offset: 0x114

Table 1299.
SBPI_RDATA_0
Register

Bits	Description	Type	Reset
31:0	Read payload bytes 3..0. Once read, the data in the register will automatically clear to 0.	RO	0x00000000

OTP: SBPI_RDATA_1 Register

Offset: 0x118

Table 1300.
SBPI_RDATA_1
Register

Bits	Description	Type	Reset
31:0	Read payload bytes 7..4. Once read, the data in the register will automatically clear to 0.	RO	0x00000000

OTP: SBPI_RDATA_2 Register

Offset: 0x11c

Table 1301.
SBPI_RDATA_2
Register

Bits	Description	Type	Reset
31:0	Read payload bytes 11..8. Once read, the data in the register will automatically clear to 0.	RO	0x00000000

OTP: SBPI_RDATA_3 Register

Offset: 0x120

Table 1302.
SBPI_RDATA_3
Register

Bits	Description	Type	Reset
31:0	Read payload bytes 15..12. Once read, the data in the register will automatically clear to 0.	RO	0x00000000

OTP: SBPI_STATUS Register

Offset: 0x124

Table 1303.
SBPI_STATUS Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	MISO	SBPI MISO (master in - slave out): response from SBPI	RO	-
15:13	Reserved.	-	-	-
12	FLAG	SBPI flag	RO	-
11:9	Reserved.	-	-	-
8	INSTR_MISS	Last instruction missed (dropped), as the previous has not finished running	WC	0x0
7:5	Reserved.	-	-	-
4	INSTR_DONE	Last instruction done	WC	0x0
3:1	Reserved.	-	-	-
0	RDATA_VLD	Read command has returned data	WC	0x0

OTP: USR Register

Offset: 0x128

Description

Controls for APB data read interface (USER interface)

Table 1304. USR
Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	PD	Power-down; 1 disables current reference. Must be 0 to read data from the OTP.	RW	0x0
3:1	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
0	DCTRL	1 enables USER interface; 0 disables USER interface (enables SBPI). This bit must be cleared before performing any SBPI access, such as when programming the OTP. The APB data read interface (USER interface) will be inaccessible during this time, and will return a bus error if any read is attempted.	RW	0x1

OTP: DBG Register

Offset: 0x12c

Description

Debug for OTP power-on state machine

Table 1305. DBG Register

Bits	Name	Description	Type	Reset
31:13	Reserved.	-	-	-
12	CUSTOMER_RMA_FLAG	The chip is in RMA mode	RO	-
11:8	Reserved.	-	-	-
7:4	PSM_STATE	Monitor the PSM FSM's state	RO	-
3	ROSC_UP	Ring oscillator is up and running	RO	-
2	ROSC_UP_SEEN	Ring oscillator was seen up and running	WC	0x0
1	BOOT_DONE	PSM boot done status flag	RO	-
0	PSM_DONE	PSM done status flag	RO	-

OTP: BIST Register

Offset: 0x134

Description

During BIST, count address locations that have at least one leaky bit

Table 1306. BIST Register

Bits	Name	Description	Type	Reset
31	Reserved.	-	-	-
30	CNT_FAIL	Flag if the count of address locations with at least one leaky bit exceeds cnt_max	RO	-
29	CNT_CLR	Clear counter before use	SC	0x0
28	CNT_ENA	Enable the counter before the BIST function is initiated	RW	0x0
27:16	CNT_MAX	The cnt_fail flag will be set if the number of leaky locations exceeds this number	RW	0xffff
15:13	Reserved.	-	-	-
12:0	CNT	Number of locations that have at least one leaky bit. Note: This count is true only if the BIST was initiated without the fix option.	RO	-

OTP: CRT_KEY_W0 Register

Offset: 0x138

Table 1307.
CRT_KEY_W0 Register

Bits	Description	Type	Reset
31:0	Word 0 (bits 31..0) of the key. Write only, read returns 0x0	WO	0x00000000

OTP: CRT_KEY_W1 Register

Offset: 0x13c

Table 1308.
CRT_KEY_W1 Register

Bits	Description	Type	Reset
31:0	Word 1 (bits 63..32) of the key. Write only, read returns 0x0	WO	0x00000000

OTP: CRT_KEY_W2 Register

Offset: 0x140

Table 1309.
CRT_KEY_W2 Register

Bits	Description	Type	Reset
31:0	Word 2 (bits 95..64) of the key. Write only, read returns 0x0	WO	0x00000000

OTP: CRT_KEY_W3 Register

Offset: 0x144

Table 1310.
CRT_KEY_W3 Register

Bits	Description	Type	Reset
31:0	Word 3 (bits 127..96) of the key. Write only, read returns 0x0	WO	0x00000000

OTP: CRITICAL Register

Offset: 0x148

Description

Quickly check values of critical flags read during boot up

Table 1311. CRITICAL Register

Bits	Name	Description	Type	Reset
31:18	Reserved.	-	-	-
17	RISCV_DISABLE		RO	0x0
16	ARM_DISABLE		RO	0x0
15:7	Reserved.	-	-	-
6:5	GLITCH_DETECTOR_SENS		RO	0x0
4	GLITCH_DETECTOR_ENABLE		RO	0x0
3	DEFAULT_ARCHSEL		RO	0x0
2	DEBUG_DISABLE		RO	0x0
1	SECURE_DEBUG_DISABLE		RO	0x0
0	SECURE_BOOT_ENABLE		RO	0x0

OTP: KEY_VALID Register

Offset: 0x14c

Table 1312.
KEY_VALID Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	Which keys were valid (enrolled) at boot time	RO	0x00

OTP: DEBUGEN Register

Offset: 0x150

Description

Enable a debug feature that has been disabled. Debug features are disabled if one of the relevant critical boot flags is set in OTP (DEBUG_DISABLE or SECURE_DEBUG_DISABLE), OR if a debug key is marked valid in OTP, and the matching key value has not been supplied over SWD.

Specifically:

- The DEBUG_DISABLE flag disables all debug features. This can be fully overridden by setting all bits of this register.
- The SECURE_DEBUG_DISABLE flag disables secure processor debug. This can be fully overridden by setting the PROC0_SECURE and PROC1_SECURE bits of this register.
- If a single debug key has been registered, and no matching key value has been supplied over SWD, then all debug features are disabled. This can be fully overridden by setting all bits of this register.
- If both debug keys have been registered, and the Non-secure key's value (key 6) has been supplied over SWD, secure processor debug is disabled. This can be fully overridden by setting the PROC0_SECURE and PROC1_SECURE bits of this register.
- If both debug keys have been registered, and the Secure key's value (key 5) has been supplied over SWD, then no debug features are disabled by the key mechanism. However, note that in this case debug features may still be disabled by the critical boot flags.

Table 1313. DEBUGEN Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	MISC	Enable other debug components. Specifically, the CTI, and the APB-AP used to access the RISC-V Debug Module. These components are disabled by default if either of the debug disable critical flags is set, or if at least one debug key has been enrolled and the least secure of these enrolled key values has not been provided over SWD.	RW	0x0
7:4	Reserved.	-	-	-
3	PROC1_SECURE	Permit core 1's Mem-AP to generate Secure accesses, assuming it is enabled at all. Also enable secure debug of core 1 (SPIDEN and SPNIDEN). Secure debug of core 1 is disabled by default if the secure debug disable critical flag is set, or if at least one debug key has been enrolled and the most secure of these enrolled key values not yet provided over SWD.	RW	0x0
2	PROC1	Enable core 1's Mem-AP if it is currently disabled. The Mem-AP is disabled by default if either of the debug disable critical flags is set, or if at least one debug key has been enrolled and the least secure of these enrolled key values has not been provided over SWD.	RW	0x0

Bits	Name	Description	Type	Reset
1	PROC0_SECURE	<p>Permit core 0's Mem-AP to generate Secure accesses, assuming it is enabled at all. Also enable secure debug of core 0 (SPIDEN and SPNIDEN).</p> <p>Secure debug of core 0 is disabled by default if the secure debug disable critical flag is set, or if at least one debug key has been enrolled and the most secure of these enrolled key values not yet provided over SWD.</p> <p>Note also that core Mem-APs are unconditionally disabled when a core is switched to RISC-V mode (by setting the ARCHSEL bit and performing a warm reset of the core).</p>	RW	0x0
0	PROC0	<p>Enable core 0's Mem-AP if it is currently disabled.</p> <p>The Mem-AP is disabled by default if either of the debug disable critical flags is set, or if at least one debug key has been enrolled and the least secure of these enrolled key values has not been provided over SWD.</p> <p>Note also that core Mem-APs are unconditionally disabled when a core is switched to RISC-V mode (by setting the ARCHSEL bit and performing a warm reset of the core).</p>	RW	0x0

OTP: DEBUGEN_LOCK Register

Offset: 0x154

Description

Write 1s to lock corresponding bits in DEBUGEN. This register is reset by the processor cold reset.

Table 1314.
DEBUGEN_LOCK
Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	MISC	Write 1 to lock the MISC bit of DEBUGEN. Can't be cleared once set.	RW	0x0
7:4	Reserved.	-	-	-
3	PROC1_SECURE	Write 1 to lock the PROC1_SECURE bit of DEBUGEN. Can't be cleared once set.	RW	0x0
2	PROC1	Write 1 to lock the PROC1 bit of DEBUGEN. Can't be cleared once set.	RW	0x0
1	PROC0_SECURE	Write 1 to lock the PROC0_SECURE bit of DEBUGEN. Can't be cleared once set.	RW	0x0
0	PROC0	Write 1 to lock the PROC0 bit of DEBUGEN. Can't be cleared once set.	RW	0x0

OTP: ARCHSEL Register

Offset: 0x158

Description

Architecture select (Arm/RISC-V). The default and allowable values of this register are constrained by the critical boot flags.

This register is reset by the earliest reset in the switched core power domain (before a processor cold reset).

Cores sample their architecture select signal on a warm reset. The source of the warm reset could be the system power-up state machine, the watchdog timer, Arm SYSRESETREQ or from RISC-V hartresetreq.

Note that when an Arm core is deselected, its cold reset domain is also held in reset, since in particular the SYSRESETREQ bit becomes inaccessible once the core is deselected. Note also the RISC-V cores do not have a cold reset domain, since their corresponding controls are located in the Debug Module.

Table 1315. ARCHSEL Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	CORE1	Select architecture for core 1. 0x0 → Switch core 1 to Arm (Cortex-M33) 0x1 → Switch core 1 to RISC-V (Hazard3)	RW	0x0
0	CORE0	Select architecture for core 0. 0x0 → Switch core 0 to Arm (Cortex-M33) 0x1 → Switch core 0 to RISC-V (Hazard3)	RW	0x0

OTP: ARCHSEL_STATUS Register

Offset: 0x15c

Description

Get the current architecture select state of each core. Cores sample the current value of the ARCHSEL register when their warm reset is released, at which point the corresponding bit in this register will also update.

Table 1316. ARCHSEL_STATUS Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	CORE1	Current architecture for core 0. Updated on processor warm reset. 0x0 → Core 1 is currently Arm (Cortex-M33) 0x1 → Core 1 is currently RISC-V (Hazard3)	RO	0x0
0	CORE0	Current architecture for core 0. Updated on processor warm reset. 0x0 → Core 0 is currently Arm (Cortex-M33) 0x1 → Core 0 is currently RISC-V (Hazard3)	RO	0x0

OTP: BOOTDIS Register

Offset: 0x160

Description

Tell the bootrom to ignore scratch register boot vectors (both power manager and watchdog) on the next power up.

If an early boot stage has soft-locked some OTP pages in order to protect their contents from later stages, there is a risk that Secure code running at a later stage can unlock the pages by performing a watchdog reset that resets the OTP.

This register can be used to ensure that the bootloader runs as normal on the next power up, preventing Secure code at a later stage from accessing OTP in its unlocked state.

Should be used in conjunction with the power manager BOOTDIS register.

Table 1317. BOOTDIS Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
1	NEXT	<p>This flag always ORs writes into its current contents. It can be set but not cleared by software.</p> <p>The BOOTDIS_NEXT bit is OR'd into the BOOTDIS_NOW bit when the core is powered down. Simultaneously, the BOOTDIS_NEXT bit is cleared. Setting this bit means that the boot scratch registers will be ignored following the next core power down.</p> <p>This flag should be set by an early boot stage that has soft-locked OTP pages, to prevent later stages from unlocking it via watchdog reset.</p>	RW	0x0
0	NOW	<p>When the core is powered down, the current value of BOOTDIS_NEXT is OR'd into BOOTDIS_NOW, and BOOTDIS_NEXT is cleared.</p> <p>The bootrom checks this flag before reading the boot scratch registers. If it is set, the bootrom clears it, and ignores the BOOT registers. This prevents Secure software from diverting the boot path before a bootloader has had the chance to soft lock OTP pages containing sensitive data.</p>	WC	0x0

OTP: INTR Register

Offset: 0x164

Description

Raw Interrupts

Table 1318. INTR Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	APB_RD_NSEC_FAIL		WC	0x0
3	APB_RD_SEC_FAIL		WC	0x0
2	APB_DCTRL_FAIL		WC	0x0
1	SBPI_WR_FAIL		WC	0x0
0	SBPI_FLAG_N		RO	0x0

OTP: INTE Register

Offset: 0x168

Description

Interrupt Enable

Table 1319. INTE Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	APB_RD_NSEC_FAIL		RW	0x0
3	APB_RD_SEC_FAIL		RW	0x0

Bits	Name	Description	Type	Reset
2	APB_DCTRL_FAIL		RW	0x0
1	SBPI_WR_FAIL		RW	0x0
0	SBPI_FLAG_N		RW	0x0

OTP: INTF Register

Offset: 0x16c

Description

Interrupt Force

Table 1320. INTF Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	APB_RD_NSEC_FAIL		RW	0x0
3	APB_RD_SEC_FAIL		RW	0x0
2	APB_DCTRL_FAIL		RW	0x0
1	SBPI_WR_FAIL		RW	0x0
0	SBPI_FLAG_N		RW	0x0

OTP: INTS Register

Offset: 0x170

Description

Interrupt status after masking & forcing

Table 1321. INTS Register

Bits	Name	Description	Type	Reset
31:5	Reserved.	-	-	-
4	APB_RD_NSEC_FAIL		RO	0x0
3	APB_RD_SEC_FAIL		RO	0x0
2	APB_DCTRL_FAIL		RO	0x0
1	SBPI_WR_FAIL		RO	0x0
0	SBPI_FLAG_N		RO	0x0

13.9. Predefined OTP Data Locations

This section lists OTP locations used by either the hardware (particularly the OTP power-on state machine), the bootrom, or both. This listing is for RP2350 silicon revision A2.

OTP locations are listed by row number, not by address. When read through an ECC alias, OTP rows are spaced two bytes apart in the system address space; when read through a raw alias, OTP rows are four bytes apart. Therefore the row numbers given here should be multiplied by two or four appropriately when reading OTP contents directly from software. The OTP APIs provided by the bootrom use OTP row numbers directly, so this row-to-byte-address conversion is not necessary when accessing OTP through these APIs.

For normal (non-guarded) reads, you can access error-corrected content starting at [OTP_DATA_BASE](#) (0x40130000), and raw content starting at [OTP_DATA_RAW_BASE](#) (0x40134000). The register listings below indicate whether or not a given OTP row

contains error-corrected contents. OTP never mixes error-corrected and non-error-corrected content in the same row.

All predefined data fields have some form of redundancy. Where ECC is not viable, for instance because a location is expected to have individual bits programmed at different times, best-of-three majority vote is used instead. The only exception to this is the critical hardware flags in **CRITO** and **CRIT1**. These flags use a three-of-eight vote encoding for each individual flag: the flag is considered set when at least three bits are set out of the eight redundant bit locations. The description for each row indicates the type of redundancy.

Pages 3 through 60 (rows **0x0c0** through **0xf3f**) are free for arbitrary user content such as OTP-resident bootloaders, and Raspberry Pi will avoid allocating any of these locations for bootrom configuration if possible. This is a total of 7424 bytes of ECC-protected content.

Page 2 (rows **0x080** through **0x0bf**) is also available for user content if secure boot is disabled. It is partially available if secure boot is enabled and fewer than four boot key fingerprints are registered. This is an additional 128 ECC bytes potentially available for user content.

Pages 0, 1, and 61 through 63 are reserved for future use by Raspberry Pi. Software should avoid allocating content in these regions, even if they currently have no defined use in this data listing.

Table 1322. List of OTP_DATA registers

Offset	Name	Info
0x000	CHIPID0	Bits 15:0 of public device ID. (ECC) The CHIPID0..3 rows contain a 64-bit random identifier for this chip, which can be read from the USB bootloader PICOBOOT interface or from the <code>get_sys_info</code> ROM API. The number of random bits makes the occurrence of twins exceedingly unlikely: for example, a fleet of a hundred million devices has a 99.97% probability of no twinned IDs. This is estimated to be lower than the occurrence of process errors in the assignment of sequential random IDs, and for practical purposes CHIPID may be treated as unique.
0x001	CHIPID1	Bits 31:16 of public device ID (ECC)
0x002	CHIPID2	Bits 47:32 of public device ID (ECC)
0x003	CHIPID3	Bits 63:48 of public device ID (ECC)
0x004	RANDID0	Bits 15:0 of private per-device random number (ECC) The RANDID0..7 rows form a 128-bit random number generated during device test. This ID is not exposed through the USB PICOBOOT <code>GET_INFO</code> command or the ROM <code>get_sys_info()</code> API. However note that the USB PICOBOOT OTP access point can read the entirety of page 0, so this value is not meaningfully private unless the USB PICOBOOT interface is disabled via the <code>DISABLE_BOOTSEL_USB_PICOBOOT_IFC</code> flag in <code>BOOT_FLAGS0</code> .
0x005	RANDID1	Bits 31:16 of private per-device random number (ECC)
0x006	RANDID2	Bits 47:32 of private per-device random number (ECC)
0x007	RANDID3	Bits 63:48 of private per-device random number (ECC)
0x008	RANDID4	Bits 79:64 of private per-device random number (ECC)
0x009	RANDID5	Bits 95:80 of private per-device random number (ECC)
0x00a	RANDID6	Bits 111:96 of private per-device random number (ECC)

Offset	Name	Info
0x00b	RANDID7	Bits 127:112 of private per-device random number (ECC)
0x010	ROSC_CALIB	Ring oscillator frequency in kHz, measured during manufacturing (ECC) This is measured at 1.1 V, at room temperature, with the ROSC configuration registers in their reset state.
0x011	LPOSC_CALIB	Low-power oscillator frequency in Hz, measured during manufacturing (ECC) This is measured at 1.1V, at room temperature, with the LPOSC trim register in its reset state.
0x018	NUM_GPIOS	The number of main user GPIOs (bank 0). Should read 48 in the QFN80 package, and 30 in the QFN60 package. (ECC)
0x036	INFO_CRC0	Lower 16 bits of CRC32 of OTP addresses 0x00 through 0x6b (polynomial 0x4c11db7, input reflected, output reflected, seed all-ones, final XOR all-ones) (ECC)
0x037	INFO_CRC1	Upper 16 bits of CRC32 of OTP addresses 0x00 through 0x6b (ECC)
0x038	CRIT0	Page 0 critical boot flags (RBIT-8)
0x039	CRIT0_R1	Redundant copy of CRIT0
0x03a	CRIT0_R2	Redundant copy of CRIT0
0x03b	CRIT0_R3	Redundant copy of CRIT0
0x03c	CRIT0_R4	Redundant copy of CRIT0
0x03d	CRIT0_R5	Redundant copy of CRIT0
0x03e	CRIT0_R6	Redundant copy of CRIT0
0x03f	CRIT0_R7	Redundant copy of CRIT0
0x040	CRIT1	Page 1 critical boot flags (RBIT-8)
0x041	CRIT1_R1	Redundant copy of CRIT1
0x042	CRIT1_R2	Redundant copy of CRIT1
0x043	CRIT1_R3	Redundant copy of CRIT1
0x044	CRIT1_R4	Redundant copy of CRIT1
0x045	CRIT1_R5	Redundant copy of CRIT1
0x046	CRIT1_R6	Redundant copy of CRIT1
0x047	CRIT1_R7	Redundant copy of CRIT1
0x048	BOOT_FLAGS0	Disable/Enable boot paths/features in the RP2350 mask ROM. Disables always supersede enables. Enables are provided where there are other configurations in OTP that must be valid. (RBIT-3)
0x049	BOOT_FLAGS0_R1	Redundant copy of BOOT_FLAGS0
0x04a	BOOT_FLAGS0_R2	Redundant copy of BOOT_FLAGS0

Offset	Name	Info
0x04b	BOOT_FLAGS1	Disable/Enable boot paths/features in the RP2350 mask ROM. Disables always supersede enables. Enables are provided where there are other configurations in OTP that must be valid. (RBIT-3)
0x04c	BOOT_FLAGS1_R1	Redundant copy of BOOT_FLAGS1
0x04d	BOOT_FLAGS1_R2	Redundant copy of BOOT_FLAGS1
0x04e	DEFAULT_BOOT_VERSION0	Default boot version thermometer counter, bits 23:0 (RBIT-3)
0x04f	DEFAULT_BOOT_VERSION0_R1	Redundant copy of DEFAULT_BOOT_VERSION0
0x050	DEFAULT_BOOT_VERSION0_R2	Redundant copy of DEFAULT_BOOT_VERSION0
0x051	DEFAULT_BOOT_VERSION1	Default boot version thermometer counter, bits 47:24 (RBIT-3)
0x052	DEFAULT_BOOT_VERSION1_R1	Redundant copy of DEFAULT_BOOT_VERSION1
0x053	DEFAULT_BOOT_VERSION1_R2	Redundant copy of DEFAULT_BOOT_VERSION1
0x054	FLASH_DEVINFO	Stores information about external flash device(s). (ECC) Assumed to be valid if BOOT_FLAGS0_FLASH_DEVINFO_ENABLE is set.
0x055	FLASH_PARTITION_SLOT_SIZE	Gap between partition table slot 0 and slot 1 at the start of flash (the default size is 4096 bytes) (ECC) Enabled by the OVERRIDE_FLASH_PARTITION_SLOT_SIZE bit in BOOT_FLAGS , the size is 4096 * (value + 1)
0x056	BOOTSEL_LED_CFG	Pin configuration for LED status, used by USB bootloader. (ECC) Must be valid if BOOT_FLAGS0_ENABLE_BOOTSEL_LED is set.
0x057	BOOTSEL_PLL_CFG	Optional PLL configuration for BOOTSEL mode. (ECC)
0x058	BOOTSEL_XOSC_CFG	Non-default crystal oscillator configuration for the USB bootloader. (ECC)
0x059	USB_BOOT_FLAGS	USB boot specific feature flags (RBIT-3)
0x05a	USB_BOOT_FLAGS_R1	Redundant copy of USB_BOOT_FLAGS
0x05b	USB_BOOT_FLAGS_R2	Redundant copy of USB_BOOT_FLAGS
0x05c	USB_WHITE_LABEL_ADDR	Row index of the USB_WHITE_LABEL structure within OTP (ECC)
0x05e	OTPBOOT_SRC	OTP start row for the OTP boot image. (ECC)
0x05f	OTPBOOT_LEN	Length in rows of the OTP boot image. (ECC)
0x060	OTPBOOT_DST0	Bits 15:0 of the OTP boot image load destination (and entry point). (ECC)
0x061	OTPBOOT_DST1	Bits 31:16 of the OTP boot image load destination (and entry point). (ECC)
0x080	BOOTKEY0_0	Bits 15:0 of SHA-256 hash of boot key 0 (ECC)
0x081	BOOTKEY0_1	Bits 31:16 of SHA-256 hash of boot key 0 (ECC)
0x082	BOOTKEY0_2	Bits 47:32 of SHA-256 hash of boot key 0 (ECC)
0x083	BOOTKEY0_3	Bits 63:48 of SHA-256 hash of boot key 0 (ECC)
0x084	BOOTKEY0_4	Bits 79:64 of SHA-256 hash of boot key 0 (ECC)
0x085	BOOTKEY0_5	Bits 95:80 of SHA-256 hash of boot key 0 (ECC)

Offset	Name	Info
0x086	BOOTKEY0_6	Bits 111:96 of SHA-256 hash of boot key 0 (ECC)
0x087	BOOTKEY0_7	Bits 127:112 of SHA-256 hash of boot key 0 (ECC)
0x088	BOOTKEY0_8	Bits 143:128 of SHA-256 hash of boot key 0 (ECC)
0x089	BOOTKEY0_9	Bits 159:144 of SHA-256 hash of boot key 0 (ECC)
0x08a	BOOTKEY0_10	Bits 175:160 of SHA-256 hash of boot key 0 (ECC)
0x08b	BOOTKEY0_11	Bits 191:176 of SHA-256 hash of boot key 0 (ECC)
0x08c	BOOTKEY0_12	Bits 207:192 of SHA-256 hash of boot key 0 (ECC)
0x08d	BOOTKEY0_13	Bits 223:208 of SHA-256 hash of boot key 0 (ECC)
0x08e	BOOTKEY0_14	Bits 239:224 of SHA-256 hash of boot key 0 (ECC)
0x08f	BOOTKEY0_15	Bits 255:240 of SHA-256 hash of boot key 0 (ECC)
0x090	BOOTKEY1_0	Bits 15:0 of SHA-256 hash of boot key 1 (ECC)
0x091	BOOTKEY1_1	Bits 31:16 of SHA-256 hash of boot key 1 (ECC)
0x092	BOOTKEY1_2	Bits 47:32 of SHA-256 hash of boot key 1 (ECC)
0x093	BOOTKEY1_3	Bits 63:48 of SHA-256 hash of boot key 1 (ECC)
0x094	BOOTKEY1_4	Bits 79:64 of SHA-256 hash of boot key 1 (ECC)
0x095	BOOTKEY1_5	Bits 95:80 of SHA-256 hash of boot key 1 (ECC)
0x096	BOOTKEY1_6	Bits 111:96 of SHA-256 hash of boot key 1 (ECC)
0x097	BOOTKEY1_7	Bits 127:112 of SHA-256 hash of boot key 1 (ECC)
0x098	BOOTKEY1_8	Bits 143:128 of SHA-256 hash of boot key 1 (ECC)
0x099	BOOTKEY1_9	Bits 159:144 of SHA-256 hash of boot key 1 (ECC)
0x09a	BOOTKEY1_10	Bits 175:160 of SHA-256 hash of boot key 1 (ECC)
0x09b	BOOTKEY1_11	Bits 191:176 of SHA-256 hash of boot key 1 (ECC)
0x09c	BOOTKEY1_12	Bits 207:192 of SHA-256 hash of boot key 1 (ECC)
0x09d	BOOTKEY1_13	Bits 223:208 of SHA-256 hash of boot key 1 (ECC)
0x09e	BOOTKEY1_14	Bits 239:224 of SHA-256 hash of boot key 1 (ECC)
0x09f	BOOTKEY1_15	Bits 255:240 of SHA-256 hash of boot key 1 (ECC)
0x0a0	BOOTKEY2_0	Bits 15:0 of SHA-256 hash of boot key 2 (ECC)
0x0a1	BOOTKEY2_1	Bits 31:16 of SHA-256 hash of boot key 2 (ECC)
0x0a2	BOOTKEY2_2	Bits 47:32 of SHA-256 hash of boot key 2 (ECC)
0x0a3	BOOTKEY2_3	Bits 63:48 of SHA-256 hash of boot key 2 (ECC)
0x0a4	BOOTKEY2_4	Bits 79:64 of SHA-256 hash of boot key 2 (ECC)
0x0a5	BOOTKEY2_5	Bits 95:80 of SHA-256 hash of boot key 2 (ECC)
0x0a6	BOOTKEY2_6	Bits 111:96 of SHA-256 hash of boot key 2 (ECC)
0x0a7	BOOTKEY2_7	Bits 127:112 of SHA-256 hash of boot key 2 (ECC)
0x0a8	BOOTKEY2_8	Bits 143:128 of SHA-256 hash of boot key 2 (ECC)
0x0a9	BOOTKEY2_9	Bits 159:144 of SHA-256 hash of boot key 2 (ECC)

Offset	Name	Info
0x0aa	BOOTKEY2_10	Bits 175:160 of SHA-256 hash of boot key 2 (ECC)
0x0ab	BOOTKEY2_11	Bits 191:176 of SHA-256 hash of boot key 2 (ECC)
0x0ac	BOOTKEY2_12	Bits 207:192 of SHA-256 hash of boot key 2 (ECC)
0x0ad	BOOTKEY2_13	Bits 223:208 of SHA-256 hash of boot key 2 (ECC)
0x0ae	BOOTKEY2_14	Bits 239:224 of SHA-256 hash of boot key 2 (ECC)
0x0af	BOOTKEY2_15	Bits 255:240 of SHA-256 hash of boot key 2 (ECC)
0x0b0	BOOTKEY3_0	Bits 15:0 of SHA-256 hash of boot key 3 (ECC)
0x0b1	BOOTKEY3_1	Bits 31:16 of SHA-256 hash of boot key 3 (ECC)
0x0b2	BOOTKEY3_2	Bits 47:32 of SHA-256 hash of boot key 3 (ECC)
0x0b3	BOOTKEY3_3	Bits 63:48 of SHA-256 hash of boot key 3 (ECC)
0x0b4	BOOTKEY3_4	Bits 79:64 of SHA-256 hash of boot key 3 (ECC)
0x0b5	BOOTKEY3_5	Bits 95:80 of SHA-256 hash of boot key 3 (ECC)
0x0b6	BOOTKEY3_6	Bits 111:96 of SHA-256 hash of boot key 3 (ECC)
0x0b7	BOOTKEY3_7	Bits 127:112 of SHA-256 hash of boot key 3 (ECC)
0x0b8	BOOTKEY3_8	Bits 143:128 of SHA-256 hash of boot key 3 (ECC)
0x0b9	BOOTKEY3_9	Bits 159:144 of SHA-256 hash of boot key 3 (ECC)
0x0ba	BOOTKEY3_10	Bits 175:160 of SHA-256 hash of boot key 3 (ECC)
0x0bb	BOOTKEY3_11	Bits 191:176 of SHA-256 hash of boot key 3 (ECC)
0x0bc	BOOTKEY3_12	Bits 207:192 of SHA-256 hash of boot key 3 (ECC)
0x0bd	BOOTKEY3_13	Bits 223:208 of SHA-256 hash of boot key 3 (ECC)
0x0be	BOOTKEY3_14	Bits 239:224 of SHA-256 hash of boot key 3 (ECC)
0x0bf	BOOTKEY3_15	Bits 255:240 of SHA-256 hash of boot key 3 (ECC)
0xf48	KEY1_0	Bits 15:0 of OTP access key 1 (ECC)
0xf49	KEY1_1	Bits 31:16 of OTP access key 1 (ECC)
0xf4a	KEY1_2	Bits 47:32 of OTP access key 1 (ECC)
0xf4b	KEY1_3	Bits 63:48 of OTP access key 1 (ECC)
0xf4c	KEY1_4	Bits 79:64 of OTP access key 1 (ECC)
0xf4d	KEY1_5	Bits 95:80 of OTP access key 1 (ECC)
0xf4e	KEY1_6	Bits 111:96 of OTP access key 1 (ECC)
0xf4f	KEY1_7	Bits 127:112 of OTP access key 1 (ECC)
0xf50	KEY2_0	Bits 15:0 of OTP access key 2 (ECC)
0xf51	KEY2_1	Bits 31:16 of OTP access key 2 (ECC)
0xf52	KEY2_2	Bits 47:32 of OTP access key 2 (ECC)
0xf53	KEY2_3	Bits 63:48 of OTP access key 2 (ECC)
0xf54	KEY2_4	Bits 79:64 of OTP access key 2 (ECC)
0xf55	KEY2_5	Bits 95:80 of OTP access key 2 (ECC)

Offset	Name	Info
0xf56	KEY2_6	Bits 111:96 of OTP access key 2 (ECC)
0xf57	KEY2_7	Bits 127:112 of OTP access key 2 (ECC)
0xf58	KEY3_0	Bits 15:0 of OTP access key 3 (ECC)
0xf59	KEY3_1	Bits 31:16 of OTP access key 3 (ECC)
0xf5a	KEY3_2	Bits 47:32 of OTP access key 3 (ECC)
0xf5b	KEY3_3	Bits 63:48 of OTP access key 3 (ECC)
0xf5c	KEY3_4	Bits 79:64 of OTP access key 3 (ECC)
0xf5d	KEY3_5	Bits 95:80 of OTP access key 3 (ECC)
0xf5e	KEY3_6	Bits 111:96 of OTP access key 3 (ECC)
0xf5f	KEY3_7	Bits 127:112 of OTP access key 3 (ECC)
0xf60	KEY4_0	Bits 15:0 of OTP access key 4 (ECC)
0xf61	KEY4_1	Bits 31:16 of OTP access key 4 (ECC)
0xf62	KEY4_2	Bits 47:32 of OTP access key 4 (ECC)
0xf63	KEY4_3	Bits 63:48 of OTP access key 4 (ECC)
0xf64	KEY4_4	Bits 79:64 of OTP access key 4 (ECC)
0xf65	KEY4_5	Bits 95:80 of OTP access key 4 (ECC)
0xf66	KEY4_6	Bits 111:96 of OTP access key 4 (ECC)
0xf67	KEY4_7	Bits 127:112 of OTP access key 4 (ECC)
0xf68	KEY5_0	Bits 15:0 of OTP access key 5 (ECC)
0xf69	KEY5_1	Bits 31:16 of OTP access key 5 (ECC)
0xf6a	KEY5_2	Bits 47:32 of OTP access key 5 (ECC)
0xf6b	KEY5_3	Bits 63:48 of OTP access key 5 (ECC)
0xf6c	KEY5_4	Bits 79:64 of OTP access key 5 (ECC)
0xf6d	KEY5_5	Bits 95:80 of OTP access key 5 (ECC)
0xf6e	KEY5_6	Bits 111:96 of OTP access key 5 (ECC)
0xf6f	KEY5_7	Bits 127:112 of OTP access key 5 (ECC)
0xf70	KEY6_0	Bits 15:0 of OTP access key 6 (ECC)
0xf71	KEY6_1	Bits 31:16 of OTP access key 6 (ECC)
0xf72	KEY6_2	Bits 47:32 of OTP access key 6 (ECC)
0xf73	KEY6_3	Bits 63:48 of OTP access key 6 (ECC)
0xf74	KEY6_4	Bits 79:64 of OTP access key 6 (ECC)
0xf75	KEY6_5	Bits 95:80 of OTP access key 6 (ECC)
0xf76	KEY6_6	Bits 111:96 of OTP access key 6 (ECC)
0xf77	KEY6_7	Bits 127:112 of OTP access key 6 (ECC)
0xf79	KEY1_VALID	Valid flag for key 1.
0xf7a	KEY2_VALID	Valid flag for key 2.

Offset	Name	Info
0xf7b	KEY3_VALID	Valid flag for key 3.
0xf7c	KEY4_VALID	Valid flag for key 4.
0xf7d	KEY5_VALID	Valid flag for key 5.
0xf7e	KEY6_VALID	Valid flag for key 6.
0xf80	PAGE0_LOCK0	Lock configuration LSBs for page 0 (rows 0x0 through 0x3f).
0xf81	PAGE0_LOCK1	Lock configuration MSBs for page 0 (rows 0x0 through 0x3f).
0xf82	PAGE1_LOCK0	Lock configuration LSBs for page 1 (rows 0x40 through 0x7f).
0xf83	PAGE1_LOCK1	Lock configuration MSBs for page 1 (rows 0x40 through 0x7f).
0xf84	PAGE2_LOCK0	Lock configuration LSBs for page 2 (rows 0x80 through 0xbff).
0xf85	PAGE2_LOCK1	Lock configuration MSBs for page 2 (rows 0x80 through 0xbff).
0xf86	PAGE3_LOCK0	Lock configuration LSBs for page 3 (rows 0xc0 through 0xff).
0xf87	PAGE3_LOCK1	Lock configuration MSBs for page 3 (rows 0xc0 through 0xff).
0xf88	PAGE4_LOCK0	Lock configuration LSBs for page 4 (rows 0x100 through 0x13f).
0xf89	PAGE4_LOCK1	Lock configuration MSBs for page 4 (rows 0x100 through 0x13f).
0xf8a	PAGE5_LOCK0	Lock configuration LSBs for page 5 (rows 0x140 through 0x17f).
0xf8b	PAGE5_LOCK1	Lock configuration MSBs for page 5 (rows 0x140 through 0x17f).
0xf8c	PAGE6_LOCK0	Lock configuration LSBs for page 6 (rows 0x180 through 0x1bf).
0xf8d	PAGE6_LOCK1	Lock configuration MSBs for page 6 (rows 0x180 through 0x1bf).
0xf8e	PAGE7_LOCK0	Lock configuration LSBs for page 7 (rows 0x1c0 through 0x1ff).
0xf8f	PAGE7_LOCK1	Lock configuration MSBs for page 7 (rows 0x1c0 through 0x1ff).
0xf90	PAGE8_LOCK0	Lock configuration LSBs for page 8 (rows 0x200 through 0x23f).
0xf91	PAGE8_LOCK1	Lock configuration MSBs for page 8 (rows 0x200 through 0x23f).
0xf92	PAGE9_LOCK0	Lock configuration LSBs for page 9 (rows 0x240 through 0x27f).
0xf93	PAGE9_LOCK1	Lock configuration MSBs for page 9 (rows 0x240 through 0x27f).
0xf94	PAGE10_LOCK0	Lock configuration LSBs for page 10 (rows 0x280 through 0x2bf).
0xf95	PAGE10_LOCK1	Lock configuration MSBs for page 10 (rows 0x280 through 0x2bf).
0xf96	PAGE11_LOCK0	Lock configuration LSBs for page 11 (rows 0x2c0 through 0x2ff).
0xf97	PAGE11_LOCK1	Lock configuration MSBs for page 11 (rows 0x2c0 through 0x2ff).
0xf98	PAGE12_LOCK0	Lock configuration LSBs for page 12 (rows 0x300 through 0x33f).
0xf99	PAGE12_LOCK1	Lock configuration MSBs for page 12 (rows 0x300 through 0x33f).
0xf9a	PAGE13_LOCK0	Lock configuration LSBs for page 13 (rows 0x340 through 0x37f).

Offset	Name	Info
0xf9b	PAGE13_LOCK1	Lock configuration MSBs for page 13 (rows 0x340 through 0x37f).
0xf9c	PAGE14_LOCK0	Lock configuration LSBs for page 14 (rows 0x380 through 0x3bf).
0xf9d	PAGE14_LOCK1	Lock configuration MSBs for page 14 (rows 0x380 through 0x3bf).
0xf9e	PAGE15_LOCK0	Lock configuration LSBs for page 15 (rows 0x3c0 through 0x3ff).
0xf9f	PAGE15_LOCK1	Lock configuration MSBs for page 15 (rows 0x3c0 through 0x3ff).
0xfa0	PAGE16_LOCK0	Lock configuration LSBs for page 16 (rows 0x400 through 0x43f).
0xfa1	PAGE16_LOCK1	Lock configuration MSBs for page 16 (rows 0x400 through 0x43f).
0xfa2	PAGE17_LOCK0	Lock configuration LSBs for page 17 (rows 0x440 through 0x47f).
0xfa3	PAGE17_LOCK1	Lock configuration MSBs for page 17 (rows 0x440 through 0x47f).
0xfa4	PAGE18_LOCK0	Lock configuration LSBs for page 18 (rows 0x480 through 0x4bf).
0xfa5	PAGE18_LOCK1	Lock configuration MSBs for page 18 (rows 0x480 through 0x4bf).
0xfa6	PAGE19_LOCK0	Lock configuration LSBs for page 19 (rows 0x4c0 through 0x4ff).
0xfa7	PAGE19_LOCK1	Lock configuration MSBs for page 19 (rows 0x4c0 through 0x4ff).
0xfa8	PAGE20_LOCK0	Lock configuration LSBs for page 20 (rows 0x500 through 0x53f).
0xfa9	PAGE20_LOCK1	Lock configuration MSBs for page 20 (rows 0x500 through 0x53f).
0xfaa	PAGE21_LOCK0	Lock configuration LSBs for page 21 (rows 0x540 through 0x57f).
0xfb0	PAGE21_LOCK1	Lock configuration MSBs for page 21 (rows 0x540 through 0x57f).
0xfc0	PAGE22_LOCK0	Lock configuration LSBs for page 22 (rows 0x580 through 0x5bf).
0xfc1	PAGE22_LOCK1	Lock configuration MSBs for page 22 (rows 0x580 through 0x5bf).
0xfae	PAGE23_LOCK0	Lock configuration LSBs for page 23 (rows 0x5c0 through 0x5ff).
0xfb0	PAGE23_LOCK1	Lock configuration MSBs for page 23 (rows 0x5c0 through 0x5ff).
0xfb1	PAGE24_LOCK0	Lock configuration LSBs for page 24 (rows 0x600 through 0x63f).
0xfb2	PAGE24_LOCK1	Lock configuration MSBs for page 24 (rows 0x600 through 0x63f).

Offset	Name	Info
0xfb2	PAGE25_LOCK0	Lock configuration LSBs for page 25 (rows 0x640 through 0x67f).
0xfb3	PAGE25_LOCK1	Lock configuration MSBs for page 25 (rows 0x640 through 0x67f).
0xfb4	PAGE26_LOCK0	Lock configuration LSBs for page 26 (rows 0x680 through 0x6bf).
0xfb5	PAGE26_LOCK1	Lock configuration MSBs for page 26 (rows 0x680 through 0x6bf).
0xfb6	PAGE27_LOCK0	Lock configuration LSBs for page 27 (rows 0x6c0 through 0x6ff).
0xfb7	PAGE27_LOCK1	Lock configuration MSBs for page 27 (rows 0x6c0 through 0x6ff).
0xfb8	PAGE28_LOCK0	Lock configuration LSBs for page 28 (rows 0x700 through 0x73f).
0xfb9	PAGE28_LOCK1	Lock configuration MSBs for page 28 (rows 0x700 through 0x73f).
0xfbfa	PAGE29_LOCK0	Lock configuration LSBs for page 29 (rows 0x740 through 0x77f).
0xfbdb	PAGE29_LOCK1	Lock configuration MSBs for page 29 (rows 0x740 through 0x77f).
0xfc0c	PAGE30_LOCK0	Lock configuration LSBs for page 30 (rows 0x780 through 0x7bf).
0xfbdb	PAGE30_LOCK1	Lock configuration MSBs for page 30 (rows 0x780 through 0x7bf).
0xfbbe	PAGE31_LOCK0	Lock configuration LSBs for page 31 (rows 0x7c0 through 0x7ff).
0xfbfb	PAGE31_LOCK1	Lock configuration MSBs for page 31 (rows 0x7c0 through 0x7ff).
0xfc00	PAGE32_LOCK0	Lock configuration LSBs for page 32 (rows 0x800 through 0x83f).
0xfc01	PAGE32_LOCK1	Lock configuration MSBs for page 32 (rows 0x800 through 0x83f).
0xfc02	PAGE33_LOCK0	Lock configuration LSBs for page 33 (rows 0x840 through 0x87f).
0xfc03	PAGE33_LOCK1	Lock configuration MSBs for page 33 (rows 0x840 through 0x87f).
0xfc04	PAGE34_LOCK0	Lock configuration LSBs for page 34 (rows 0x880 through 0x8bf).
0xfc05	PAGE34_LOCK1	Lock configuration MSBs for page 34 (rows 0x880 through 0x8bf).
0xfc06	PAGE35_LOCK0	Lock configuration LSBs for page 35 (rows 0x8c0 through 0x8ff).
0xfc07	PAGE35_LOCK1	Lock configuration MSBs for page 35 (rows 0x8c0 through 0x8ff).
0xfc08	PAGE36_LOCK0	Lock configuration LSBs for page 36 (rows 0x900 through 0x93f).

Offset	Name	Info
0xfc9	PAGE36_LOCK1	Lock configuration MSBs for page 36 (rows 0x900 through 0x93f).
0xfcA	PAGE37_LOCK0	Lock configuration LSBs for page 37 (rows 0x940 through 0x97f).
0xfcB	PAGE37_LOCK1	Lock configuration MSBs for page 37 (rows 0x940 through 0x97f).
0xfcC	PAGE38_LOCK0	Lock configuration LSBs for page 38 (rows 0x980 through 0x9bf).
0xfcD	PAGE38_LOCK1	Lock configuration MSBs for page 38 (rows 0x980 through 0x9bf).
0xfcE	PAGE39_LOCK0	Lock configuration LSBs for page 39 (rows 0x9c0 through 0x9ff).
0xfcF	PAGE39_LOCK1	Lock configuration MSBs for page 39 (rows 0x9c0 through 0x9ff).
0xfd0	PAGE40_LOCK0	Lock configuration LSBs for page 40 (rows 0xa00 through 0xa3f).
0xfd1	PAGE40_LOCK1	Lock configuration MSBs for page 40 (rows 0xa00 through 0xa3f).
0xfd2	PAGE41_LOCK0	Lock configuration LSBs for page 41 (rows 0xa40 through 0xa7f).
0xfd3	PAGE41_LOCK1	Lock configuration MSBs for page 41 (rows 0xa40 through 0xa7f).
0xfd4	PAGE42_LOCK0	Lock configuration LSBs for page 42 (rows 0xa80 through 0xabf).
0xfd5	PAGE42_LOCK1	Lock configuration MSBs for page 42 (rows 0xa80 through 0xabf).
0xfd6	PAGE43_LOCK0	Lock configuration LSBs for page 43 (rows 0xac0 through 0xaff).
0xfd7	PAGE43_LOCK1	Lock configuration MSBs for page 43 (rows 0xac0 through 0xaff).
0xfd8	PAGE44_LOCK0	Lock configuration LSBs for page 44 (rows 0xb00 through 0xb3f).
0xfd9	PAGE44_LOCK1	Lock configuration MSBs for page 44 (rows 0xb00 through 0xb3f).
0xfdA	PAGE45_LOCK0	Lock configuration LSBs for page 45 (rows 0xb40 through 0xb7f).
0xfdB	PAGE45_LOCK1	Lock configuration MSBs for page 45 (rows 0xb40 through 0xb7f).
0xfdC	PAGE46_LOCK0	Lock configuration LSBs for page 46 (rows 0xb80 through 0xbbf).
0xfdD	PAGE46_LOCK1	Lock configuration MSBs for page 46 (rows 0xb80 through 0xbbf).
0xfdE	PAGE47_LOCK0	Lock configuration LSBs for page 47 (rows 0xbc0 through 0xbff).
0xfdF	PAGE47_LOCK1	Lock configuration MSBs for page 47 (rows 0xbc0 through 0xbff).

Offset	Name	Info
0xfe0	PAGE48_LOCK0	Lock configuration LSBs for page 48 (rows 0xc00 through 0xc3f).
0xfe1	PAGE48_LOCK1	Lock configuration MSBs for page 48 (rows 0xc00 through 0xc3f).
0xfe2	PAGE49_LOCK0	Lock configuration LSBs for page 49 (rows 0xc40 through 0xc7f).
0xfe3	PAGE49_LOCK1	Lock configuration MSBs for page 49 (rows 0xc40 through 0xc7f).
0xfe4	PAGE50_LOCK0	Lock configuration LSBs for page 50 (rows 0xc80 through 0xcbf).
0xfe5	PAGE50_LOCK1	Lock configuration MSBs for page 50 (rows 0xc80 through 0xcbf).
0xfe6	PAGE51_LOCK0	Lock configuration LSBs for page 51 (rows 0xcc0 through 0cff).
0xfe7	PAGE51_LOCK1	Lock configuration MSBs for page 51 (rows 0xcc0 through 0cff).
0xfe8	PAGE52_LOCK0	Lock configuration LSBs for page 52 (rows 0xd00 through 0xd3f).
0xfe9	PAGE52_LOCK1	Lock configuration MSBs for page 52 (rows 0xd00 through 0xd3f).
0xfea	PAGE53_LOCK0	Lock configuration LSBs for page 53 (rows 0xd40 through 0xd7f).
0xeb	PAGE53_LOCK1	Lock configuration MSBs for page 53 (rows 0xd40 through 0xd7f).
0xfec	PAGE54_LOCK0	Lock configuration LSBs for page 54 (rows 0xd80 through 0xdbf).
0xfed	PAGE54_LOCK1	Lock configuration MSBs for page 54 (rows 0xd80 through 0xdbf).
0xfee	PAGE55_LOCK0	Lock configuration LSBs for page 55 (rows 0xdc0 through 0dff).
0xfef	PAGE55_LOCK1	Lock configuration MSBs for page 55 (rows 0xdc0 through 0dff).
0xff0	PAGE56_LOCK0	Lock configuration LSBs for page 56 (rows 0xe00 through 0xe3f).
0xff1	PAGE56_LOCK1	Lock configuration MSBs for page 56 (rows 0xe00 through 0xe3f).
0xff2	PAGE57_LOCK0	Lock configuration LSBs for page 57 (rows 0xe40 through 0xe7f).
0xff3	PAGE57_LOCK1	Lock configuration MSBs for page 57 (rows 0xe40 through 0xe7f).
0xff4	PAGE58_LOCK0	Lock configuration LSBs for page 58 (rows 0xe80 through 0xebf).
0xff5	PAGE58_LOCK1	Lock configuration MSBs for page 58 (rows 0xe80 through 0xebf).
0xff6	PAGE59_LOCK0	Lock configuration LSBs for page 59 (rows 0xec0 through 0eff).

Offset	Name	Info
0xff7	PAGE59_LOCK1	Lock configuration MSBs for page 59 (rows 0xec0 through 0xeff).
0xff8	PAGE60_LOCK0	Lock configuration LSBs for page 60 (rows 0xf00 through 0xf3f).
0xff9	PAGE60_LOCK1	Lock configuration MSBs for page 60 (rows 0xf00 through 0xf3f).
0xffa	PAGE61_LOCK0	Lock configuration LSBs for page 61 (rows 0xf40 through 0xf7f).
0xffb	PAGE61_LOCK1	Lock configuration MSBs for page 61 (rows 0xf40 through 0xf7f).
0ffc	PAGE62_LOCK0	Lock configuration LSBs for page 62 (rows 0xf80 through 0xfb0).
0ffd	PAGE62_LOCK1	Lock configuration MSBs for page 62 (rows 0xf80 through 0xfb0).
0ffe	PAGE63_LOCK0	Lock configuration LSBs for page 63 (rows 0xfc0 through 0xffff).
0fff	PAGE63_LOCK1	Lock configuration MSBs for page 63 (rows 0xfc0 through 0xffff).

OTP_DATA: CHIPID0 Register

Offset: 0x000

Table 1323. CHIPID0 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 15:0 of public device ID. (ECC) The CHIPID0..3 rows contain a 64-bit random identifier for this chip, which can be read from the USB bootloader PICOBLOCK interface or from the get_sys_info ROM API. The number of random bits makes the occurrence of twins exceedingly unlikely: for example, a fleet of a hundred million devices has a 99.97% probability of no twinned IDs. This is estimated to be lower than the occurrence of process errors in the assignment of sequential random IDs, and for practical purposes CHIPID may be treated as unique.	RO	-

OTP_DATA: CHIPID1 Register

Offset: 0x001

Table 1324. CHIPID1 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 31:16 of public device ID (ECC)	RO	-

OTP_DATA: CHIPID2 Register

Offset: 0x002

Table 1325. CHIPID2 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 47:32 of public device ID (ECC)	RO	-

OTP_DATA: CHIPID3 Register

Offset: 0x003

Table 1326. CHIPID3 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 63:48 of public device ID (ECC)	RO	-

OTP_DATA: RANDID0 Register

Offset: 0x004

Table 1327. RANDID0 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 15:0 of private per-device random number (ECC) The RANDID0..7 rows form a 128-bit random number generated during device test. This ID is not exposed through the USB PICOBLOCK GET_INFO command or the ROM <code>get_sys_info()</code> API. However note that the USB PICOBLOCK OTP access point can read the entirety of page 0, so this value is not meaningfully private unless the USB PICOBLOCK interface is disabled via the DISABLE_BOOTSEL_USB_PICOBLOCK_IFC flag in BOOT_FLAGS0.	RO	-

OTP_DATA: RANDID1 Register

Offset: 0x005

Table 1328. RANDID1 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 31:16 of private per-device random number (ECC)	RO	-

OTP_DATA: RANDID2 Register

Offset: 0x006

Table 1329. RANDID2 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 47:32 of private per-device random number (ECC)	RO	-

OTP_DATA: RANDID3 Register

Offset: 0x007

Table 1330. RANDID3 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-

Bits	Description	Type	Reset
15:0	Bits 63:48 of private per-device random number (ECC)	RO	-

OTP_DATA: RANDID4 Register

Offset: 0x008

Table 1331. RANDID4 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 79:64 of private per-device random number (ECC)	RO	-

OTP_DATA: RANDID5 Register

Offset: 0x009

Table 1332. RANDID5 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 95:80 of private per-device random number (ECC)	RO	-

OTP_DATA: RANDID6 Register

Offset: 0x00a

Table 1333. RANDID6 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 111:96 of private per-device random number (ECC)	RO	-

OTP_DATA: RANDID7 Register

Offset: 0x00b

Table 1334. RANDID7 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 127:112 of private per-device random number (ECC)	RO	-

OTP_DATA: ROSC_CALIB Register

Offset: 0x010

Table 1335. ROSC_CALIB Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Ring oscillator frequency in kHz, measured during manufacturing (ECC) This is measured at 1.1 V, at room temperature, with the ROSC configuration registers in their reset state.	RO	-

OTP_DATA: LPOSC_CALIB Register

Offset: 0x011

Table 1336.
LPOSC_CALIB Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Low-power oscillator frequency in Hz, measured during manufacturing (ECC) This is measured at 1.1V, at room temperature, with the LPOSC trim register in its reset state.	RO	-

OTP_DATA: NUM_GPIOS Register

Offset: 0x018

Table 1337.
NUM_GPIOS Register

Bits	Description	Type	Reset
31:8	Reserved.	-	-
7:0	The number of main user GPIOs (bank 0). Should read 48 in the QFN80 package, and 30 in the QFN60 package. (ECC)	RO	-

OTP_DATA: INFO_CRC0 Register

Offset: 0x036

Table 1338.
INFO_CRC0 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Lower 16 bits of CRC32 of OTP addresses 0x00 through 0x6b (polynomial 0x4c11db7, input reflected, output reflected, seed all-ones, final XOR all-ones) (ECC)	RO	-

OTP_DATA: INFO_CRC1 Register

Offset: 0x037

Table 1339.
INFO_CRC1 Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Upper 16 bits of CRC32 of OTP addresses 0x00 through 0x6b (ECC)	RO	-

OTP_DATA: CRITO Register

Offset: 0x038

Description

Page 0 critical boot flags (RBIT-8)

Table 1340. CRITO Register

Bits	Name	Description	Type	Reset
31:2	Reserved.	-	-	-
1	RISCV_DISABLE	Permanently disable RISC-V processors (Hazard3)	RO	-
0	ARM_DISABLE	Permanently disable ARM processors (Cortex-M33)	RO	-

OTP_DATA: CRITO_R1, CRITO_R2, ..., CRITO_R6, CRITO_R7 Registers

Offsets: 0x039, 0x03a, ..., 0x03e, 0x03f

Table 1341. CRITO_R1,
CRITO_R2, ...,
CRITO_R6, CRITO_R7
Registers

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Redundant copy of CRITO	RO	-

OTP_DATA: CRIT1 Register

Offset: 0x040

Description

Page 1 critical boot flags (RBIT-8)

Table 1342. CRIT1
Register

Bits	Name	Description	Type	Reset
31:7	Reserved.	-	-	-
6:5	GLITCH_DETECT OR_SENS	Increase the sensitivity of the glitch detectors from their default.	RO	-
4	GLITCH_DETECT OR_ENABLE	Arm the glitch detectors to reset the system if an abnormal clock/power event is observed.	RO	-
3	BOOT_ARCH	Set the default boot architecture, 0=ARM 1=RISC-V. Ignored if ARM_DISABLE, RISCV_DISABLE or SECURE_BOOT_ENABLE is set.	RO	-
2	DEBUG_DISABLE	Disable all debug access	RO	-
1	SECURE_DEBUG_DISABLE	Disable Secure debug access	RO	-
0	SECURE_BOOT_ENABLE	Enable boot signature enforcement, and permanently disable the RISC-V cores.	RO	-

OTP_DATA: CRIT1_R1, CRIT1_R2, ..., CRIT1_R6, CRIT1_R7 Registers

Offsets: 0x041, 0x042, ..., 0x046, 0x047

Table 1343. CRIT1_R1,
CRIT1_R2, ...,
CRIT1_R6, CRIT1_R7
Registers

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Redundant copy of CRIT1	RO	-

OTP_DATA: BOOT_FLAGS0 Register

Offset: 0x048

Description

Disable/Enable boot paths/features in the RP2350 mask ROM. Disables always supersede enables. Enables are provided where there are other configurations in OTP that must be valid. (RBIT-3)

Table 1344.
BOOT_FLAGS0
Register

Bits	Name	Description	Type	Reset
31:22	Reserved.	-	-	-
21	DISABLE_SRAM_WINDOW_BOOT		RO	-
20	DISABLE_XIP_ACCESS_ON_SRAM_ENTRY	Disable all access to XIP after entering an SRAM binary. Note that this will cause bootrom APIs that access XIP to fail, including APIs that interact with the partition table.	RO	-

Bits	Name	Description	Type	Reset
19	DISABLE_BOOTSEL_UART_BOOT		RO	-
18	DISABLE_BOOTSEL_USB_PICOBOOT_IFC		RO	-
17	DISABLE_BOOTSEL_USB_MSD_IFC		RO	-
16	DISABLE_WATCHDOG_SCRATCH		RO	-
15	DISABLE_POWER_SCRATCH		RO	-
14	ENABLE OTP_BOOT	<p>Enable OTP boot. A number of OTP rows specified by OTPBOOT_LEN will be loaded, starting from OTPBOOT_SRC, into the SRAM location specified by OTPBOOT_DST1 and OTPBOOT_DST0.</p> <p>The loaded program image is stored with ECC, 16 bits per row, and must contain a valid IMAGE_DEF. Do not set this bit without first programming an image into OTP and configuring OTPBOOT_LEN, OTPBOOT_SRC, OTPBOOT_DST0 and OTPBOOT_DST1.</p> <p>Note that OTPBOOT_LEN and OTPBOOT_SRC must be even numbers of OTP rows. Equivalently, the image must be a multiple of 32 bits in size, and must start at a 32-bit-aligned address in the ECC read data address window.</p>	RO	-
13	DISABLE OTP_BOOT	Takes precedence over ENABLE OTP_BOOT.	RO	-
12	DISABLE_FLASH_BOOT		RO	-
11	ROLLBACK_REQUIRED	Require binaries to have a rollback version. Set automatically the first time a binary with a rollback version is booted.	RO	-
10	HASHED_PARTITION_TABLE	Require a partition table to be hashed (if not signed)	RO	-
9	SECURE_PARTITION_TABLE	Require a partition table to be signed	RO	-
8	DISABLE_AUTO_SWITCH_ARCH	Disable auto-switch of CPU architecture on boot when the (only) binary to be booted is for the other Arm/RISC-V architecture and both architectures are enabled	RO	-
7	SINGLE_FLASH_BINARY	Restrict flash boot path to use of a single binary at the start of flash	RO	-
6	OVERRIDE_FLASH_PARTITION_SLOT_SIZE	<p>Override the limit for default flash metadata scanning.</p> <p>The value is specified in FLASH_PARTITION_SLOT_SIZE. Make sure FLASH_PARTITION_SLOT_SIZE is valid before setting this bit</p>	RO	-

Bits	Name	Description	Type	Reset
5	FLASH_DEVINFO_ENABLE	Mark FLASH_DEVINFO as containing valid, ECC'd data which describes external flash devices.	RO	-
4	FAST_SIGCHECK_ROSC_DIV	Enable quartering of ROSC divisor during signature check, to reduce secure boot time	RO	-
3	FLASH_IO_VOLTAGE_1V8	If 1, configure the QSPI pads for 1.8 V operation when accessing flash for the first time from the bootrom, using the VOLTAGE_SELECT register for the QSPI pads bank. This slightly improves the input timing of the pads at low voltages, but does not affect their output characteristics. If 0, leave VOLTAGE_SELECT in its reset state (suitable for operation at and above 2.5 V)	RO	-
2	ENABLE_BOOTSEL_NON_DEFAULT_PLL_XOSC_CFG	Enable loading of the non-default XOSC and PLL configuration before entering BOOTSEL mode. Ensure that BOOTSEL_XOSC_CFG and BOOTSEL_PLL_CFG are correctly programmed before setting this bit. If this bit is set, user software may use the contents of BOOTSEL_PLL_CFG to calculate the expected XOSC frequency based on the fixed USB boot frequency of 48 MHz.	RO	-
1	ENABLE_BOOTSEL_LED	Enable bootloader activity LED. If set, bootsel_led_cfg is assumed to be valid	RO	-
0	DISABLE_BOOTSEL_EXEC2		RO	-

OTP_DATA: BOOT_FLAGS0_R1, BOOT_FLAGS0_R2 Registers

Offsets: 0x049, 0x04a

Table 1345.
BOOT_FLAGS0_R1,
BOOT_FLAGS0_R2
Registers

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Redundant copy of BOOT_FLAGS0	RO	-

OTP_DATA: BOOT_FLAGS1 Register

Offset: 0x04b

Description

Disable/Enable boot paths/features in the RP2350 mask ROM. Disables always supersede enables. Enables are provided where there are other configurations in OTP that must be valid. (RBIT-3)

Table 1346.
BOOT_FLAGS1
Register

Bits	Name	Description	Type	Reset
31:20	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
19	DOUBLE_TAP	<p>Enable entering BOOTSEL mode via double-tap of the RUN/RSTn pin. Adds a significant delay to boot time, as configured by DOUBLE_TAP_DELAY.</p> <p>This functions by waiting at startup (i.e. following a reset) to see if a second reset is applied soon afterward. The second reset is detected by the bootrom with help of the POWMAN_CHIP_RESET_DOUBLE_TAP flag, which is not reset by the external reset pin, and the bootrom enters BOOTSEL mode (NSBOOT) to await further instruction over USB or UART.</p>	RO	-
18:16	DOUBLE_TAP_DELAY	<p>Adjust how long to wait for a second reset when double tap BOOTSEL mode is enabled via DOUBLE_TAP. The minimum is 50 milliseconds, and each unit of this field adds an additional 50 milliseconds.</p> <p>For example, settings this field to its maximum value of 7 will cause the chip to wait for 400 milliseconds at boot to check for a second reset which requests entry to BOOTSEL mode.</p> <p>200 milliseconds (DOUBLE_TAP_DELAY=3) is a good intermediate value.</p>	RO	-
15:12	Reserved.	-	-	-
11:8	KEY_INVALID	<p>Mark a boot key as invalid, or prevent it from ever becoming valid. The bootrom will ignore any boot key marked as invalid during secure boot signature checks.</p> <p>Each bit in this field corresponds to one of the four 256-bit boot key hashes that may be stored in page 2 of the OTP.</p> <p>When provisioning boot keys, it's recommended to mark any boot key slots you don't intend to use as KEY_INVALID, so that spurious keys can not be installed at a later time.</p>	RO	-
7:4	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
3:0	KEY_VALID	<p>Mark each of the possible boot keys as valid. The bootrom will check signatures against all valid boot keys, and ignore invalid boot keys.</p> <p>Each bit in this field corresponds to one of the four 256-bit boot key hashes that may be stored in page 2 of the OTP.</p> <p>A KEY_VALID bit is ignored if the corresponding KEY_INVALID bit is set. Boot keys are considered valid only when KEY_VALID is set and KEY_INVALID is clear.</p> <p>Do not mark a boot key as KEY_VALID if it does not contain a valid SHA-256 hash of your secp256k1 public key. Verify keys after programming, before setting the KEY_VALID bits – a boot key with uncorrectable ECC faults will render your device unbootable if secure boot is enabled.</p> <p>Do not enable secure boot without first installing a valid key. This will render your device unbootable.</p>	RO	-

OTP_DATA: BOOT_FLAGS1_R1, BOOT_FLAGS1_R2 Registers

Offsets: 0x04c, 0x04d

Table 1347.
BOOT_FLAGS1_R1,
BOOT_FLAGS1_R2
Registers

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Redundant copy of BOOT_FLAGS1	RO	-

OTP_DATA: DEFAULT_BOOT_VERSION0 Register

Offset: 0x04e

Table 1348.
DEFAULT_BOOT_VERS
ION0 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Default boot version thermometer counter, bits 23:0 (RBIT-3)	RO	-

OTP_DATA: DEFAULT_BOOT_VERSION0_R1, DEFAULT_BOOT_VERSION0_R2 Registers

Offsets: 0x04f, 0x050

Table 1349.
DEFAULT_BOOT_VERS
ION0_R1,
DEFAULT_BOOT_VERS
ION0_R2 Registers

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Redundant copy of DEFAULT_BOOT_VERSION0	RO	-

OTP_DATA: DEFAULT_BOOT_VERSION1 Register

Offset: 0x051

Table 1350.
DEFAULT_BOOT_VERS
ION1 Register

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Default boot version thermometer counter, bits 47:24 (RBIT-3)	RO	-

OTP_DATA: DEFAULT_BOOT_VERSION1_R1, DEFAULT_BOOT_VERSION1_R2 Registers

Offsets: 0x052, 0x053

Table 1351.
DEFAULT_BOOT_VERS
ION1_R1,
DEFAULT_BOOT_VERS
ION1_R2 Registers

Bits	Description	Type	Reset
31:24	Reserved.	-	-
23:0	Redundant copy of DEFAULT_BOOT_VERSION1	RO	-

OTP_DATA: FLASH_DEVINFO Register

Offset: 0x054

Description

Stores information about external flash device(s). (ECC)

Assumed to be valid if BOOT_FLAGS0_FLASH_DEVINFO_ENABLE is set.

Table 1352.
FLASH_DEVINFO
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:12	CS1_SIZE	<p>The size of the flash/PSRAM device on chip select 1 (addressable at 0x11000000 through 0x11fffff).</p> <p>A value of zero is decoded as a size of zero (no device). Nonzero values are decoded as 4KiB << CS1_SIZE. For example, four megabytes is encoded with a CS1_SIZE value of 10, and 16 megabytes is encoded with a CS1_SIZE value of 12.</p> <p>When BOOT_FLAGS0_FLASH_DEVINFO_ENABLE is not set, a default of zero is used.</p> <p>0x0 → NONE 0x1 → 8K 0x2 → 16K 0x3 → 32K 0x4 → 64K 0x5 → 128K 0x6 → 256K 0x7 → 512K 0x8 → 1M 0x9 → 2M 0xa → 4M 0xb → 8M 0xc → 16M</p>	RO	-

Bits	Name	Description	Type	Reset
11:8	CS0_SIZE	<p>The size of the flash/PSRAM device on chip select 0 (addressable at 0x10000000 through 0x10fffff).</p> <p>A value of zero is decoded as a size of zero (no device). Nonzero values are decoded as 4kiB << CS0_SIZE. For example, four megabytes is encoded with a CS0_SIZE value of 10, and 16 megabytes is encoded with a CS0_SIZE value of 12.</p> <p>When <code>BOOT_FLAGS0_FLASH_DEVINFO_ENABLE</code> is not set, a default of 12 (16 MiB) is used.</p> <p>0x0 → NONE 0x1 → 8K 0x2 → 16K 0x3 → 32K 0x4 → 64K 0x5 → 128K 0x6 → 256K 0x7 → 512K 0x8 → 1M 0x9 → 2M 0xa → 4M 0xb → 8M 0xc → 16M</p>	RO	-
7	D8H_ERASE_SUPPORTED	<p>If true, all attached devices are assumed to support (or ignore, in the case of PSRAM) a block erase command with a command prefix of D8h, an erase size of 64 kiB, and a 24-bit address. Almost all 25-series flash devices support this command.</p> <p>If set, the bootrom will use the D8h erase command where it is able, to accelerate bulk erase operations. This makes flash programming faster.</p> <p>When <code>BOOT_FLAGS0_FLASH_DEVINFO_ENABLE</code> is not set, this field defaults to false.</p>	RO	-
6	Reserved.	-	-	-
5:0	CS1_GPIO	<p>Indicate a GPIO number to be used for the secondary flash chip select (CS1), which selects the external QSPI device mapped at system addresses 0x11000000 through 0x11fffff. There is no such configuration for CS0, as the primary chip select has a dedicated pin.</p> <p>On RP2350 the permissible GPIO numbers are 0, 8, 19 and 47.</p> <p>Ignored if CS1_size is zero. If CS1_SIZE is nonzero, the bootrom will automatically configure this GPIO as a second chip select upon entering the flash boot path, or entering any other path that may use the QSPI flash interface, such as <code>BOOTSEL</code> mode (nsboot).</p>	RO	-

OTP_DATA: FLASH_PARTITION_SLOT_SIZE Register

Offset: 0x055

Table 1353.
FLASH_PARTITION_SLOT_SIZE Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Gap between partition table slot 0 and slot 1 at the start of flash (the default size is 4096 bytes) (ECC) Enabled by the OVERRIDE_FLASH_PARTITION_SLOT_SIZE bit in BOOT_FLAGS, the size is 4096 * (value + 1)	RO	-

OTP_DATA: BOOTSEL_LED_CFG Register

Offset: 0x056

Description

Pin configuration for LED status, used by USB bootloader. (ECC)

Must be valid if BOOT_FLAGS0_ENABLE_BOOTSEL_LED is set.

Table 1354.
BOOTSEL_LED_CFG Register

Bits	Name	Description	Type	Reset
31:9	Reserved.	-	-	-
8	ACTIVELOW	LED is active-low. (Default: active-high.)	RO	-
7:6	Reserved.	-	-	-
5:0	PIN	GPIO index to use for bootloader activity LED.	RO	-

OTP_DATA: BOOTSEL_PLL_CFG Register

Offset: 0x057

Description

Optional PLL configuration for BOOTSEL mode. (ECC)

This should be configured to produce an exact 48 MHz based on the crystal oscillator frequency. User mode software may also use this value to calculate the expected crystal frequency based on an assumed 48 MHz PLL output.

If no configuration is given, the crystal is assumed to be 12 MHz.

The PLL frequency can be calculated as:

$$\text{PLL out} = (\text{XOSC frequency} / (\text{REFDIV}+1)) \times \text{FBDIV} / (\text{POSTDIV1} \times \text{POSTDIV2})$$

Conversely the crystal frequency can be calculated as:

$$\text{XOSC frequency} = 48 \text{ MHz} \times (\text{REFDIV}+1) \times (\text{POSTDIV1} \times \text{POSTDIV2}) / \text{FBDIV}$$

(Note the +1 on REFDIV is because the value stored in this OTP location is the actual divisor value minus one.)

Valid if and only if BOOTSEL_CFG_NONDEFAULT_CLOCK_CFG bit is set. That bit should be set only after this row and BOOTSEL_XOSC_CFG are both correctly programmed.

Table 1355.
BOOTSEL_PLL_CFG Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15	REFDIV	PLL reference divisor, minus one. Programming a value of 0 means a reference divisor of 1. Programming a value of 1 means a reference divisor of 2 (for exceptionally fast XIN inputs)	RO	-

Bits	Name	Description	Type	Reset
14:12	POSTDIV2	PLL post-divide 2 divisor, in the range 1..7 inclusive.	RO	-
11:9	POSTDIV1	PLL post-divide 1 divisor, in the range 1..7 inclusive.	RO	-
8:0	FBDIV	PLL feedback divisor, in the range 16..320 inclusive.	RO	-

OTP_DATA: BOOTSEL_XOSC_CFG Register

Offset: 0x058

Description

Non-default crystal oscillator configuration for the USB bootloader. (ECC)

These values may also be used by user code configuring the crystal oscillator.

Valid if and only if BOOTSEL_CFG_NONDEFAULT_CLOCK_CFG bit is set. That bit should be set only after this row and BOOTSEL_PLL_CFG are both correctly programmed.

Table 1356.
BOOTSEL_XOSC_CFG
Register

Bits	Name	Description	Type	Reset
31:16	Reserved.	-	-	-
15:14	RANGE	Value of the XOSC_CTRL_FREQ_RANGE register. 0x0 → 1_15MHZ 0x1 → 10_30MHZ 0x2 → 25_60MHZ 0x3 → 40_100MHZ	RO	-
13:0	STARTUP	Value of the XOSC_STARTUP register	RO	-

OTP_DATA: USB_BOOT_FLAGS Register

Offset: 0x059

Description

USB boot specific feature flags (RBIT-3)

Table 1357.
USB_BOOT_FLAGS
Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23	DP_DM_SWAP	Swap DM/DP during USB boot, to support board layouts with mirrored USB routing (deliberate or accidental).	RO	-
22	WHITE_LABEL_AD DR_VALID	valid flag for INFO_UF2_TXT_BOARD_ID_STRDEF entry of the USB_WHITE_LABEL struct (index 15)	RO	-
21:16	Reserved.	-	-	-
15	WL_INFO_UF2_TX T_BOARD_ID_STR DEF_VALID	valid flag for the USB_WHITE_LABEL_ADDR field	RO	-
14	WL_INFO_UF2_TX T_MODEL_STRDE F_VALID	valid flag for INFO_UF2_TXT_MODEL_STRDEF entry of the USB_WHITE_LABEL struct (index 14)	RO	-
13	WL_INDEX_HTM_ REDIRECT_NAME_ _STRDEF_VALID	valid flag for INDEX_HTM_REDIRECT_NAME_STRDEF entry of the USB_WHITE_LABEL struct (index 13)	RO	-

Bits	Name	Description	Type	Reset
12	WL_INDEX_HTM_REDIRECT_URL_STRDEF_VALID	valid flag for INDEX_HTM_REDIRECT_URL_STRDEF entry of the USB_WHITE_LABEL struct (index 12)	RO	-
11	WL_SCSI_INQUIRY_VERSION_STRDEF_VALID	valid flag for SCSI_INQUIRY_VERSION_STRDEF entry of the USB_WHITE_LABEL struct (index 11)	RO	-
10	WL_SCSI_INQUIRY_PRODUCT_STRDEF_VALID	valid flag for SCSI_INQUIRY_PRODUCT_STRDEF entry of the USB_WHITE_LABEL struct (index 10)	RO	-
9	WL_SCSI_INQUIRY_VENDOR_STRDEF_VALID	valid flag for SCSI_INQUIRY_VENDOR_STRDEF entry of the USB_WHITE_LABEL struct (index 9)	RO	-
8	WL_VOLUME_LABEL_STRDEF_VALID	valid flag for VOLUME_LABEL_STRDEF entry of the USB_WHITE_LABEL struct (index 8)	RO	-
7	WL_USB_CONFIG_ATTRIBUTES_MAX_POWER_VALUES_VALID	valid flag for USB_CONFIG_ATTRIBUTES_MAX_POWER_VALUES entry of the USB_WHITE_LABEL struct (index 7)	RO	-
6	WL_USB_DEVICE_SERIAL_NUMBER_STRDEF_VALID	valid flag for USB_DEVICE_SERIAL_NUMBER_STRDEF entry of the USB_WHITE_LABEL struct (index 6)	RO	-
5	WL_USB_DEVICE_PRODUCT_STRDEF_VALID	valid flag for USB_DEVICE_PRODUCT_STRDEF entry of the USB_WHITE_LABEL struct (index 5)	RO	-
4	WL_USB_DEVICE_MANUFACTURER_STRDEF_VALID	valid flag for USB_DEVICE_MANUFACTURER_STRDEF entry of the USB_WHITE_LABEL struct (index 4)	RO	-
3	WL_USB_DEVICE_LANG_ID_VALUE_VALID	valid flag for USB_DEVICE_LANG_ID_VALUE entry of the USB_WHITE_LABEL struct (index 3)	RO	-
2	WL_USB_DEVICE_SERIAL_NUMBER_VALUE_VALID	valid flag for USB_DEVICE_BCD_DEVICEVALUE entry of the USB_WHITE_LABEL struct (index 2)	RO	-
1	WL_USB_DEVICE_PID_VALUE_VALID	valid flag for USB_DEVICE_PID_VALUE entry of the USB_WHITE_LABEL struct (index 1)	RO	-
0	WL_USB_DEVICE_VID_VALUE_VALID	valid flag for USB_DEVICE_VID_VALUE entry of the USB_WHITE_LABEL struct (index 0)	RO	-

OTP_DATA: USB_BOOT_FLAGS_R1, USB_BOOT_FLAGS_R2 Registers

Offsets: 0x05a, 0x05b

Table 1358.
USB_BOOT_FLAGS_R1,
USB_BOOT_FLAGS_R2
Registers

Bits	Description	Type	Reset
31:24	Reserved.	-	-

Bits	Description	Type	Reset
23:0	Redundant copy of USB_BOOT_FLAGS	RO	-

OTP_DATA: USB_WHITE_LABEL_ADDR Register

Offset: 0x05c

Table 1359.
USB_WHITE_LABEL_A
DDR Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-

Bits	Description	Type	Reset
15:0	<p>Row index of the USB_WHITE_LABEL structure within OTP (ECC)</p> <p>The table has 16 rows, each of which are also ECC and marked valid by the corresponding valid bit in USB_BOOT_FLAGS (ECC).</p> <p>The entries are either _VALUEs where the 16 bit value is used as is, or _STRDEFs which acts as a pointers to a string value.</p> <p>The value stored in a _STRDEF is two separate bytes: The low seven bits of the first (LSB) byte indicates the number of characters in the string, and the top bit of the first (LSB) byte if set to indicate that each character in the string is two bytes (Unicode) versus one byte if unset. The second (MSB) byte represents the location of the string data, and is encoded as the number of rows from this USB_WHITE_LABEL_ADDR; i.e. the row of the start of the string is USB_WHITE_LABEL_ADDR value + msb_byte.</p> <p>In each case, the corresponding valid bit enables replacing the default value for the corresponding item provided by the boot rom.</p> <p>Note that Unicode _STRDEFs are only supported for USB_DEVICE_PRODUCT_STRDEF, USB_DEVICE_SERIAL_NUMBER_STRDEF and USB_DEVICE_MANUFACTURER_STRDEF. Unicode values will be ignored if specified for other fields, and non-unicode values for these three items will be converted to Unicode characters by setting the upper 8 bits to zero.</p> <p>Note that if the USB_WHITE_LABEL structure or the corresponding strings are not readable by BOOTSEL mode based on OTP permissions, or if alignment requirements are not met, then the corresponding default values are used.</p> <p>The index values indicate where each field is located (row USB_WHITE_LABEL_ADDR value + index):</p> <ul style="list-style-type: none"> 0x0000 → INDEX_USB_DEVICE_VID_VALUE 0x0001 → INDEX_USB_DEVICE_PID_VALUE 0x0002 → INDEX_USB_DEVICE_BCD_DEVICE_VALUE 0x0003 → INDEX_USB_DEVICE_LANG_ID_VALUE 0x0004 → INDEX_USB_DEVICE_MANUFACTURER_STRDEF 0x0005 → INDEX_USB_DEVICE_PRODUCT_STRDEF 0x0006 → INDEX_USB_DEVICE_SERIAL_NUMBER_STRDEF 0x0007 → INDEX_USB_CONFIG_ATTRIBUTES_MAX_POWER_VALUES 0x0008 → INDEX_VOLUME_LABEL_STRDEF 0x0009 → INDEX_SCSI_INQUIRY_VENDOR_STRDEF 0x000a → INDEX_SCSI_INQUIRY_PRODUCT_STRDEF 0x000b → INDEX_SCSI_INQUIRY_VERSION_STRDEF 0x000c → INDEX_INDEX_HTM_REDIRECT_URL_STRDEF 0x000d → INDEX_INDEX_HTM_REDIRECT_NAME_STRDEF 0x000e → INDEX_INFO_UF2_TXT_MODEL_STRDEF 0x000f → INDEX_INFO_UF2_TXT_BOARD_ID_STRDEF 	RO	-

OTP_DATA: OTPBOOT_SRC Register

Offset: 0x05e

Table 1360.
OTPBOOT_SRC
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	OTP start row for the OTP boot image. (ECC) If OTP boot is enabled, the bootrom will load from this location into SRAM and then directly enter the loaded image. Note that the image must be signed if SECURE_BOOT_ENABLE is set. The image itself is assumed to be ECC-protected. This must be an even number. Equivalently, the OTP boot image must start at a word-aligned location in the ECC read data address window.	RO	-

OTP_DATA: OTPBOOT_LEN Register

Offset: 0x05f

Table 1361.
OTPBOOT_LEN
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Length in rows of the OTP boot image. (ECC) OTPBOOT_LEN must be even. The total image size must be a multiple of 4 bytes (32 bits).	RO	-

OTP_DATA: OTPBOOT_DST0 Register

Offset: 0x060

Table 1362.
OTPBOOT_DST0
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 15:0 of the OTP boot image load destination (and entry point). (ECC) This must be a location in main SRAM (main SRAM is addresses 0x20000000 through 0x20082000) and must be word-aligned.	RO	-

OTP_DATA: OTPBOOT_DST1 Register

Offset: 0x061

Table 1363.
OTPBOOT_DST1
Register

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 31:16 of the OTP boot image load destination (and entry point). (ECC) This must be a location in main SRAM (main SRAM is addresses 0x20000000 through 0x20082000) and must be word-aligned.	RO	-

OTP_DATA: BOOTKEY0_0, BOOTKEY0_1, ..., BOOTKEY3_14, BOOTKEY3_15 Registers

Offsets: 0x080, 0x081, ..., 0x0be, 0x0bf

Table 1364.
BOOTKEY0_0,
BOOTKEY0_1, ...,
BOOTKEY3_14,
BOOTKEY3_15
Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits $N + 15 : N$ of SHA-256 hash of boot key K (ECC)	RO	-

OTP_DATA: KEY1_0, KEY2_0, ..., KEY5_0, KEY6_0 Registers

Offsets: 0xf48, 0xf50, ..., 0xf68, 0xf70

Table 1365. KEY1_0,
KEY2_0, ..., KEY5_0,
KEY6_0 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 15:0 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_1, KEY2_1, ..., KEY5_1, KEY6_1 Registers

Offsets: 0xf49, 0xf51, ..., 0xf69, 0xf71

Table 1366. KEY1_1,
KEY2_1, ..., KEY5_1,
KEY6_1 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 31:16 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_2, KEY2_2, ..., KEY5_2, KEY6_2 Registers

Offsets: 0xf4a, 0xf52, ..., 0xf6a, 0xf72

Table 1367. KEY1_2,
KEY2_2, ..., KEY5_2,
KEY6_2 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 47:32 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_3, KEY2_3, ..., KEY5_3, KEY6_3 Registers

Offsets: 0xf4b, 0xf53, ..., 0xf6b, 0xf73

Table 1368. KEY1_3,
KEY2_3, ..., KEY5_3,
KEY6_3 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 63:48 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_4, KEY2_4, ..., KEY5_4, KEY6_4 Registers

Offsets: 0xf4c, 0xf54, ..., 0xf6c, 0xf74

Table 1369. KEY1_4,
KEY2_4, ..., KEY5_4,
KEY6_4 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 79:64 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_5, KEY2_5, ..., KEY5_5, KEY6_5 Registers

Offsets: 0xf4d, 0xf55, ..., 0xf6d, 0xf75

Table 1370. KEY1_5,
KEY2_5, ..., KEY5_5,
KEY6_5 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-

Bits	Description	Type	Reset
15:0	Bits 95:80 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_6, KEY2_6, ..., KEY5_6, KEY6_6 Registers

Offsets: 0xf4e, 0xf56, ..., 0xf6e, 0xf76

Table 1371. KEY1_6, KEY2_6, ..., KEY5_6, KEY6_6 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 111:96 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_7, KEY2_7, ..., KEY5_7, KEY6_7 Registers

Offsets: 0xf4f, 0xf57, ..., 0xf6f, 0xf77

Table 1372. KEY1_7, KEY2_7, ..., KEY5_7, KEY6_7 Registers

Bits	Description	Type	Reset
31:16	Reserved.	-	-
15:0	Bits 127:112 of OTP access key n (ECC)	RO	-

OTP_DATA: KEY1_VALID Register

Offset: 0xf79

Description

Valid flag for key 1. Once the valid flag is set, the key can no longer be read or written, and becomes a valid fixed key for protecting OTP pages.

Table 1373. KEY1_VALID Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	VALID_R2	Redundant copy of VALID, with 3-way majority vote	RO	-
15:9	Reserved.	-	-	-
8	VALID_R1	Redundant copy of VALID, with 3-way majority vote	RO	-
7:1	Reserved.	-	-	-
0	VALID		RO	-

OTP_DATA: KEY2_VALID Register

Offset: 0xf7a

Description

Valid flag for key 2. Once the valid flag is set, the key can no longer be read or written, and becomes a valid fixed key for protecting OTP pages.

Table 1374. KEY2_VALID Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	VALID_R2	Redundant copy of VALID, with 3-way majority vote	RO	-
15:9	Reserved.	-	-	-
8	VALID_R1	Redundant copy of VALID, with 3-way majority vote	RO	-
7:1	Reserved.	-	-	-

Bits	Name	Description	Type	Reset
0	VALID		RO	-

OTP_DATA: KEY3_VALID Register

Offset: 0xf7b

Description

Valid flag for key 3. Once the valid flag is set, the key can no longer be read or written, and becomes a valid fixed key for protecting OTP pages.

Table 1375.
KEY3_VALID Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	VALID_R2	Redundant copy of VALID, with 3-way majority vote	RO	-
15:9	Reserved.	-	-	-
8	VALID_R1	Redundant copy of VALID, with 3-way majority vote	RO	-
7:1	Reserved.	-	-	-
0	VALID		RO	-

OTP_DATA: KEY4_VALID Register

Offset: 0xf7c

Description

Valid flag for key 4. Once the valid flag is set, the key can no longer be read or written, and becomes a valid fixed key for protecting OTP pages.

Table 1376.
KEY4_VALID Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	VALID_R2	Redundant copy of VALID, with 3-way majority vote	RO	-
15:9	Reserved.	-	-	-
8	VALID_R1	Redundant copy of VALID, with 3-way majority vote	RO	-
7:1	Reserved.	-	-	-
0	VALID		RO	-

OTP_DATA: KEY5_VALID Register

Offset: 0xf7d

Description

Valid flag for key 5. Once the valid flag is set, the key can no longer be read or written, and becomes a valid fixed key for protecting OTP pages.

Table 1377.
KEY5_VALID Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	VALID_R2	Redundant copy of VALID, with 3-way majority vote	RO	-
15:9	Reserved.	-	-	-
8	VALID_R1	Redundant copy of VALID, with 3-way majority vote	RO	-

Bits	Name	Description	Type	Reset
7:1	Reserved.	-	-	-
0	VALID		RO	-

OTP_DATA: KEY6_VALID Register

Offset: 0xf7e

Description

Valid flag for key 6. Once the valid flag is set, the key can no longer be read or written, and becomes a valid fixed key for protecting OTP pages.

Table 1378.
KEY6_VALID Register

Bits	Name	Description	Type	Reset
31:17	Reserved.	-	-	-
16	VALID_R2	Redundant copy of VALID, with 3-way majority vote	RO	-
15:9	Reserved.	-	-	-
8	VALID_R1	Redundant copy of VALID, with 3-way majority vote	RO	-
7:1	Reserved.	-	-	-
0	VALID		RO	-

OTP_DATA: PAGE0_LOCK0, PAGE1_LOCK0, ..., PAGE61_LOCK0, PAGE62_LOCK0 Registers

Offsets: 0xf80, 0xf82, ..., 0xffa, 0ffc

Description

Lock configuration LSBs for page N (rows $0x40 * N$ through $0x40 * N + 0x3f$). Locks are stored with 3-way majority vote encoding, so that bits can be set independently.

This OTP location is always readable, and is write-protected by its own permissions.

Table 1379.
PAGE0_LOCK0,
PAGE1_LOCK0, ...,
PAGE61_LOCK0,
PAGE62_LOCK0
Registers

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	R2	Redundant copy of bits 7:0	RO	-
15:8	R1	Redundant copy of bits 7:0	RO	-
7	Reserved.	-	-	-
6	NO_KEY_STATE	State when at least one key is registered for this page and no matching key has been entered. 0x0 → read_only 0x1 → inaccessible	RO	-
5:3	KEY_R	Index 1-6 of a hardware key which must be entered to grant read access, or 0 if no such key is required.	RO	-
2:0	KEY_W	Index 1-6 of a hardware key which must be entered to grant write access, or 0 if no such key is required.	RO	-

OTP_DATA: PAGE0_LOCK1, PAGE1_LOCK1, ..., PAGE61_LOCK1, PAGE62_LOCK1 Registers

Offsets: 0xf81, 0xf83, ..., 0ffb, 0ffd

Description

Lock configuration MSBs for page N (rows $0x40 * N$ through $0x40 * N + 0x3f$). Locks are stored with 3-way majority vote encoding, so that bits can be set independently.

This OTP location is always readable, and is write-protected by its own permissions.

Table 1380.
PAGE0_LOCK1,
PAGE1_LOCK1, ...,
PAGE61_LOCK1,
PAGE62_LOCK1
Registers

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	R2	Redundant copy of bits 7:0	RO	-
15:8	R1	Redundant copy of bits 7:0	RO	-
7:6	Reserved.	-	-	-
5:4	LOCK_BL	Dummy lock bits reserved for bootloaders (including the RP2350 USB bootloader) to store their own OTP access permissions. No hardware effect, and no corresponding SW_LOCKx registers. 0x0 → Bootloader permits user reads and writes to this page 0x1 → Bootloader permits user reads of this page 0x2 → Do not use. Behaves the same as INACCESSIBLE 0x3 → Bootloader does not permit user access to this page	RO	-
3:2	LOCK_NS	Lock state for Non-secure accesses to this page. Thermometer-coded, so lock state can be advanced permanently from any state to any less-permissive state by programming OTP. Software can also advance the lock state temporarily (until next OTP reset) using the SW_LOCKx registers. Note that READ_WRITE and READ_ONLY are equivalent in hardware, as the SBPI programming interface is not accessible to Non-secure software. However, Secure software may check these bits to apply write permissions to a Non-secure OTP programming API. 0x0 → Page can be read by Non-secure software, and Secure software may permit Non-secure writes. 0x1 → Page can be read by Non-secure software 0x2 → Do not use. Behaves the same as INACCESSIBLE. 0x3 → Page can not be accessed by Non-secure software.	RO	-
1:0	LOCK_S	Lock state for Secure accesses to this page. Thermometer-coded, so lock state can be advanced permanently from any state to any less-permissive state by programming OTP. Software can also advance the lock state temporarily (until next OTP reset) using the SW_LOCKx registers. 0x0 → Page is fully accessible by Secure software. 0x1 → Page can be read by Secure software, but can not be written. 0x2 → Do not use. Behaves the same as INACCESSIBLE. 0x3 → Page can not be accessed by Secure software.	RO	-

OTP_DATA: PAGE63_LOCK0 Register

Offset: 0xffe

Description

Lock configuration LSBs for page 63 (rows 0xfc0 through 0xffff). Locks are stored with 3-way majority vote encoding, so that bits can be set independently.

This OTP location is always readable, and is write-protected by its own permissions.

Table 1381.
PAGE63_LOCK0
Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	R2	Redundant copy of bits 7:0	RO	-
15:8	R1	Redundant copy of bits 7:0	RO	-
7	RMA	Decommission for RMA of a suspected faulty device. This re-enables the factory test JTAG interface, and makes pages 3 through 61 of the OTP permanently inaccessible.	RO	-
6	NO_KEY_STATE	State when at least one key is registered for this page and no matching key has been entered. 0x0 → read_only 0x1 → inaccessible	RO	-
5:3	KEY_R	Index 1-6 of a hardware key which must be entered to grant read access, or 0 if no such key is required.	RO	-
2:0	KEY_W	Index 1-6 of a hardware key which must be entered to grant write access, or 0 if no such key is required.	RO	-

OTP_DATA: PAGE63_LOCK1 Register

Offset: 0xffff

Description

Lock configuration MSBs for page 63 (rows 0xfc0 through 0xffff). Locks are stored with 3-way majority vote encoding, so that bits can be set independently.

This OTP location is always readable, and is write-protected by its own permissions.

Table 1382.
PAGE63_LOCK1
Register

Bits	Name	Description	Type	Reset
31:24	Reserved.	-	-	-
23:16	R2	Redundant copy of bits 7:0	RO	-
15:8	R1	Redundant copy of bits 7:0	RO	-
7:6	Reserved.	-	-	-
5:4	LOCK_BL	Dummy lock bits reserved for bootloaders (including the RP2350 USB bootloader) to store their own OTP access permissions. No hardware effect, and no corresponding SW_LOCKx registers. 0x0 → Bootloader permits user reads and writes to this page 0x1 → Bootloader permits user reads of this page 0x2 → Do not use. Behaves the same as INACCESSIBLE 0x3 → Bootloader does not permit user access to this page	RO	-

Bits	Name	Description	Type	Reset
3:2	LOCK_NS	<p>Lock state for Non-secure accesses to this page. Thermometer-coded, so lock state can be advanced permanently from any state to any less-permissive state by programming OTP. Software can also advance the lock state temporarily (until next OTP reset) using the SW_LOCKx registers.</p> <p>Note that READ_WRITE and READ_ONLY are equivalent in hardware, as the SBPI programming interface is not accessible to Non-secure software. However, Secure software may check these bits to apply write permissions to a Non-secure OTP programming API.</p> <p>0x0 → Page can be read by Non-secure software, and Secure software may permit Non-secure writes.</p> <p>0x1 → Page can be read by Non-secure software</p> <p>0x2 → Do not use. Behaves the same as INACCESSIBLE.</p> <p>0x3 → Page can not be accessed by Non-secure software.</p>	RO	-
1:0	LOCK_S	<p>Lock state for Secure accesses to this page. Thermometer-coded, so lock state can be advanced permanently from any state to any less-permissive state by programming OTP. Software can also advance the lock state temporarily (until next OTP reset) using the SW_LOCKx registers.</p> <p>0x0 → Page is fully accessible by Secure software.</p> <p>0x1 → Page can be read by Secure software, but can not be written.</p> <p>0x2 → Do not use. Behaves the same as INACCESSIBLE.</p> <p>0x3 → Page can not be accessed by Secure software.</p>	RO	-

Chapter 14. Electrical and Mechanical

Appendix A: Register Field Types

Changes from RP2040

Register field types are unchanged. The descriptions have been clarified.

Standard types

RW:

- Read/Write
- Read operation returns the register value
- Write operation updates the register value

RO:

- Read-only
- Read operation returns the register value
- Write operations are ignored

WO:

- Write-only
- Read operation returns 0
- Write operation updates the register value

Clear types

SC:

- Self-Clearing
- Writing a 1 to a bit in an SC field will trigger an event, once the event is triggered the bit clears automatically
- Writing a 0 to a bit in an SC field does nothing

WC:

- Write-Clear
- Writing a 1 to a bit in a WC field will write that bit to 0

- Writing a 0 to a bit in a WC field does nothing
- Read operation returns the register value

FIFO types

These fields are used for reading and writing data to and from FIFOs. Accompanying registers provide FIFO control and status. There is no fixed format for the control and status registers, as they are specific to each FIFO interface.

RWF:

- Read/Write FIFO
- Reading this field returns data from a FIFO
 - When the read is complete, the data value is removed from the FIFO
 - If the FIFO is empty, a default value will be returned; the default value is specific to each FIFO interface
- Data written to this field is pushed to a FIFO, Behaviour when the FIFO is full is specific to each FIFO interface
- Read and write operations may access different FIFOs

RF:

- Read FIFO
- Functions the same as RWF, but read-only

WF:

- Write FIFO
- Functions the same as RWF, but write-only

Appendix B: Units Used in This Document

This datasheet uses SI units, plus the IEC 60027-2 convention for power-of-two prefixes.

Memory and Storage Capacity

This datasheet uses the following units to measure memory and storage capacity:

- b: bit, the most basic unit of information in computing and digital communication. A logical state with two possible values: 0 and 1.
- B: byte, $8 \text{ b} = 2^3 \text{ b}$
- kb: Kilobit, 1000 b
- Kib: Kibibit, $1024 \text{ b} = 2^{10} \text{ b}$
- kB: Kilobyte, 1000 B
- KiB: Kibibyte, $1024 \text{ B} = 2^{10} \text{ B}$
- Mb: Megabit, $1,000,000 \text{ b}$
- Mib: Mebibit, $1,048,576 \text{ b} = 1024^2 \text{ b}$
- MB: Megabyte, $1,000,000 \text{ B}$
- MiB: Mebibyte, $1,048,576 \text{ B} = 1024^2 \text{ B} = 2^{20} \text{ B}$
- Gb: Gigabit, $1,000,000,000 \text{ b}$
- Gib: Gibibit, $1,073,741,824 \text{ b} = 1024^3 \text{ b} = 2^{30} \text{ b}$
- GB: Gigabyte, $1,000,000,000 \text{ B}$
- GiB: Gibibyte, $1,073,741,824 \text{ B} = 1024^3 \text{ B} = 2^{30} \text{ B}$

Physical Quantities

The following units express physical quantities such as voltage and frequency:

- A: ampere, unit of electrical current, one coulomb per second
 - mA: milliampere, 10^{-3} A
 - μA : microampere, 10^{-6} A
- Ω : ohm, unit of electrical impedance or resistance, one volt per ampere
 - M Ω : megohm, $10^6 \Omega$
 - k Ω : kilohm, $10^3 \Omega$
 - m Ω : milliohm, $10^{-3} \Omega$
- V: volt, unit of electrical potential difference, one joule per coulomb
 - kV: kilovolt, 10^3 V
 - mV: millivolt, 10^{-3} V

- μ V: microvolt, 10^{-6} V
- Hz: hertz, unit of frequency, one period per second (s^{-1})
 - kHz: kilohertz, 10^3 Hz
 - MHz: megahertz, 10^6 Hz
 - GHz: gigahertz, 10^9 Hz
- F: farad, unit of electrical capacitance, one coulomb per volt
 - mF: millifarad, 10^{-3} F
 - μ F: microfarad, 10^{-6} F
 - nF: nanofarad, 10^{-9} F
 - pF: picofarad, 10^{-12} F
- H: henry, unit of electrical inductance, one volt-second per ampere (VsA^{-1})
 - mH: millihenry, 10^{-3} H
 - μ H: microhenry, 10^{-6} H
 - nH: nanohenry, 10^{-9} H
- s: second, unit of time
 - ms: millisecond, 10^{-3} s
 - μ s: microsecond, 10^{-6} s
 - ns: nanosecond, 10^{-9} s
 - ps: picosecond, 10^{-12} s
- J: joule, unit of energy or work, one newton-metre
- C: coulomb, unit of electrical charge
 - mC: millicoulomb, 10^{-3} C
 - μ C: microcoulomb, 10^{-6} C
 - nC: nanocoulomb, 10^{-9} C
- m: metre, unit of length or distance
 - mm: millimetre, 10^{-3} m
- $^{\circ}$ C: degree celsius, unit of temperature
- W: watt, unit of power, one joule per second (Js^{-1}) or one volt-ampere (VA)
 - mW: milliwatt, 10^{-3} W
 - μ W: microwatt, 10^{-6} W
 - nW: nanowatt, 10^{-9} W

Scale Prefixes

The units described in previous sections use the following conventions for unit scale prefixes:

- G: giga, factor of 10^9 (one short billion)
- Gi: gibi, factor of 2^{30} (approximately one short billion, exactly a power of two)
- M: mega, factor of 10^6 (one million)

- Mi: mebi, factor of 2^{20} (approximately one million, exactly a power of two)
- k: kilo, factor of 10^3 (one thousand)
- Ki: kibi, factor of 2^{10} (approximately one thousand, exactly a power of two)
- c: centi, factor of 10^{-2} (one hundredth)
- m: milli, factor of 10^{-3} (one thousandth)
- μ : micro, factor of 10^{-6} (one millionth)
- n: nano, factor of 10^{-9} (one short billionth)
- p: pico, factor of 10^{-12} (one short trillionth)

Appendix E: Errata

Alphabetical by section.

ACCESSCTRL

RP2350-E3

Reference	RP2350-E3
Summary	In QFN-60 package, GPIO_NSmask controls wrong PADS registers
Affects	RP2350 A2, QFN-60 package only
Description	<p>RP2350 remaps IOs, their control registers and their ADC channels so that both package sizes appear to have consecutively numbered GPIOs, even though for physical design reasons the QFN-60 package bonds out a sparse selection of IO pads.</p> <p>The connection between the GPIO_NSmask0/GPIO_NSmask1 registers and the PADS registers does not take this remapping into account. Consequently, in the QFN-60 package only, the GPIO_NSmask0 register bits are applied to registers for the wrong pads. Specifically, PADS_BANK0 registers 29 through 0 are controlled by the concatenation of GPIO_NSmask bits 47 through 44, 39 through 33, 30 through 28, 24 through 17 and 15 through 8 (all inclusive ranges).</p> <p>This means that granting Non-secure access to the PADS registers in the QFN-60 package does not allow Non-secure software to control the correct pads. It may also allow Non-secure control of pads that are not granted in GPIO_NSmask0.</p> <p>Note that the QSPI PADS registers (Bank 1) are not affected, as these are not remapped for different packages.</p>
Workaround	<p>Disable Non-secure access to the PADS registers by clearing PADS_BANK0.NSP, NSU.</p> <p>Implement a Secure Gateway (Arm) or ecall handler (RISC-V) to permit Non-secure/U-mode code to read/write its assigned PADS_BANK0 registers.</p>
Fixed by	Documentation, software

Bootrom

RP2350-E10

Reference	RP2350-E10
Summary	UF2 drag & drop doesn't work with partition tables
Affects	RP2350 A2

Description	When dragging & dropping a UF2 onto the USB Mass Storage Device, the bootrom on chip revision A2 does not set up the flash before checking the partition table. This causes the UF2 download to fail if there is a partition table present.
Workaround	<p>Add a single block at the start of the UF2 with an Absolute family ID, targeting the end of Flash, with block number set to 0 and number of blocks set to 2. This block will be written to flash first but doesn't reboot the device, and sets up the flash for the rest of the UF2 to be downloaded correctly.</p> <p>This is handled for you automatically by picotool in the SDK, which adds this block when generating UF2s if the <code>--abs</code> flag is specified.</p>
Fixed by	Documentation, software

DMA

RP2350-E5

Reference	RP2350-E5
Summary	Interactions between <code>CHAIN_TO</code> and <code>ABORT</code> of active channels
Affects	RP2350 A2
Description	<p>The <code>CHAN_ABORT</code> register commands a DMA channel to stop issuing transfers, and to clear its <code>BUSY</code> flag once in-flight transfers have completed. This was originally intended for recovering channels which are stuck with their <code>DREQ</code> low. An <code>ABORT</code> is initiated by writing a bitmap of aborted channels to <code>CHAN_ABORT</code>. Bits remain set until each channel comes to rest.</p> <p>This erratum is a compound of two behaviours: first, aborting a channel will cause its <code>CHAIN_TO</code> to fire, if and only if the aborted channel is the last channel to have completed a write transfer. Second, a channel undergoing an <code>ABORT</code> is susceptible to be re-triggered on the last cycle before the <code>ABORT</code> register clears, because the channel is both inactive and enabled on this cycle, and the <code>ABORT</code> itself does not inhibit triggering. However, since the <code>ABORT</code> is still in effect, the transfer count is held at zero. On the cycle after the <code>ABORT</code> finishes, the channel completes because its transfer counter is zero. This causes the channel's <code>IRQ</code> and <code>CHAIN_TO</code> to fire on the cycle after the <code>ABORT</code> completes.</p> <p>These two behaviours are problematic when aborting multiple channels that chain to one another, since they may cause the channels to immediately restart post-abort.</p>
Workaround	Before aborting an active channel, clear the <code>EN</code> bit (<code>CH0_CTRL_TRIG.EN</code>) of both the aborted channel and any channel it chains to. This ensures the channel is not susceptible to re-triggering.
Fixed by	Documentation

RP2350-E8

Reference	RP2350-E8
Summary	<code>CHAIN_TO</code> may not fire for zero-length transfers
Affects	RP2350 A2

Description	<p>The <code>CTRL.CHAIN_TO</code> field configures a channel to start another channel once it completes its programmed sequence of transfers. The <code>CHAIN_TO</code> takes place on the cycle where the channel's last write completes, and the chainee becomes active on the next cycle.</p> <p>The hardware implementation assumes that <code>CHAIN_TO</code> always happens as a result of a write completion. This is not the case when a channel is triggered with a transfer count of zero: in this case the channel completes on the cycle immediately after the trigger, without performing any bus accesses.</p> <p>A <code>CHAIN_TO</code> from a channel started with a transfer count of zero will fire if and only if that channel is the last channel to have completed a write transfer. This is true only when the channel in question has previously performed a non-zero-length sequence of transfers, and no other channel has completed a write since.</p>
Workaround	Do not use <code>CHAIN_TO</code> in conjunction with zero-length transfers. Avoid zero-length transfers in the middle of control block lists, and replace them with dummy transfers if possible.
Fixed by	Documentation

GPIO

RP2350-E9

Reference	RP2350-E9
Summary	Latching behaviour on Bank 0 GPIO pull-down resistors
Affects	RP2350 A2
Description	<p>This issue presents under the following conditions, for any GPIO 0 through 47:</p> <ol style="list-style-type: none"> 1. Pull-down resistor is enabled in <code>GPIO0.PDE</code> 2. Pull-up resistor is disabled in <code>GPIO0.PUE</code> 3. Input buffer is enabled in <code>GPIO0.IE</code> 4. Isolation is clear in <code>GPIO0.ISO</code>, or the previous were true at the point isolation was set 5. Input is driven externally to a logic-1 state <p>When all of the above conditions are met, the pad enables the pull-up resistor in addition to the requested pull-down resistor. This is caused by faulty bus-keeper logic within the pad macro itself. Due to asymmetries in the pull-up and pull-down circuitry, this pulls the pad to a voltage somewhat above mid-rail (around 2.1 to 2.2 V when IOVDD is 3.3 V), which latches the pad in a logic-1 state.</p> <p>Note that this does not affect the pull-down behaviour of the pads immediately following a PoR or RUN reset, because the input enable field is initially clear. The pull-down resistors function normally in this state.</p> <p>This issue does not affect the QSPI pads, which use a different pad macro without the faulty bus keeper logic. The USB PHY's pull-down resistors are also unaffected.</p> <p>This issue does also affect the SWD pads, which use the same fault-tolerant pad macro as the Bank 0 GPIOs. However, both SWD pads are pull-up by default, so there is no ill effect.</p>

Workaround	<p>When pull-down behaviour is required, clear the pad input enable in <code>GPIO0.IE</code> (for GPIOs 0 through 47) to ensure that only the pull-down resistor is enabled.</p> <p>To read the state of a pulled-down GPIO from software, enable the input buffer by setting <code>GPIO0.IE</code> immediately before reading, and then re-disable immediately afterwards. Note that if the pad is already a logic-0, re-enabling the input does not disturb the pull-down state.</p>
Fixed by	Documentation

Hazard3

RP2350-E4

Reference	RP2350-E4
Summary	System Bus Access stalls indefinitely when core 1 is in clock-gated sleep
Affects	RP2350 A2
Description	<p>System Bus Access (SBA) is a RISC-V debug feature that allows the Debug Module direct access to the system bus, independent of the state of harts in the system. RP2350 implements SBA by arbitrating Debug Module bus accesses with the core 1 load/store port.</p> <p>Hazard3 implements custom low-power states controlled by the <code>MSLEEP</code> CSR. When <code>MSLEEP.DEEPSLEEP</code> is set, Hazard3 completely gates its clock, with the exception of the minimal logic required to wake again. Due to a design oversight, this also clock-gates the arbiter between SBA and load/store bus access. (This is addressed in upstream commit c11581e.)</p> <p>Consequently, if you initiate an SBA transfer whilst <code>MSLEEP.DEEPSLEEP</code> is set on core 1, and core 1 is in a WFI-equivalent sleep state, the SBA transfer will make no progress until core 1 wakes from the WFI state. The processor wakes upon an enabled interrupt being asserted, or a debug halt request.</p>
Workaround	<p>Either configure your debug translator to not use SBA, or do not enter clock-gated sleep on core 1. The A2 bootrom mitigates this issue by not setting DEEPSLEEP in the initial core 1 wait-for-launch code.</p> <p>Note that the processors are synthesised with hierarchical clock gating, so the top-level clock gate controlled by the DEEPSLEEP flag brings minimal power savings over a default WFI sleep state.</p>
Fixed by	Documentation

RP2350-E6

Reference	RP2350-E6
Summary	PMPCFGx RWX fields are transposed
Affects	RP2350 A2

Description	The Physical Memory Protection unit (PMP) defines read, write and execute permissions (RWX) for configurable ranges of physical memory. The RWX permissions for four regions are packed into each 32-bit PMPCFG register, PMPCFG0 through PMPCFG3 . Per the RISC-V privileged ISA specification, the permission fields are ordered X, W, R from MSB to LSB. Hazard3 implements them in the order R, W, X. This means software using the correct bit order will have its read permissions applied as execute, and vice versa. (See upstream commit 7d37029 .)
Workaround	When configuring PMP with X != R, use the bit order implemented by this version of Hazard3. In the SDK, the hardware/regs/rvcsr.h register header provides bitfield definitions for the as-implemented order when building for RP2350.
Fixed by	Documentation

RP2350-E7

Reference	RP2350-E7
Summary	U-mode does not ignore <code>mstatus.mie</code>
Affects	RP2350 A2
Description	<p>The <code>MSTATUS.MIE</code> bit is a global enable for interrupts which target M-mode. Software generally clears this momentarily to ensure short critical sections are atomic with respect to interrupt handlers.</p> <p>The RISC-V privileged ISA specification requires that the interrupt enable flag for a given privilege mode is treated as 1 when the hart is in a lower privilege mode. In this case, <code>mstatus.mie</code> should be treated as 1 when the core is in U-mode.</p> <p>Hazard3 does not implement this rule, so entering U-mode with M-mode interrupts disabled results in no M-mode interrupts being taken. (See upstream commit a84742a.)</p>
Workaround	When returning to U-mode from M-mode via an <code>mret</code> with <code>mstatus.mpp == 0</code> , ensure <code>mstatus.mpie</code> is set, so that IRQs will be enabled by the return.
Fixed by	Documentation

SIO

RP2350-E1

Reference	RP2350-E1
Affects	RP2350 A2
Summary	Interpolator <code>OVERF</code> bits are broken by new right-rotate behaviour

Description	<p>RP2350 replaces the interpolator right-shift with a right-rotate, so that left shifts can be synthesised. This is useful for scaled indexed addressing in tight address-generating loops.</p>
	<p>The <code>OVERF</code> flag functions by checking for nonzero bits in the post-shift value that have been masked out by the <code>MSB</code> mask configured by the <code>CTRL_LANE0_MASK_MSB</code> and <code>CTRL_LANE1_MASK_MSB</code> register fields. This is used to discard samples outside of the $[0, 1]$ wrapping domain of UV coordinates represented by <code>ACCUM0</code> and <code>ACCUM1</code>, for example in affine-transformed sprite sampling.</p>
	<p>The issue occurs because the right-rotate causes nonzero LSBs to be rotated up to the MSBs. These nonzero bits spuriously set the <code>OVERF</code> flag.</p>
Workaround	<p>Either compute <code>OVERF</code> manually by checking the <code>ACCUM0/ACCUM1</code> MSBs, or precompute the bounds in advance to avoid per-sample checks.</p>
Fixed by	<p>Documentation</p>

RP2350-E2

Reference	RP2350-E2
Summary	SIO SPINLOCK writes are mirrored at <code>+0x80</code> offset
Affects	RP2350 A2
Description	<p>The SIO contains spinlock registers, <code>SPINLOCK0</code> through <code>SPINLOCK31</code>. Reading a spinlock register attempts to claim it, returning nonzero if the claim was successful and 0 if unsuccessful. Writing to a spinlock register releases it, so the next claim will be successful. SIO spinlock registers are at register offsets <code>0x100</code> through <code>0x17c</code> within SIO.</p> <p>RP2350 adds new SIO registers at register offsets <code>0x180</code> and above: Doorbells, the PERI_NONSEC register, the RISC-V soft IRQ register, the RISC-V MTIME registers, and the TMDS encoder.</p> <p>The SIO address decoder detects writes to spinlocks by decoding on bit 8 of the address. This means writes in the range <code>0x180</code> through <code>0x1fc</code> are spuriously detected as writes to the corresponding spinlock address 128 bytes below, in the range <code>0x100</code> through <code>0x17c</code>. Writing to any of these high registers will set the corresponding lock to the unclaimed state.</p> <p>This erratum only affects writes to the spinlock registers. Reads are correctly decoded, so are not affected by accesses above <code>0x17c</code>.</p>
Workaround	<p>Use processor atomic instructions instead of the SIO spinlocks. The SDK <code>hardware_sync_spin_lock</code> library uses software lock variables by default when building for RP2350, instead of hardware spinlocks.</p> <p>The following SIO spinlocks can be used normally as they do not alias with writable registers: 5, 6, 7, 10, 11, and 18 through 31. Some of the other lock addresses may be used safely depending on which of the high-addressed SIO registers are in use.</p> <p>Note that locks 18 through 24 alias with some read-only TMDS encoder registers, which is safe as only writes are misdecoded.</p>
Fixed by	Documentation, software

Appendix F: Documentation Release History



Raspberry Pi is a trademark of Raspberry Pi Ltd