Chipyard Peripheral User Manual

V0.1 2022-12-12

RISC-V-based 32/64-bit SoCs

Introduction

This user manual targets Chipyard SoC designers and embedded system programmers. It provides information on how to use the Chipyard SoC memory and peripherals. The Chipyard SoC is a highly customizable series of SoCs with different core architecture, memory sizes, and peripherals.

For information on the RISC-V core, refer to the The RISC-V Instruction Set Manual.

Related documents

The RISC-V Instruction Set Manual Volume I: Unprivileged ISA, available from:
https://github.com/riscv/riscv-isa-manual/releases/download/draft-20221206-b7080e0/riscv-spec.pdf

The RISC-V Instruction Set Manual Volume II: Privileged Architecture, available from:
https://github.com/riscv/riscv-isa-manual/releases/download/draft-20221206-b7080e0/riscv-privileged.pdf

Chipyard Documentation, available from: https://chipyard.readthedocs.io/en/stable

Chipyard SoC SDK User Manual, available from:
https://docs.google.com/document/d/17grQ84aCPfWsdhXTRVun_Yj1vKWPbR697m29066y8ro/edit?usp=sharing

Plan (Preface?):

I plan to document the peripherals in the `testchipip` folder. I can expand on the work I've done for BearlyML spec sheet, and write more on the procedures on how to use the functionality of each peripheral, similar to the format of ST user manuals (the image below). I'm not sure if there's already available driver codes for these peripherals, but if not, I can provide a set of bare-metal library, referencing ST's HAL library format. I'll try to make the software toolchain more accessible, supporting all three operating systems (Win, Mac, Linux).

This won't take too long (hopefully). And after that I wish to also try to make the chipyard flow more accessible and easy to set up on a new machine, as currently we are running into compatibility issues etc. when configuring chipyard environment in the bringup class.

Environment on GCP, AWS, Azure etc.

— Yufeng

Reference:

https://www.st.com/resource/en/reference_manual/rm0440-stm32g4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

https://dl.sipeed.com/LONGAN/Nano/DOC/GD32VF103_User_Manual_EN_V1.2.pdf

https://cdn.sparkfun.com/assets/7/f/0/2/7/fe310-g002-manual-v19p05.pdf

Contents

1 Core-Local Interruptor (CLINT)

1.1 Introduction

The Core-Local Interruptor mode has one 64-bit timer register (CLINT_MTIME) and one or many sets of 32-bit machine-mode software interrupt-pending register (CLINT_MSIP) and 64-bit machine-mode timer compare register (CLINT_MTIMECMP). The number of machine-mode software interrupt-pending registers and machine-mode timer compare register equals to the number of harts in the system.

1.2 CLINT main features

- 64-bit timer providing wall-clock time
- Configurable software local core interrupt generation to any hart
- Configurable software local time interrupt generation to any hart

1.3 CLINT functional description

1.3.1 MSIPx Functionality

MSIP register can be used to generate machine-mode software interrupts. Each MSIP register is a 32-bit wide register where the upper 31 bits are read-only and tied to 0. The least significant bit is reflected in the MSIP bit of the MIP CSR. On reset, each MSIP register is cleared to zero.

Software interrupts are most useful for interprocessor communication in multi-hart systems, as harts may write each other's MSIP bits to effect interprocessor interrupts.

1.3.2 MTIME Functionality

MTIME is a 64-bit read-write register that contains the number of cycles counted from the RTC_CLK input.

A timer interrupt is pending whenever MTIME is greater than or equal to the value in the MTIMECMP register.

The timer interrupt is reflected in the MTIP bit of the MIP CSR register.

On reset, MTIME is cleared to zero. The MTIMECMP registers are not reset.

1.3.3 MTIMECMPx Configuration

Setting MTIMECMP value in RV32 systems.

```
# New comparand is in a1:a0.

li t0, -1

la t1, mtimecmp

sw t0, 0(t1) # No smaller than old value.
```

sw a1, 4(t1) # No smaller than new value.

sw a0, 0(t1) # New value.

1.4 CLINT registers

This section gives a detailed description of the CLINT registers.

The peripheral registers can be written in word, half-word, or byte mode.

1.4.1 Machine-mode software interrupt pending register n (MSIPn)

Address offset: 0x00 + 0x04 * n

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															MSIP
															rw

0	**MSIP**	Machine-mode software interrupt pending for hart0.
		This bit is controlled by software.
		0: No software interrupt pending
		1: Software interrupt pending

1.4.2 Machine-mode time compare register n (MTIMECMPn)

Address offset: 0x4000 + 0x04 * n

Reset value: <previous value>

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
MTIMECMP[63:48]															
rw															

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
MTIMECMP[47:32]															
rw															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MTIMECMP[31:16]															
rw															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MTIMECMP[15:0]															
rw															

63:0	**MTIMECMP**	Machine-mode time compare target

These bits are controlled by software. Hardware will generate a machine-mode timer interrupt to hart 0 when MTIME0 is greater than or equal to MTIMECMP0.

1.4.3 Machine-mode time register (MTIME)

Address offset: 0xBFF8

Reset value: 0x0000_0000

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
MTIME[63:48]															
rw															
47	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**	**32**
MTIME[47:32]															
rw															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
MTIME[31:16]															
rw															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MTIME[15:0]															
rw															

63:0 **MTIME** Machine-mode time

These bits are controlled by software and hardware.

2 Platform-Level Interrupt Controller (PLIC)

3 General Purpose I/Os (GPIO)

3.1 Introduction

Each general-purpose I/O port has five 32-bit configuration registers (GPIOx_INPUT_EN, GPIOx_OUTPUT_EN, GPIOx_PUE GPIOx_DS, and GPIOx_XOR), two 32-bit data registers (GPIOx_INPUT_VAL and GPIOx_OUTPUT_VAL) and eight 32-bit interrupt control/status register (GPIOx_RISE_IE, GPIOx_RISE_IP, GPIOx_FALL_IE, GPIOx_FALL_IP, GPIOx_HIGH_IE, GPIOx_HIGH_IP, GPIOx_LOW_IE, GPIOx_LOW_IP). In addition, all GPIOs have two 32-bit alternate function selection registers (GPIOx_IOF_EN and GPIOx_IOF_SEL).

3.2 GPIO main features

- Output states: push-pull + pull-up/down
- Output data from output value register (GPIOx_OUTPUT_VAL) or peripheral (alternate function output)
- Drive strength selection for each I/O
- Input states: floating or pull-up
- Input data to input value register (GPIOx_INPUT_VAL) or peripheral (alternate function input)
- Alternate function selection registers
- Bit invert register (GPIOx_XOR) for fast output inversion

3.3 GPIO functional description

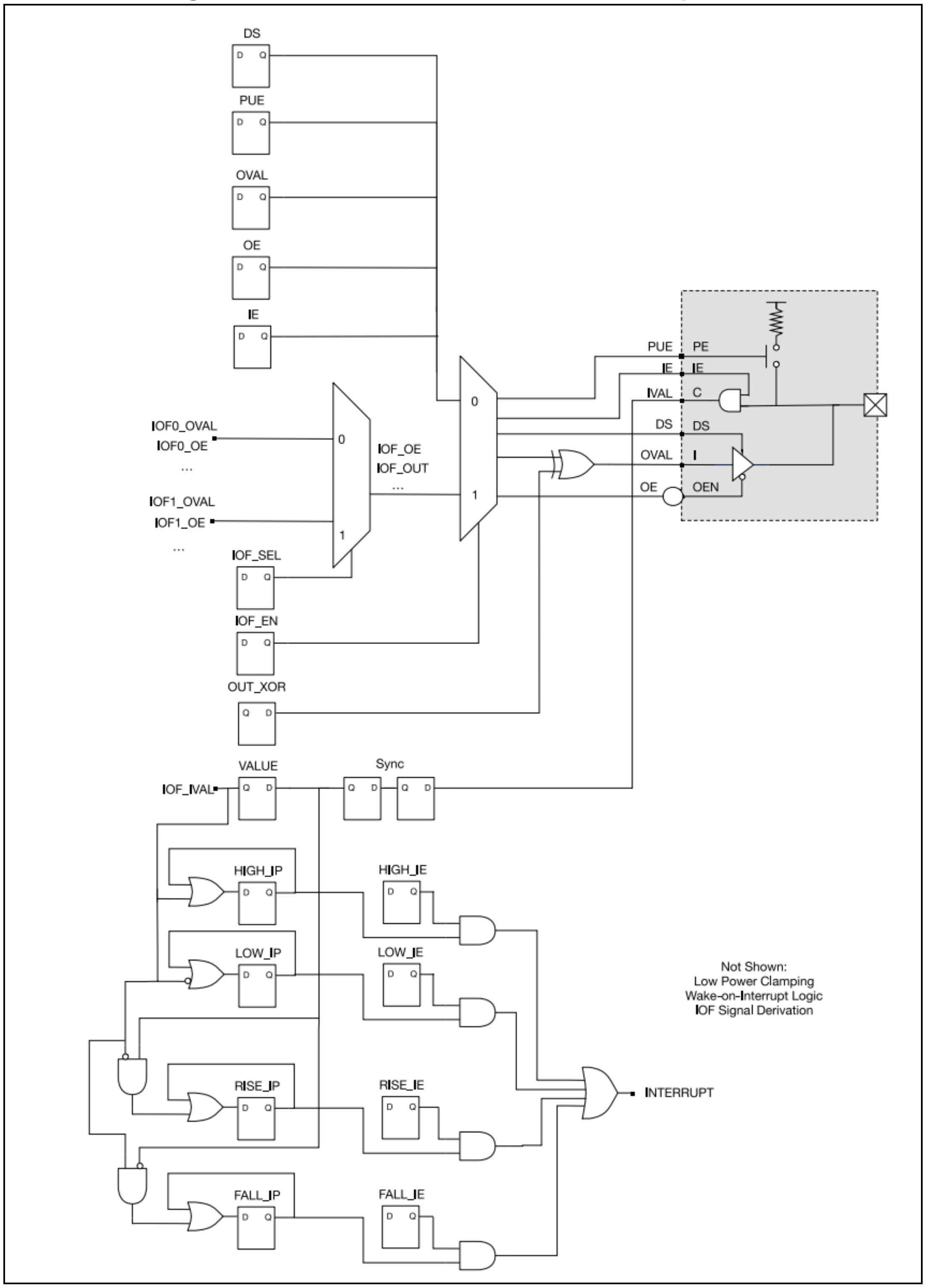
Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

- Disabled
- Input floating
- Input pull-up
- Output push-pull with pull-up capability
- Alternate function input with pull-up capability
- Alternate function output with pull-up capability

Each I/O port bit is freely programmable. However, the I/O port registers have to be accessed as 32-bit words, half-words, or bytes. Processors with RISC-V 'A' extension are capable of atomic operation such as read/modify access to any of the GPIO registers.

Figure 1 shows the basic structures of a standard GPIO I/O port bit.

Figure 1. Basic structure of a standard GPIO I/O port bit



3.3.1 General-purpose I/O (GPIO)

During and after reset, the alternate functions are not active and the I/O ports are disabled. The GPIOx_INPUT_EN, GPIOx_OUTPUT_EN, and GPIOx_PUE are asynchronously reset to 0. The other registers are synchronously reset to 0.

When the pin is configured as output, the value written to the output value register (GPIOx_OUTPUT_VAL) is output on the I/O pin.

The input value register (GPIOx_INPUT_VAL) captures the data present on the I/O pin at every memory clock cycle.

All GPIO pins have weak internal pull-up resistors, which can be activated or not depending on the value in the GPIOx_PUE register.

3.3.1 I/O pin alternate function multiplexer and mapping

The device I/O pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral alternate function (AF) connected to an I/O pin at a time. In this way, there can be no conflict between peripherals available on the same I/O pin.

Each I/O pin has a multiplexer with up to two alternate function inputs that can be configured through the GPIOx_IOF_SEL registers.

- After reset, the multiplexer selection is alternate function 0. The I/Os are configured in alternate function mode through GPIOx_IOF_EN register.
- The specific alternate function assignments is dependent on the chip setting, and may differ across different chips.
- When a pin is set to alternate function mode, it is possible that the GPIO registers GPIOx_OUTPUT_EN, GPIOx_PUE, GPIOx_DS, GPIOx_INPUT_EN may not be used to control the pin directly. Rather, the pins may be controlled by hardware driving the IOF. Which functionalities are controlled by the IOF and which are controlled by the software registers are fixed in the hardware on a per-IOF basis. Those that are not controlled by the hardware continue to be controlled by the software registers.

3.3.3 I/O port control registers

Each of the GPIO ports has five 32-bit configuration registers (GPIOx_INPUT_EN, GPIOx_OUTPUT_EN, GPIOx_PUE GPIOx_DS, and GPIOx_XOR) to configure up to 32 I/Os. The GPIOx_INPUT_EN register is used to enable the input mode. The GPIOx_OUTPUT_EN register is used to enable the output mode. The input and output can be enabled simultaneously to monitor the actual output status in cases like bus arbitration applications. The GPIOx_PUE register is used to enable the pull-up resistor whatever the I/O direction. The GPIOx_DS register is used to select the driving strength. The pin driving strength is device-dependent, which can be found in device datasheet. The GPIOx_XOR register is to configure the polarity of the output pin.

3.3.4 I/O port data registers

Each of the GPIO ports has two 32-bit data registers (GPIOx_INPUT_VAL, GPIOx_OUTPUT_VAL). GPIOx_OUTPUT_VAL stores the data to be output. It is read/write accessible. The data input through the I/O pin are stored into the read-only register GPIOx_INPUT_VAL.

There is no need for the software to disable interrupts when programming the GPIOx_OUTPUT_VAL at bit level. It is possible to modify one or more bits in a single atomic TileLink write access.

See Section 3.x.x: GPIO Input Value Register and Section 3.x.x: GPIO Output Value Register for the register descriptions.

3.3.5 I/O alternate function input/output

IOF functionalities have not been implemented in current research chips.

3.3.6 External interrupt/wakeup lines

All ports have external interrupt capability. To use external interrupt lines, the port must be configured in input mode.

Refer to Section x: Peripheral Platform Interrupt Controller (PLIC) and to Section x.x.x: Wakeup event management.

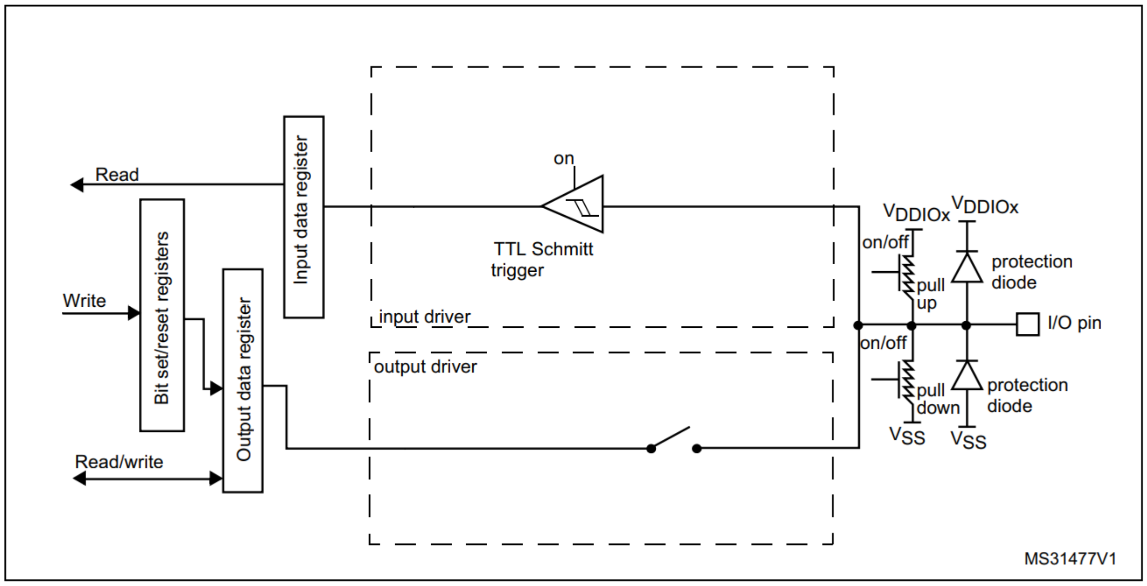
3.3.7 Input configuration

When the I/O port is programmed as input:

- The Schmitt trigger input is activated
- The pull-up resistor is activated depending on the value in the GPIOx_PUE register
- The data present on the I/O pin are sampled into the input data register every TileLink clock cycle
- A read access to the input value register provides the I/O state

Figure 2 shows the input configuration of the I/O port bit

Figure 2. Input floating / pull-up configurations



3.3.8 Output configuration

When the I/O port is programmed as output:

- The Schmitt trigger input is activated

- The pull-up resistor is activated depending on the value in the GPIOx_PUE register
- The data present on the I/O pin are sampled into the input data register every TileLink clock cycle
- A read access to the input value register provides the I/O state if input mode is enabled
- A read access to the output value register gets the last written value

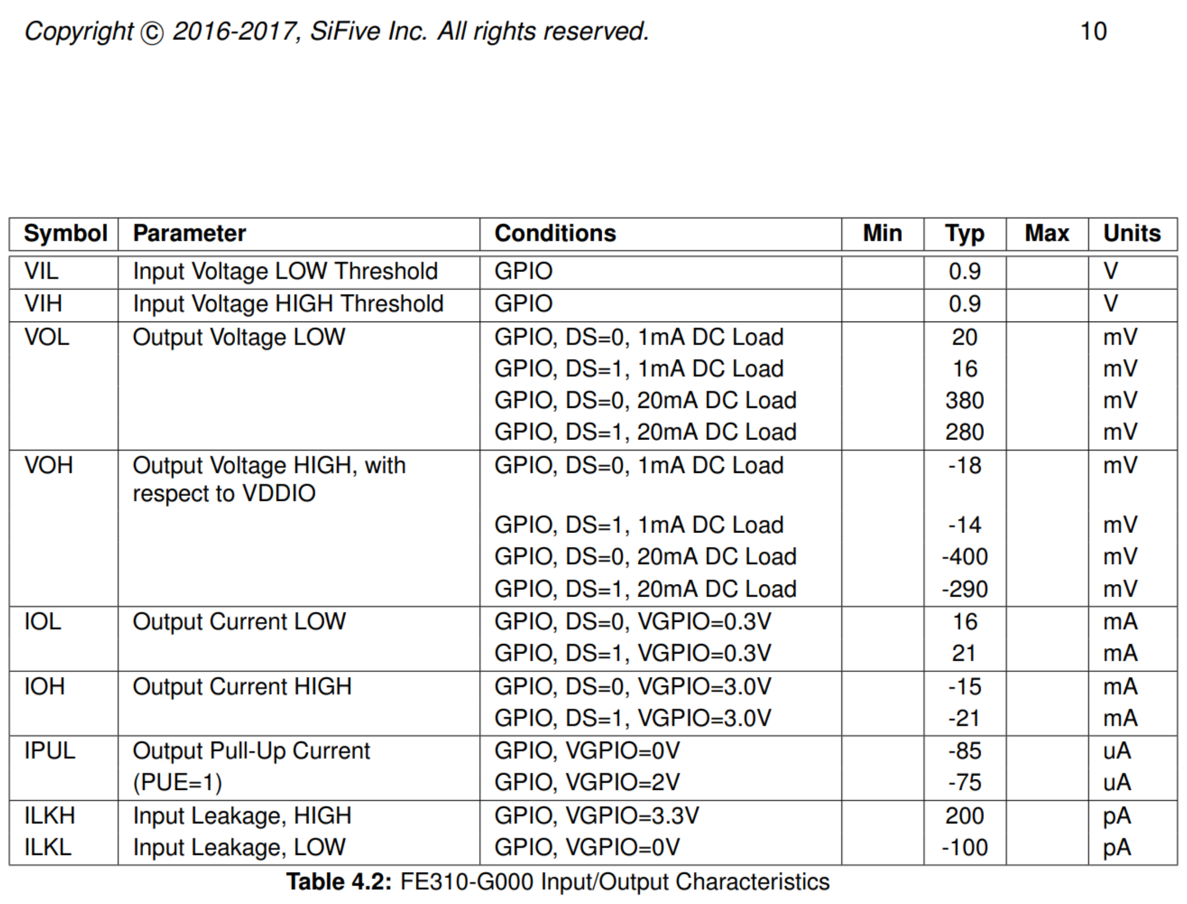
Figure 3 shows the input configuration of the I/O port bit

Figure 3. Output configuration

TODO

3.3.9 Alternate function configuration

TODO



Symbol	Parameter	Conditions	Min	Typ	Max	Units
VIL	Input Voltage LOW Threshold	GPIO		0.9		V
VIH	Input Voltage HIGH Threshold	GPIO		0.9		V
VOL	Output Voltage LOW	GPIO, DS=0, 1mA DC Load		20		mV
		GPIO, DS=1, 1mA DC Load		16		mV
		GPIO, DS=0, 20mA DC Load		380		mV
		GPIO, DS=1, 20mA DC Load		280		mV
VOH	Output Voltage HIGH, with respect to VDDIO	GPIO, DS=0, 1mA DC Load		-18		mV
		GPIO, DS=1, 1mA DC Load		-14		mV
		GPIO, DS=0, 20mA DC Load		-400		mV
		GPIO, DS=1, 20mA DC Load		-290		mV
IOL	Output Current LOW	GPIO, DS=0, VGPIO=0.3V		16		mA
		GPIO, DS=1, VGPIO=0.3V		21		mA
IOH	Output Current HIGH	GPIO, DS=0, VGPIO=3.0V		-15		mA
		GPIO, DS=1, VGPIO=3.0V		-21		mA
IPUL	Output Pull-Up Current (PUE=1)	GPIO, VGPIO=0V		-85		uA
		GPIO, VGPIO=2V		-75		uA
ILKH	Input Leakage, HIGH	GPIO, VGPIO=3.3V		200		pA
ILKL	Input Leakage, LOW	GPIO, VGPIO=0V		-100		pA

Table 4.2: FE310-G000 Input/Output Characteristics

3.4 GPIO registers

This section gives a detailed description of the GPIO registers.

The peripheral registers can be written in word, half-word or byte mode.

3.4.1 Input value register (INPUT_VAL)

Address offset: 0x00

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VAL31	VAL30	VAL29	VAL28	VAL27	VAL26	VAL25	VAL24	VAL23	VAL22	VAL21	VAL20	VAL19	VAL18	VAL17	VAL16
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL15	VAL14	VAL13	VAL12	VAL11	VAL10	VAL9	VAL8	VAL7	VAL6	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

31:0 **VALy** Port value bit y(y=0..31)

These bits are read-only. They contain the input value of the corresponding I/O port.

0: Input signal low
1: Input signal high

3.4.2 Input enable register (INPUT_EN)

Address offset: 0x04

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EN31	EN30	EN29	EN28	EN27	EN26	EN25	EN24	EN23	EN22	EN21	EN20	EN19	EN18	EN17	EN16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN15	EN14	EN13	EN12	EN11	EN10	EN9	EN8	EN7	EN6	EN5	EN4	EN3	EN2	EN1	EN0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:0 **ENy** Port enable bit y(y=0..31)

These bits are read-and-write. They contain the input enable status of the corresponding I/O port.

0: Input disabled
1: Input enabled

3.4.3 Output value register (OUTPUT_VAL)

Address offset: 0x08

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VAL31	VAL30	VAL29	VAL28	VAL27	VAL26	VAL25	VAL24	VAL23	VAL22	VAL21	VAL20	VAL19	VAL18	VAL17	VAL16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL15	VAL14	VAL13	VAL12	VAL11	VAL10	VAL9	VAL8	VAL7	VAL6	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:0	**VALy**	Port value bit y(y=0..31) These bits are read-and-write. They contain the output value of the corresponding I/O port. 0: Output signal low 1: Output signal high

3.4.4 Output enable register (OUTPUT_EN)

Address offset: 0x0C

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EN31	EN30	EN29	EN28	EN27	EN26	EN25	EN24	EN23	EN22	EN21	EN20	EN19	EN18	EN17	EN16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN15	EN14	EN13	EN12	EN11	EN10	EN9	EN8	EN7	EN6	EN5	EN4	EN3	EN2	EN1	EN0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:0	**ENy**	Port enable bit y(y=0..31) These bits are read-and-write. They contain the output enable status of the corresponding I/O port. 0: Input disabled 1: Input enabled

3.4.5 Pullup Enable register (OUTPUT_PUE)

Address offset: 0x10

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EN31	EN30	EN29	EN28	EN27	EN26	EN25	EN24	EN23	EN22	EN21	EN20	EN19	EN18	EN17	EN16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN15	EN14	EN13	EN12	EN11	EN10	EN9	EN8	EN7	EN6	EN5	EN4	EN3	EN2	EN1	EN0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31:0	**ENy**	Port internal pull-up resistor enable bit y(y=0..31) These bits are read-and-write. They contain the pull-up resistor enable status of the corresponding I/O port.

0: Pull-up resistor disabled
1: Pull-up resistor enabled

4 Universal Asynchronous Receiver/Transmitter (UART)

4.1 Introduction

The UART offers a flexible means to perform Full-duplex data exchange with external devices. A very wide range of baud rates can be achieved through a fractional baud rate generator. The UART peripheral does not support hardware flow control or other modem control signals, or synchronous serial data transfers.

4.2 UART main features

- Full-duplex asynchronous communication
- Baud rate generator systems
- 16× Rx oversampling with 2/3 majority voting per bit
- Two internal FIFOs for transmit and receive data with programmable watermark interrupts
- A common programmable transmit and receive baud rate
- Configurable stop bits (1 or 2 stop bits)
- Separate enable bits for transmitter and receiver
- Interrupt sources with flags

4.3 UART functional description

Figure 4. UART block diagram

4.3.1 UART signals

UART bidirectional communications require two pins: Receive Data In (RX) and Transmit Data Out (TX):

- **RX** (Receive Data Input)

  RX is the serial data input. Oversampling techniques are used for data recovery. They discriminate between valid incoming data and noise.

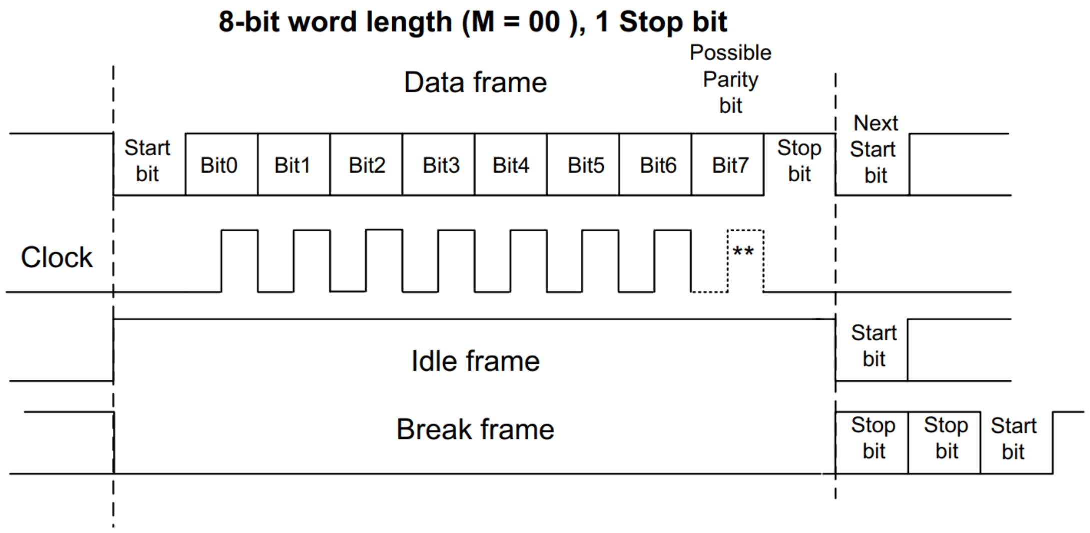
- **TX** (Transmit Data Output)

  When the transmitter is enabled and no data needs to be transmitted, the TX pin is High.

The signal (TX or RX) is in low state during the start bit. It is in high state during the stop bit.

Transmission and reception are driven by a common baud rate generator.

Figure 5. Stop bit programming



4.3.2 UART FIFOs and watermark

The USART comes with a Transmit FIFO (TXFIFO) and a Receive FIFO (RXFIFO).

It is possible to configure the TXFIFO and RXFIFO levels at which the Tx and RX interrupts are triggered. These thresholds are programmed through TXCNT in TXCTRL control register and RXCNT bitfields in RXCTRL control register.

In this case:

- When the number of received data in the RXFIFO is strictly greater than the count specified by the RXCNT bits fields, and the RXWM is set in the IE register, the corresponding interrupt is generated. The interrupt bit is cleared when sufficient entries have been dequeued to fall below the watermark.

  This means that the RXFIFO is filled until the number of data in the RXFIFO is equal to the programmed threshold.

  As an example, when the RXCNT is programmed to ‘111’, the RXFT flag is set when a number of data corresponding to the FIFO size has been received.

- When the number of pending data in the TXFIFO is strictly less than the count specified by the TXCNT bits fields, and the TXWM is set in the IE register, the corresponding interrupt is generated. The interrupt bit is cleared when sufficient entries have been enqueued to exceed the watermark

  This means that the TXFIFO is emptied until the number of empty locations in the TXFIFO is equal to the programmed threshold.

4.3.3 UART transmitter

The Transmit Enable bit (TXEN) in the TXCTRL control register must be set in order to activate the transmitter function. The data in the transmit shift register is output on the TX pin.

Character transmission

During an UART transmission, data shifts out the least significant bit first on the TX pin.

The data written to the transmit data register (TXDATA) are queued in the TXFIFO.

Every character is preceded by a start bit which corresponds to a low logic level for one bit period. The character is terminated by a configurable number of stop bits.

Character transmission procedure

To transmit a character, follow the sequence below:

1. Program the number of stop bits in TXCTRL.
2. Select the desired baud rate using the DIV register.
3. Enable the USART by writing the TXEN bit in TXCTRL register to 1.
4. Wait until the FULL flag in TXDATA register is cleared. The FULL flag is cleared by hardware to indicate that
   - the TXFIFO is not full
   - the next data can be written to the TXDATA register without overwriting the previous data

   When the TXFIFO is full, the FULL flag stays at ‘1’ even after a write operation to TXDATA register, and the byte written is lost. It is cleared when the TXFIFO is not full.

   Alternatively, interrupts can be generated and data can be written to the FIFO when the TXFIFO threshold is reached. In this case, the CPU can write a block of data defined by the programmed trigger level.
5. Write the data to send in the TXDATA register. Repeat step 4 and 5 for each data to be transmitted.
6. When the last data is written to the USART_TDR register, wait until FULL flag in TXDATA register is cleared. This check is required to avoid corrupting the last transmission when the UART is disabled.

4.3.3 UART receiver

Start bit detection

Character reception

During an USART reception, data are shifted out least significant bit first (default configuration) through the RX pin.

Character reception procedure

To receive a character, follow the sequence below:

1. Program the number of stop bits in TXCTRL.
2. Select the desired baud rate using the DIV register.

3. Enable the USART by writing the RXEN bit in RXCTRL register to 1.
7. Wait until the EMPTY flag in RXDATA register is set. The
8. Enable the USART by writing the TXEN bit in TXCTRL register to 1.
9. Wait until the FULL flag in TXDATA register is cleared. The EMPTY flag is set by hardware to indicate that
   - the RXFIFO is not empty
   - the data can be read from the RXDATA register

   The EMPTY flag is cleared when the TXFIFO is not full.

   Alternatively, interrupts can be generated and data can be written to the FIFO when the TXFIFO threshold is reached. In this case, the CPU can write a block of data defined by the programmed trigger level.

4.4 UART registers

4.4.1 Transmit data register (TXDATA)

Address offset: 0x00

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FULL	Reserved														
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DATA							
								w							

31	**FULL**	Transmit FIFO full flag This bit is read-only. The FULL flag indicates whether the transmit FIFO is able to accept new entries. When full, writes to data are ignored. 0: TX FIFO is not full 1: TX FIFO is full
7:0	**DATA**	Data field. These bits are write-only. When written, puts the value into TX FIFO. When read, the bits will always be zero. These bits are set and cleared by software.

4.4.2 Receive data register (RXDATA)

Address offset: 0x04

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EMPTY	Reserved														
r															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DATA							
								r							

31	**EMPTY**	Receive FIFO empty flag This bit is read-only. The EMPTY flag indicates whether the reception FIFO is able to accept new entries. 0: RX FIFO is not empty 1: RX FIFO is empty
7:0	**DATA**	Data field. These bits are read-only. Reading from this register will automatically deque one entry from the RX FIFO.

4.4.3 Transmit control register (TXCTRL)

Address offset: 0x08

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TXCNT		
													rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														NSTOP	TXEN
														rw	rw

18:16	**TXCNT**	Transmit FIFO watermark trigger These bits are read-and-write. These bits specify the threshold at which the TX FIFO watermark interrupt triggers.
1	**NSTOP**	Stop bit configuration. This bit is read-and-write. 0: one stop bit 1: two stop bits
0	**TXEN**	Transmission enable control. This bit is read-and-write. 0: TX disabled 1: TX enabled

4.4.3 Receive control register (RXCTRL)

Address offset: 0x0C

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													RXCNT		
													rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															RXEN
															rw

18:16	**RXCNT**	Receive FIFO watermark trigger These bits are read-and-write. These bits specify the threshold at which the RX FIFO watermark interrupt triggers.
0	**RXEN**	Reception enable control. This bit is read-and-write. 0: RX disabled 1: RX enabled

5 Inter-Integrated Circuit (I2C) Interface

5.1 Introduction

The I2C (inter-integrated circuit) bus interface handles communications to the serial I2C bus. It provides multi-master capability, and controls all I2C bus-specific sequencing, protocol, arbitration and timing. It supports Standard-mode (Sm), Fast-mode (Fm) and Fast-mode Plus (Fm+).

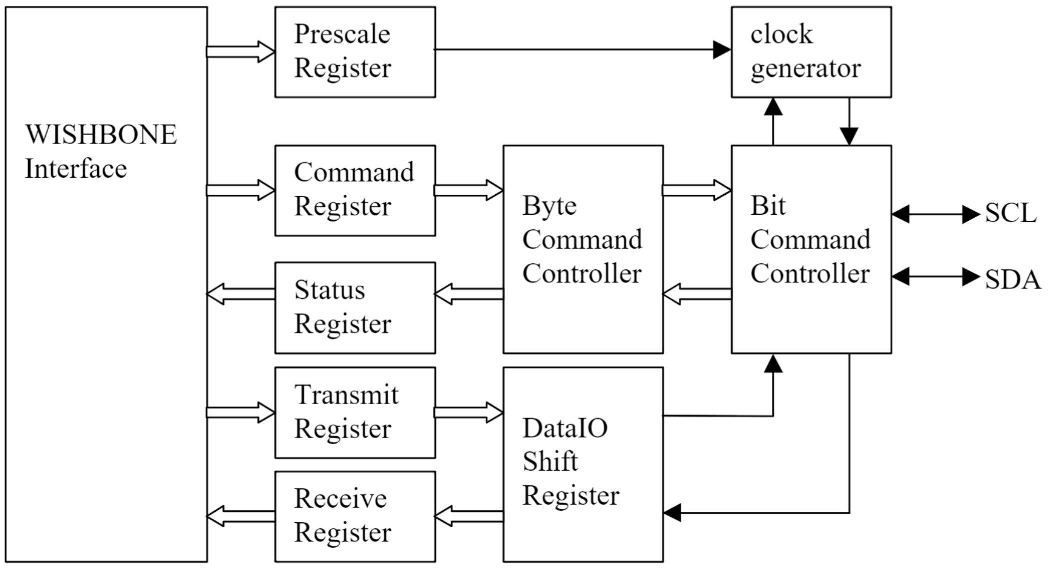
5.2 I2C main features

- I2C bus specification compatibility:
  - Slave and master modes
  - Multimaster capability
  - Standard-mode (up to 100 kHz)
  - Fast-mode (up to 400 kHz)
  - Fast-mode Plus (up to 1 MHz)
  - 7-bit and 10-bit addressing mode

Compatible with Philips I2 C standard • Multi Master Operation • Software programmable clock frequency • Clock Stretching and Wait state generation • Software programmable acknowledge bit • Interrupt or bit-polling driven byte-by-byte data-transfers • Arbitration lost interrupt, with automatic transfer cancelation • Start/Stop/Repeated Start/Acknowledge generation • Start/Stop/Repeated Start detection • Bus busy detection • Supports 7 and 10bit addressing mode • Operates from a wide range of input clock frequencies • Static synchronous design

5.3 I2C functional description

Figure 6. I2C block diagram



5.3.1 I2C signals

I2C communications require two pins: Serial Clock (SCL) and Serial Data (SDA):

- **SCL** (Serial Clock)

  SCL is the bidirectional clock signal. When the device is in master mode, the device drives the SCL pin. When the device is in slave mode, the SCL pin is in input mode.

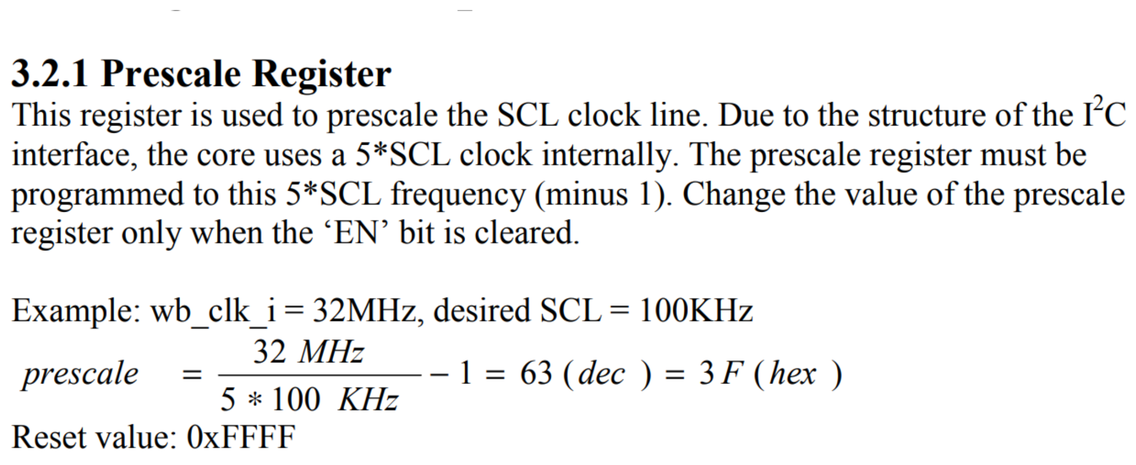
- **SDA** (Serial Data)

  SDA is the bidirectional data signal.

I2C operates in open-drain(OD) mode, which requires a pull-up resistor to the highest positive voltage supply on SCL and SDA pins.

5.3.2 I2C clock requirements

The Clock Generator generates an internal 4*Fscl clock enable signal which triggers all synchronous elements in the Bit Command Controller. It also handles clock stretching needed by some slaves.



3.2.1 Prescale Register

This register is used to prescale the SCL clock line. Due to the structure of the I^{2}C interface, the core uses a 5*SCL clock internally. The prescale register must be programmed to this 5*SCL frequency (minus 1). Change the value of the prescale register only when the 'EN' bit is cleared.

Example: wb_clk_i = 32MHz, desired SCL = 100KHz

$$prescale = \frac{32\ MHz}{5 * 100\ KHz} - 1 = 63\ (dec) = 3F\ (hex)$$

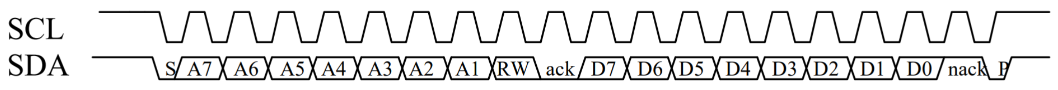
Reset value: 0xFFFF

5.3.3 I2C Communication Flow

Normally, a standard communication consists of four parts:

1. START signal generation
2. Slave address transfer
3. Data transfer
4. STOP signal generation

Figure 7. I2C frame



START signal

When the bus is free/idle, meaning no master device is engaging the bus (both SCL and SDA lines are high), a master can initiate a transfer by sending a START signal. A START signal is defined as a high-to-low transition of SDA while SCL is high. The START signal denotes the beginning of a new data transfer. A Repeated START is a START signal without first generating a STOP signal. The master uses this method to communicate with another slave or the same slave in a different transfer direction (e.g. writing to device to reading from device) without releasing the bus.

The core generates a START signal when the STA bit in the Status Command Register (STAT_CMD) is set and the RD or WR bits are set. Depending on the current status of the SCL line, a START or Repeated START is generated.

Slave address transfer

The first byte of data transferred by the master immediately after the START signal is the slave address.

This is a seven-bit calling address followed by a R/W bit. The R/W bit signals the slave data transfer direction. No two slaves in the system can have the same address. Only the slave with an address that matches the one transmitted by the master will respond by returning an acknowledge bit by pulling the SDA low at the nineth SCL clock cycle.

The core treats a Slave Address Transfer as any other write action. Store the slave device’s address in the Data Register (DATA) and set the WR bit in the Status Command Register (STAT_CMD). The core will then transfer the slave address on the bus.

Data transfer

Once successful slave addressing is achieved, the data transfer can proceed on a byte-by-byte basis in the direction specified by the R/W bit sent by the master. Each transferred byte is followed by an acknowledge bit on the nineth SCL clock cycle.

If the slave signals a No Acknowledge, the master can generate a STOP signal to abort the data transfer or generate a repeated START signal and start a new transfer cycle.

If the master, as the receiving device, does not acknowledge the slave, the slave releases the SDA line for the master to generate a STOP or repeated START signal.

For writing data to a slave, store the data to transmit in the Data Register (DATA) and set the WR bit in the Status Command Register (STAT_CMD). For reading data from a slave, set the RD bit in the Status Command Register (STAT_CMD). During a transfer the core set the TIP flag, indicating that a Transfer is In Progress and the peripheral is busy. When the transfer is done the TIP flag is reset, the IF flag set and, when enabled, an interrupt is generated. The Data Register (DATA) contains valid data after the IF flag has been set. The user may issue a new write or read command when the TIP flag is reset.

STOP signal

The master can terminate the communication by generating a STOP signal. A STOP signal is defined as a low-to-high transition of SDA while SCL is high.

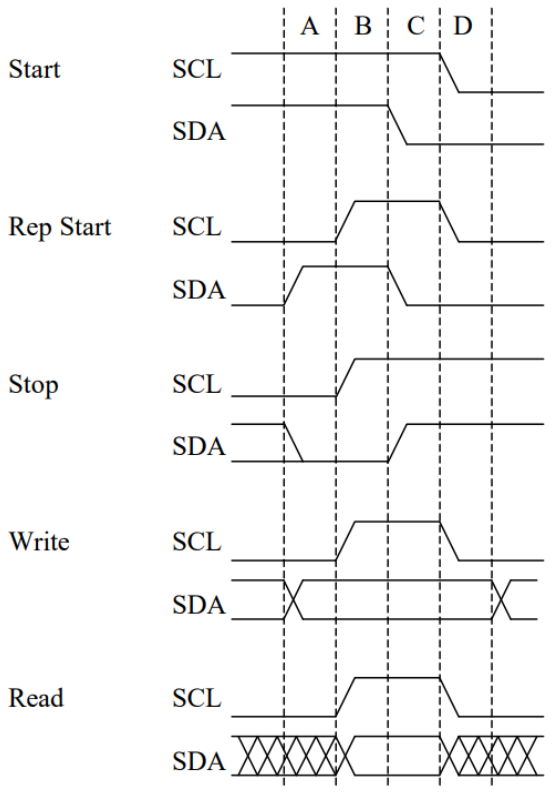
Mode of Operation

In Master mode, the I2C interface initiates a data transfer and generates the clock signal. A serial data transfer always begins with a START condition and ends with a STOP condition. Both START and STOP conditions are generated in master mode by software.

In Slave mode, the interface is capable of recognizing its own addresses (7 or 10-bit), and the general call address. The general call address detection can be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, MSB first. The first byte(s) following the START condition contain the address (one in 7-bit mode, two in 10-bit mode). The address is always transmitted in Master mode. A ninth clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter.

Figure 8. I2C conditions



5.3.4 Data Reception

1) generate start signal

2) write slave address + write bit

3) receive acknowledge from slave

4) write memory location

5) receive acknowledge from slave

6) generate repeated start signal

7) write slave address + read bit

8) receive acknowledge from slave

9) read byte from slave

10) write no acknowledge (NACK) to slave, indicating end of transfer

11) generate stop signal

5.3.5 Data Transfer

1) generate start command

2) write slave address + write bit

3) receive acknowledge from slave

4) write data

5) receive acknowledge from slave

6) generate stop command

5.4 I2C registers

This section gives a detailed description of the GPIO registers.

The peripheral registers can be written in word, half-word or byte mode.

5.4.1 Prescaler low register (PRESCAL_LO)

Address offset: 0x00

Reset value: 0x0000_00FF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								PRESCAL_LO							
								rw							

7:0	**PRESCAL_LO**	Lower part of the prescaler value. These bits are set and cleared by software.

5.4.2 Prescaler high register (PRESCAL_HI)

Address offset: 0x04

Reset value: 0x0000_00FF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved								PRESCAL_HI							
								rw							

7:0	**PRESCAL_HI**	Higher part of the prescaler value. These bits are set and cleared by software.

5.4.3 Control register (CTRL)

Address offset: 0x08

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved								EN	IEN	Reserved					
								rw	rw						

7	**EN**	I2C core enable bit. These bits are set and cleared by software. 0: core disabled 1: core enabled
6	**IEN**	I2C interrupt enable bit. These bits are set and cleared by software. 0: interrupt disabled 1: interrupt enabled

5.4.4 Data register (DATA)

Address offset: 0x0C

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved								DATA							
								rw							

7:0	**DATA**	Data field. When read, returns the last data received by I2C RX; when written, puts the value into I2C TX buffer. When used with I2C START condition, the LSB is I2C R/W mode bit. These bits are set and cleared by software. Note that writing and reading are operating on different fields. 0: write command 1: read command

5.4.5 Status command register (STAT_CMD)

Address offset: 0x10

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								RXACK	BUSY	RD	WR	ACK	Resv.	TIP	IF
								STA	STO						
								r	r	w	w	w	r	r	rw
								w	w						

7	**RXACK / STA**	Receive acknowledgment status / start bit. When read, return the received acknowledge status from the slave; when written, generate (repeated) start condition These bits are set by software, and set and cleared by hardware. When read: 0: acknowledge received (ACK) 1: no acknowledge received (NACK) When written: 0: do not generate START (no effect) 1: generate START
6	**BUSY / STO**	Busy / stop bit. When read, return the bus status; when written, generate stop condition These bits are set by software, and set and cleared by hardware. These bits are set and cleared by software and hardware. When read: 0: detected STOP condition 1: detected START condition When written:

		0: do not generate STOP (no effect) 1: generate STOP
5	**RD**	Initiate a read from slave operation. When written: 0: do nothing 1: start a read operation
4	**WR**	Initiate a write to slave operation. When written: 0: do nothing 1: start a write operation
3	**ACK**	Send acknowledge bit When written: 0: do nothing 1: send acknowledge bit
1	**TIP**	Indicate if a transfer is in progress. When read: 0: transfer is complete 1: transfer in progress
0	**IF**	Indicate if an interrupt is pending Will cause a processor interrupt request if IEN bit is set When read: 0: transfer is complete 1: transfer in progress

6 Quad-Serial Peripheral Interface (QUADSPI)

6.1 Introduction

The QUADSPI is a specialized communication interface targeting single, dual or quad SPI Flash memories.

It can operate in any of the three following modes:

- indirect mode: all the operations are performed using the QSPI registers
- status polling mode: the external Flash memory status register is periodically read and an interrupt can be generated in case of flag setting
- memory-mapped mode: the external Flash memory is mapped to the device address space and is seen by the system as if it was an internal memory

6.2 QUADSPI main features

- Three functional modes: indirect, status-polling, and memory-mapped
- Fully programmable opcode for both indirect and memory-mapped mode
- Fully programmable frame format for both indirect and memory mapped mode
- Integrated FIFO for reception and transmission
- 8, 16, and 32-bit data accesses are allowed
- Interrupt generation on FIFO threshold, timeout, operation complete, and access error

6.3 QUADSPI functional description

6.3.1 QUADSPI signals

QUADSPI communications require the following pins:

- **SCLK** (Clock)

  SCLK is the clock signal. When the device is in master mode, the device drives the SCLK pin. When the device is in slave mode, the SCLK pin is in input mode.

- **IO0** (Data Line 0)

  IO0 is the first bidirectional data pin.

- **IO1** (Data Line 1)

  IO1 is the second bidirectional data pin.

- **IO2** (Data Line 2)

  IO2 is the third bidirectional data pin.

- **IO3** (Data Line 3)

  IO3 is the fourth bidirectional data pin.

- **CSn** (Chip Select)

CSn is the active-low chip select signal. When the device is in master mode, the device drives the CSn pin. When the device is in slave mode, the CSn pin is in input mode.

The QUADSPI bus allows communication between one master device and one or more slave devices. The bus consists of at least two wires - one for the clock signal and the other for synchronous data transfer. Other signals can be added depending on the data exchange between QUADSPI nodes and their slave-select signal management.

6.3.1 QUADSPI command sequence

The QUADSPI communicates with the Flash memory using commands. Each command can include five phases: instruction, address, alternate byte, dummy, data. Any of these phases can be configured to be skipped, but at least one of the instruction, address, alternate byte, or data phase must be present.

CSn falls before the start of each command and rises again after each command finishes.

7 Serial Peripheral Interface (SPI)

7.1 Introduction

The SPI interface can be used to communicate with external devices using the SPI protocol.

The serial peripheral interface (SPI) protocol supports half-duplex, full-duplex and simplex synchronous, serial communication with external devices. The interface can be configured as master and in this case it provides the communication clock (SCLK) to the external slave device.

7.2 SPI main features

- Master or slave operation
- Full-duplex synchronous transfers
- 4 to 16-bit data size selection
- Master mode baud rate prescalers up to fPCLK/2
- NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
- Programmable clock polarity and phase
- Programmable data order with MSB-first or LSB-first shifting
- Dedicated transmission and reception flags with interrupt capability
- SPI bus busy status flag
- Two 32-bit embedded Rx and Tx FIFOs with DMA capability

7.3 SPI functional description

7.3.1 SPI signals

The SPI allows synchronous, serial communication between the MCU and external devices. Application software can manage the communication by polling the status flag or using dedicated SPI interrupt.

- **SCLK** (Clock)

  SCLK is the clock signal. When the device is in master mode, the device drives the SCLK pin. When the device is in slave mode, the SCLK pin is in input mode.

- **MISO** (Master In / Slave Out Data)

  In the general case, this pin is used to transmit data in slave mode and receive data in master mode.

- **MOSI** (Master Out / Slave In Data)

  In the general case, this pin is used to transmit data in master mode and receive data in slave mode.

- **CSn** (Chip Select)

  Depending on the SPI and CSn settings, this pin can be used to select an individual slave device for communication.

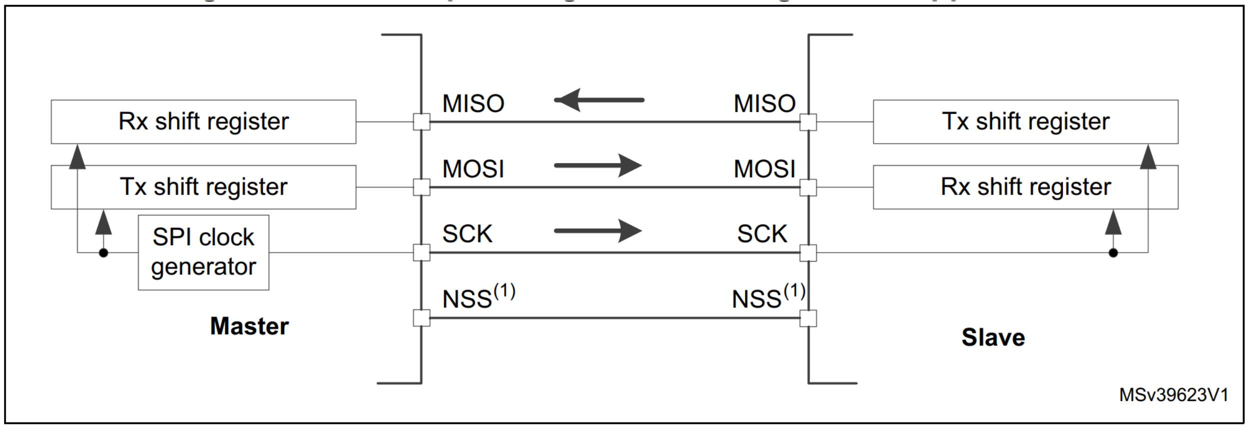
The SPI bus allows communication between one master device and one or more slave devices. The bus consists of at least two wires - one for the clock signal and the other for synchronous data transfer. Other signals can be added depending on the data exchange between SPI nodes and their slave-select signal management.

7.3.2 Communications between one master and one slave

The SPI allows the MCU to communicate using different configurations, depending on the device targeted and the application requirements. These configurations use 2 or 3 wires (with software NSS management) or 3 or 4 wires (with hardware NSS management). Communication is always initiated by the master

Full-duplex communication

By default, the SPI is configured for full-duplex communication. In this configuration, the shift registers of the master and slave are linked using two unidirectional lines between the MOSI and the MISO pins. During SPI communication, data is shifted synchronously on the SCK clock edges provided by the master. The master transmits the data to be sent to the slave via the MOSI line and receives data from the slave via the MISO line. When the data frame transfer is complete (all the bits are shifted) the information between the master and slave is exchanged.



8 Timer (TIM)

9 Pulse-Width Modulation (PWM)

10 Controller Area Network (CAN)

11 Ethernet (ETH)

12 Analog to Digital Converter (ADC)

13 Digital to Analog Converter (DAC)

14 BootROM

15 Serialized TileLink (SERTL)