

mmRISC-1

Technical Reference Manual

RISC-V RV32IMAFCore for MCU

Rev.07



Munetomo Maruyama

Revision History				Note
Rev	Date	Author	Description	
01	Nov.19 2021	MM	1st Release.	
02	Dec.31 2021	MM	(1) Added MRET and WFI descriptions. (2) Supported Questa as logic simulator. To do so, add an option -voptargs="+acc" in vsim command. (3) Supported Initialization of Instruction RAM in FPGA using .mif file. A conversion tool hex2mif is added in tools directory. (4) Updated JTAG interface schematic. (5) Changed operation of application mmRISC_SampleCPU. (6) Add a retro text video game StarTrek as an application.	
03	Apr.30 2022	MM	(1) verilog/common/defines.v is divided into defines_core.v for mmRISC Core and defines_chip.v for Chip System. (defines.v is not used any more.) (2) Added I2C, SPI, and SDRAM interface in Top Layer. (3) Added know-hows to use Raspberry Pi as a Development Environment including OpenOCD interface. (4) In application mmRISC_SampleCPU, added access of Acceleration Sensor on MAX10-Lite board through I2C interface. (5) Added new application mmRISC_TouchLCD which handles Adafruit-2-8-tft-touch-shield-v2 with Resistive Touch Panel or Capacitive Touch Panel for Arduino. (6) In each sample program, baud rate of UART is unified to 115200bps.	
04	Nov.20 2023	MM	(1) Added cJTAG (2-wire compact JTAG) as the debug interface. You can select the debug interface from JTAG or cJTAG. (2) Added alternative Halt-On-Reset, controlled by a hardware input signal level, instead of the one in standard RISC-V debug function. (3) Added a new JTAG DR register for user's multi purposes, for example, to configure the operation modes of the SoC from JTAG/cJTAG interface. (4) The number of 32bit secure code for authentication is expanded from one to two. (5) Supported low power mode (STBY). (6) Added precise verification methods for floating point operations. Corrected RTL code in conversion from float to int (cpu_fpu32.v). (7) Added a application program; Tic-Tac-Toe AI Game on Touch LCD panel. (8) Re-write this document by using TexShop.	
05	Sep.01 2024	MM	(1) You can switch the debug interface between 4-wire JTAG or 2-wire cJTAG by setting the signal level of "enable_cjtag" in chip_top_wrap.v. This signal is connected to GPIO2[6] corresponding to SW6 on DE10-Lite board. Therefore from this version onward, there are only two FPGA configurations; one is RV32IMAFc_JTAGcJTAG, which unifies previous RV32IMAFc_JTAG and RV32IMAFc_cJTAG; the other is RV32IMAC_JTAGcJTAG, which unifies previous RV32IMAC_JTAG and RV32IMAFc_cJTAG. (2) Even in the previous version, the detection of escape sequences in "cJTAG_2_JTAG.v" operated correctly for cJTAG waves generated by OpenOCD. However, the logic could not correctly detect some cJTAG escape waves generated by other generic debuggers, such as commercial RISC-V development IDEs. Additionally, in "debug_dtm_jtag.v", the IR in the JTAG TAP controller set to 0x1F caused the TDO output to be fixed high, which led to failures in JTAG chain tests when shifting long bit chains. These bugs have been fixed in this version, and now the mmRISC-1 can connect to generic commercial IDEs correctly. (3) Fixed a bug in the pipeline stall control for the C.FSWSP instruction (RV32FC). In the previous version, if the destination of the previous instruction was FRn, and the C.FSWSP stored the same FRn, the C.FSWSP did not stall correctly. (4) In the sample application programs, the interrupt handler routines have been re-written using "__attribute__((interrupt))", and the control of interrupt nesting has been moved to the C routine "interrupt.c" from the assembler routine "startup.S".	
06	Jan.26 2025	MM	(1) The address expression for the TINFO Register (CSR) of the Debug Trigger Module has been corrected from 0x7a3 to 0x7a4 in the document. This error was only present in the document and did not affect the RTL logic. (2) The cause field in the DCSR Register (CSR) of the Debug related function was previously set to 3' b011 (halt request) even when the cause was a Hardware Break Point. This has been corrected in the RTL description to 3' b010 (breakpoint exception by hardware trigger module) when the cause is a Hardware Break Point. The corrected RTL file is cpu_pipeline.v. This cause field is used by some commercial IDEs to implement specific I/O emulation, such as printf().	
07	Feb.01 2025	MM	(1) In the section of "Connection to the target FPGA as 2-wire cJTAG interface", added a figure to show the connection between the cJTAG debugger probe and the cJTAG signals.	

Contents

1 mmRISC-1 CPU Core	1
1.1 Overview	2
1.2 Block Diagram	3
1.3 Programmer's Model	4
1.4 Configurable Options	5
1.5 RTL Files and Structure	6
1.6 Input / Output Signals of mmRISC-1	8
1.7 Clock & Reset	11
1.8 Exceptions	12
1.9 Interrupts and INTC (Interrupt Controller)	14
1.10 Memory Model	22
1.11 CSR Map List	23
1.12 Other Register Map List except for CSR	25
1.13 Machine Mode CSR	27
1.14 Machine Mode MTIME (Memory Mapped)	31
1.15 INTC (Interrupt Controller) CSR	33
1.16 Floating Point CSR	35
1.17 Debug CSR	37
1.18 Debug Trigger Module CSR	38
1.19 DTM (Debug Transport Module) JTAG Register	41
1.20 DM (Debug Module) Register	43
1.21 DM (Debug Module) Register : Abstract Command	47
1.22 DM (Debug Module) Register : Abstract Command	49
1.23 CPU Pipeline Structure	50
1.24 FPU Pipeline Structure	52
1.25 32bit ISA Specifications	54
1.26 16bit ISA Specifications	56
1.27 32bit ISA Code Assignments	57
1.28 16bit ISA Code Assignments	59
1.29 JTAG Data Register for User	60
1.30 Halt on Reset	61
1.31 cJTAG (Compact JTAG)	63
1.32 Security (Authentication)	69
1.33 Low Power Mode	70
2 mmRISC-1 Tiny SoC for FPGA	73
2.1 Overview of SoC	74
2.2 Block Diagram of SoC	75
2.3 RTL Files and Structure of SoC	77
2.4 Input / Output Signals of SoC	79

2.5	Clock & Reset for SoC	80
2.6	Memory Map	82
2.7	Interrupt Assignment	83
2.8	Multi-Layer AHB Bus Matrix	84
2.9	Peripheral : RAM (Instruction / Data)	87
2.10	Peripheral : SDRAM Controller)	88
2.11	Peripheral : Peripheral : Interrupt Generator (INT_GEN)	89
2.12	Peripheral : UART	91
2.13	Peripheral : I2C	93
2.14	Peripheral : SPI	96
2.15	Peripheral : GPIO	99
2.16	Using Miscellaneous Features of mmRISC-1	101
3	mmRISC-1 Simulations and Verifications	105
3.1	Development Environment and Tools	106
3.2	Github Repository Files	107
3.3	RTL Verification Methods	108
3.4	Verification of Floating Point Operations	112
4	mmRISC-1 FPGA Implementation	119
4.1	FPGA Implementation	120
4.2	JTAG/cJTAG Debugger Interface for OpenOCD	126
4.3	SamplePrograms	134
5	Referenced Documents	141
5.1	Referenced Documents	142
		143

List of Figures

1.1	mmRISC-1 Block Diagram ¹	3
1.2	Programmers Model for RV32IM[A][F][C]	4
1.3	Programmers Model for RV32C/RV32CF (Compressed Instruction)	4
1.4	Register Number Indication for RV32C/RV32CF (Compressed Instruction)	4
1.5	RTL Files	7
1.6	Reset Structure	11
1.7	INTC (Interrupt Controller)	14
1.8	Little Endian Memory Organization	22
1.9	CPU Pipeline	51
1.10	CPU Pipeline Logic Structure	51
1.11	FPU Pipeline	53
1.12	FPU Pipeline Logic Structure	53
1.13	Block Diagram of JTAG Chain in mmRISC-1	60
1.14	Halt on Reset	62
1.15	Example of Control Logic for Halt on Reset	62
1.16	cJTAG Escape Sequence and Activation Sequence	64
1.17	cJTAG OSCAN1 Sequence	64
1.18	cJTAG Halt on Reset	65
1.19	External Adapter to convert JTAG to cJTAG	66
1.20	cJTAG Implementation for Silicon (SoC)	67
1.21	cJTAG Implementation for FPGA and Test Bench	68
1.22	Security Configuration	69
1.23	Block Diagram of Low Power Support	70
1.24	Timing of Low Power Support	70
2.1	Block Diagram of mmRISC-1 Tiny SoC for FPGA with 4-wire JTAG debug interface (if "enable_cjtag" is fixed to low level)	76
2.2	Block Diagram of mmRISC-1 Tiny SoC for FPGA with 2-wire cJTAG debug interface (if "enable_cjtag" is fixed to high level)	76
2.3	Structure of RTL Files of SoC	78
2.4	Clock Structure in Tiny SoC	80
2.5	Glitch-less Clock Selector	80
2.6	Reset Structure in Tiny SoC	81
2.7	I/O Buffers for I2C	93
3.1	Questa Screen Shot	109
4.1	Terasic DE10-Lite Board & OpenOCD JTAG Adapter	121
4.2	Intel Quartus Prime Lite Edition	121
4.3	Operation Positions to use special features	125

4.4	Outline of OpenOCD JTAG/cJTAG Adapter (Left:Breadboard Type, Right:Print Board Type)	126
4.5	Fundamental Connection of OpenOCD JTAG Adapter	127
4.6	Detail Schematics of OpenOCD JTAG Adapter	127
4.7	Connection of OpenOCD JTAG Adapter and PC	128
4.8	Connection between the cJTAG debugger probe and cJTAG signals	129
4.9	Supplement for cJTAG	130
4.10	Connection of Raspberry Pi and the FPGA	133
4.11	Eclipse Windows	135
4.12	Demo Program "mmRISC_TouchLCD"	138
4.13	Adafruit 2.8" TFT Touch Shield for Arduino (Left: Product ID 1651 Resistive Touch Version; Right: Product ID 1947 Capacitive Touch Version) .	139
4.14	Startup Screen of TIc-Tac-Toe Game	139
4.15	Ending Screen of TIc-Tac-Toe Game	139

List of Tables

1.1	mmRISC-1 Overview	2
1.2	Configurable Options	5
1.3	RTL Files	6
1.4	Input / Output Signals of mmRISC-1 (System)	8
1.5	Input / Output Signals of mmRISC-1 (JTAG)	8
1.6	Input / Output Signals of mmRISC-1 (Configuration)	8
1.7	Input / Output Signals of mmRISC-1 (Instruction Bus)	9
1.8	Input / Output Signals of mmRISC-1 (Data Bus)	9
1.9	Input / Output Signals of mmRISC-1 (Atomic LR/SC Monitor) ¹	10
1.10	Input / Output Signals of mmRISC-1 (Debug System Access Bus)	10
1.11	Input / Output Signals of mmRISC-1 (Interrupt)	10
1.12	Input / Output Signals of mmRISC-1 (MTIME)	10
1.13	Exceptions	12
1.14	Priority of Synchronous Exception	13
1.15	Machine Mode CSR	23
1.16	INTC (Interrupt Controller) CSR	23
1.17	FPU CSR	24
1.18	Debugger CSR	24
1.19	Debug Trigger Module CSR	24
1.20	Machine Mode MTIME Registers (Memory Mapped Registers; So far, recommended Base Address is 0x49000000)	25
1.21	DTM (Debug Transport Module) JTAG Tap Registers; Address is 5bit IR	25
1.22	DM (Debug Module) Registers; Address comes from DMI bus	25
1.23	DM (Debug Module) Abstract Command Registers; Address comes from DMI bus	26
1.24	DM (Debug Module) System Bus Access Registers; Address comes from DMI bus	26
1.25	MVENDORID	27
1.26	MARCHID	27
1.27	MIMPID	27
1.28	MSTATUS	27
1.29	MISA	28
1.30	MIE	28
1.31	MIP	28
1.32	MTVEC	28
1.33	MSCRATCH	29
1.34	MEPC	29
1.35	MCAUSE	29
1.36	MTVAL	29
1.37	MCYCLE / MCYCLEH	29

1.38 MINSTRET / MINSTRETH	29
1.39 MCOUNTINHIBIT	30
1.40 Read-Only Mirror Registers	30
1.41 MTIME_CTRL	31
1.42 MTIME_DIV	31
1.43 MTIME / MTIMEH	31
1.44 MTIMECMP / MTIMECMPPH	32
1.45 MSOFTIRQ	32
1.46 MINTCURLVL	33
1.47 MINTPRELVL	33
1.48 MINTCFGENABLE 0/1	33
1.49 MINTCFGSENSE 0/1	33
1.50 MINTPENDING 0/1	33
1.51 MINTCFGPRIORITY 0/1/2/3/4/5/6/7	34
1.52 FCSR	35
1.53 FRM	35
1.54 FFLAGS	35
1.55 FCONV	36
1.56 DCSR	37
1.57 DPC	37
1.58 TSELECT	38
1.59 TDATA1	38
1.60 TDATA2	38
1.61 TINFO	38
1.62 MCONTROL	39
1.63 ICOUNT	40
1.64 IDCODE	41
1.65 BYPASS	41
1.66 DTMCS	41
1.67 DMI	42
1.68 DMSTATUS	43
1.69 DMCONTROL	44
1.70 HARTINFO	44
1.71 HAWINDOWSEL	45
1.72 HAWINDOW	45
1.73 AUTHDATA	45
1.74 HALTSM0	45
1.75 HALTSM1	45
1.76 HALTSM2	46
1.77 HALTSM3	46
1.78 DATA 0/1	47
1.79 COMMAND for Access Register	47
1.80 COMMAND for Access Memory	47
1.81 ABSTRACTCS	48
1.82 SBCS	49
1.83 SBADDRESS0	49
1.84 SBDATA0	49
1.85 RV32I Base Instruction Set Specification	54
1.86 RV32 Zifencei Specification	54
1.87 RV32 Zicsr Specification	54

1.88	RV32M Multiply and Divide Specification	55
1.89	RV32A Atomic Memory Access Specification	55
1.90	RV32F Single Precision Floating Point Operation Specification	55
1.91	RV32 Privileged Instruction Specification	55
1.92	RV32C Compressed Instruction Specification	56
1.93	RV32FC Floating Point Compressed Instruction Specification	56
1.94	RV32I Base Instruction Set Code	57
1.95	RV32 Zifencei Code	57
1.96	RV32 Zicsr Code	57
1.97	RV32M Multiply and Divide Code	58
1.98	RV32A Atomic Memory Access Code	58
1.99	RV32F Single Precision Floating Point Operation Code	58
1.100	RV32 Privileged Instruction Code	58
1.101	RV32C Compressed Instruction Code	59
1.102	RV32FC Floating Point Compressed Instruction Code	59
1.103	JTAG Data Register for User	60
1.104	Signal of cJTAG Interface	63
1.105	cJTAG Pin Control	65
2.1	Overview of mmRISC-1 Tiny SoC for FPGA	74
2.2	RTL Files of SoC	77
2.3	Input / Output Signals of Tiny SoC Top Layer (chip_top_wrap.v)	79
2.4	Memory Map of Tiny SoC	82
2.5	Interrupt Assignment in Tiny SoC	83
2.6	Input / Output Signals of Multi-Layer AHB Bus Matrix)	84
2.7	Input / Output Signals of RAM)	87
2.8	Input / Output Signals of SDRAM Controller)	88
2.9	Input / Output Signals of INT_GEN)	89
2.10	INT_GEN Control Register IRQ_EXT)	89
2.11	INT_GEN Control Register IRQ0)	90
2.12	INT_GEN Control Register IRQ1)	90
2.13	Input / Output Signals of UART	91
2.14	UART_TXD / UART_RXD	91
2.15	UART_CSR	92
2.16	UART_BG0	92
2.17	UART_BG1	92
2.18	Input / Output Signals of I2C	93
2.19	I2C_PRERL	94
2.20	I2C_PRERH	94
2.21	I2C_CTL	94
2.22	I2C_TXR	94
2.23	I2C_RXR	94
2.24	I2C_CR	95
2.25	I2C_SR	95
2.26	Input / Output Signals of SPI	96
2.27	I2C_SPCR	97
2.28	I2C_SPSR	97
2.29	I2C_SPDR	97
2.30	I2C_SPER	98
2.31	I2C_SPCS	98

2.32 Input / Output Signals of GPIO(PORT)	99
2.33 PDR0	99
2.34 PDR1	100
2.35 PDR2	100
2.36 PDD0	100
2.37 PDD1	100
2.38 PDD2	100
2.39 Operations of Halt-on-Reset in the SoC	102
3.1 Development Environment and Tools	106
3.2 Development Tools on ARM macOS	106
3.3 Major Files in GitHub Repository	107
3.4 Scripts for Generating Floating Point Test Cases	113
3.5 OPCODEs in Test Case File	115
3.6 Exception Flag in Test Case File	115
4.1 FPGA Configuration Results of mmRISC-1	122
4.2 FPGA Pin Assignments (1)	124
4.3 FPGA Pin Assignments (2)	124
4.4 FPGA Special Function Pins	125
4.5 Assignments of GPIO of Raspberry Pi 4 Model B as an OpenOCD JTAG Debugger	132
4.6 Scores of Benchmarks of mmRISC-1 core	136

Listings

1.1	An Example of Startup Routine including Exception Handlers and Interrupt Entry Points written in Assembler language	15
1.2	An Example of Interrupt Initializations and Interrupt Handlers written in C language	16
1.3	An Example of Interrupt related routines (assembler and C) involving pairs of dedicated entry point and corresponding handler routine in order to improve the interrupt latency	20
1.4	RTL code of cjttag_adapter.v	66
1.5	I/O Port Control for cJTAG Pins	67
1.6	Authentication Command in OpenOCD Configuration File	69
2.1	An Example of Priority Configuration	85
2.2	An Example of Unacceptable Priority Configuration	85
2.3	An Example of Address Range Definition	85
2.4	Commands to generate RAM initialization Data for FPGA	87
3.1	Output of "riscv-arch-test"	110
3.2	Output of "riscv-tests"	111
3.3	Example of Generated 1-Operand Test Case (testfloat_F2S)	114
3.4	Example of Generated 2-Operand Test Case (testfloat_A)	114
3.5	Example of Generated 3-Operand Test Case (testfloat_N)	114
3.6	Example Result of 1-Operand Test Case (testfloat_F2S)	116
3.7	Example Result of 2-Operand Test Case (testfloat_A)	116
3.8	Example Result of 3-Operand Test Case (testfloat_N)	117
4.1	OpenOCD Configuration File for 4-wire JTAG interface	128
4.2	OpenOCD Configuration File for Olimex ARM-USB-OCD(H)	129
4.3	OpenOCD Configuration File for 2-wire cJTAG interface	130
4.4	OpenOCD Configuration file for Raspberry Pi	131
4.5	Star Trek Game	136
4.6	Tic-Tac-Toe on UART Terminal	139

Chapter 1

mmRISC-1 CPU Core

This chapter describes technical information of mmRISC-1 CPU Core.

1.1 Overview

The "mmRISC-1" is a RISC-V compliant CPU core with RV32IM[A][F]C ISA for MCU. An overview of mmRISC-1 specifications is shown in Table 1.1. "mmRISC" stands for "much more RISC".

Item	Description	Note
Core Name	mmRISC-1	
ISA	RV32IM[A][F]C	configurable
Multi Harts	Multi Harts Supported, 1 to 2^{20}	configurable
Pipeline	3 to 5 stages for Integer, 5 to 6 stages for Floating Point	
32bits Integer Multiply	1 cycle for MUL	
32bits Integer Division	Non-Restoring Method, 33 cycles for DIV/REM	
32bits Floating Point FADD/FMUL etc.	1 cycle for FADD.S/FSUB.S/FMUL.S/ FMADD.S/FMSUB.S/ FNMADD.S/FNMSUB.S	
32bits Floating Point FDIV/FSQRT	Goldschmidt's Algorithm, Convergence Loops = 1 16 (4typ), FDIV: 11 cycles (loops = 4), FDQRT: 19 cycles (loops = 4)	
Debug Support	External Debug Support Ver.0.13.2 with JTAG/cJTAG I/F Run / Stop / Step Abstract Command Access (Reg, Memory) System Bus Access (Memory) Hardware Break Points (Instr/Data) x 4 Instruction Count Break Point x 1 The relationship between the frequency of the system clock and TCK/TCKC does not matter.	Number of Hardware Break is configurable
Privileged Mode	Machine-Mode (M-Mode) only	
Interrupt	64 inputs Vectored Supported 16 priority levels for each	
Counters	64bits MCYCLE (Clock Cycle Counter) 64bits MINSTRET (Instruction Retired Counter) 64bits MTIME (Memory Mapped Interrupt Timer)	
Bus Interface	Instruction Fetch Bus Data and Debugger Abstract Access Bus LR/SC Monitor Bus Debugger System Bus	AHB-Lite
RTL	Verilog-2001 / System Verilog	

Table 1.1: *mmRISC-1 Overview*

1.2 Block Diagram

The block diagram of mmRISC-1 is shown in Figure 1.1. The mmRISC-1 contains multiple CPU Harts and a Debug Module with JTAG/cJTAG interface. Each Hart has an instruction fetch bus, a data access bus, and a monitor bus for LR/SC operation. The Debug Module has a dedicated system bus access capability and controls each Hart (run / stop / step, etc.). The Debug Module can access resources in each Hart such as programmer's model registers (XRn, FRn), SCR (System Control Registers) and memory-mapped devices through data access bus by debugger's abstract commands.

Each Hart consists of an instruction fetch unit with a pre-fetch buffer, a pipeline unit with an instruction decoder, a datapath unit for integer operations, CSR resources, debugger support, and a floating point operation unit. The monitor bus for LR/SC operation exists only when the Atomic ISA is enabled. The floating point operation unit exists only when the floating-point ISA is enabled.

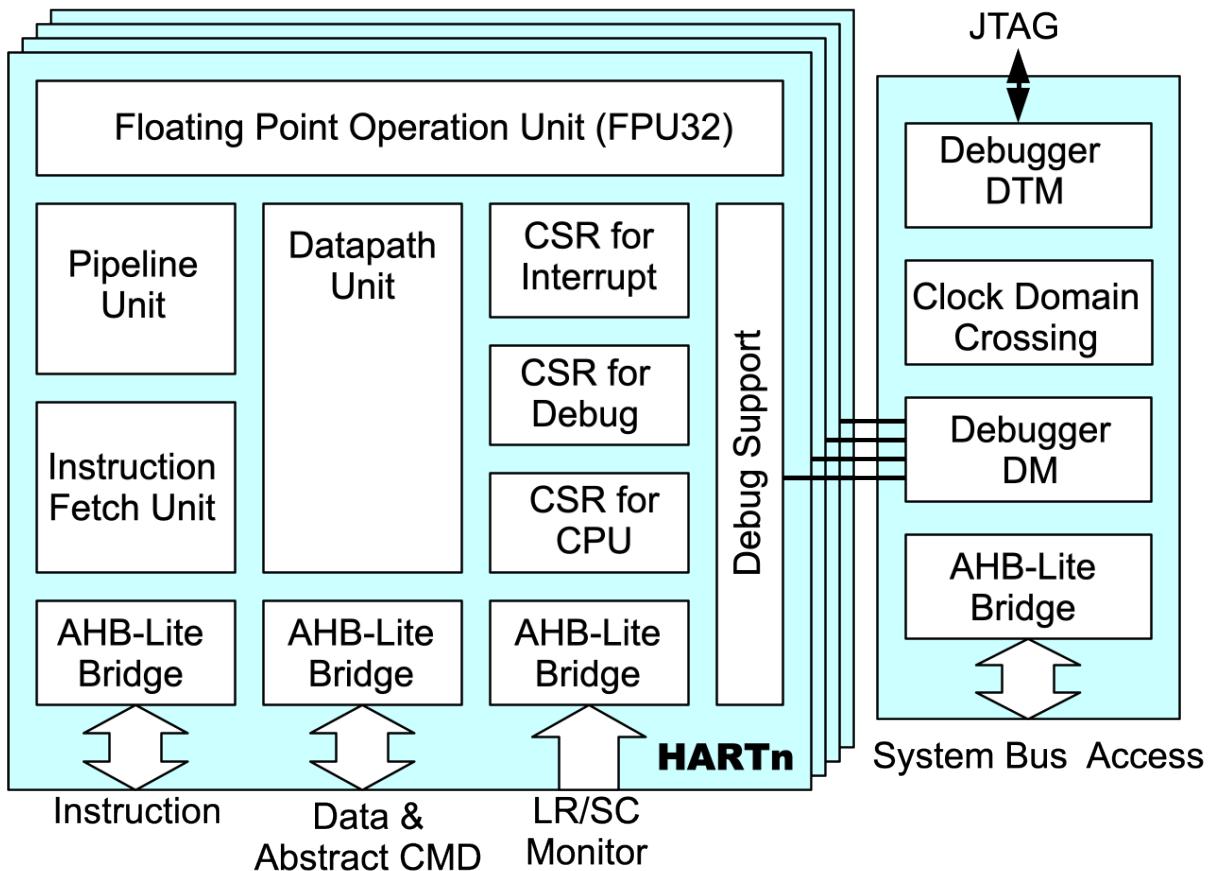


Figure 1.1: mmRISC-1 Block Diagram¹

¹cJTAG is supported by logics at outside of mmRISC core.

1.3 Programmer's Model

The programmer's model for RV32IM[A][F][C] of this core is shown in Figure 1.2. The integer registers xn (or sometimes expressed as XRn) and PC are 32bit length. The floating Point registers fn (or sometimes expressed as FRn) are also 32bit length which exist in the core only when the Floating Point ISA is enabled. The core's XLEN= 32 and FLEN = 32. In addition, Figure 1.3 shows Programmer's model for RV32C/RV32CF(Compressed Instruction Set), and Figure 1.4 shows how those registers are indicated by 3bits fields in the compressed instruction code.

Integer Registers		Floating Point Registers	
31	0	31	0
x0 (zero)		f0 (ft0)	FP Temporary
x1 (ra)		f1 (ft1)	FP Temporary
x2 (sp)		f2 (ft2)	FP Temporary
x3 (gp)		f3 (ft3)	FP Temporary
x4 (tp)		f4 (ft4)	FP Temporary
x5 (t0)		f5 (ft5)	FP Temporary
x6 (t1)		f6 (ft6)	FP Temporary
x7 (t2)		f7 (ft7)	FP Temporary
x8 (s0/fp)		f8 (fs0)	FP Saved Register
x9 (s1)		f9 (fs1)	FP Saved Register
x10 (a0)		f10 (fa0)	FP Argument / Return Value
x11 (a1)		f11 (fa1)	FP Argument / Return Value
x12 (a2)		f12 (fa2)	FP Argument
x13 (a3)		f13 (fa3)	FP Argument
x14 (a4)		f14 (fa4)	FP Argument
x15 (a5)		f15 (fa5)	FP Argument
x16 (a6)		f16 (fa6)	FP Argument
x17 (a7)		f17 (fa7)	FP Argument
x18 (s2)		f18 (fs2)	FP Saved Register
x19 (s3)		f19 (fs3)	FP Saved Register
x20 (s4)		f20 (fs4)	FP Saved Register
x21 (s5)		f21 (fs5)	FP Saved Register
x22 (s6)		f22 (fs6)	FP Saved Register
x23 (s7)		f23 (fs7)	FP Saved Register
x24 (s8)		f24 (fs8)	FP Saved Register
x25 (s9)		f25 (fs9)	FP Saved Register
x26 (s10)		f26 (fs10)	FP Saved Register
x27 (s11)		f27 (fs11)	FP Saved Register
x28 (t3)		f28 (ft8)	FP Temporary
x29 (t4)		f29 (ft9)	FP Temporary
x30 (t5)		f30 (ft10)	FP Temporary
x31 (t6)		f31 (ft11)	FP Temporary
pc	Program Counter		

Figure 1.2: Programmers Model for RV32IM[A][F][C]

Integer Registers		Floating Point Registers	
31	0	31	0
x8 (s0/fp)		f8 (fs0)	FP Saved Register
x9 (s1)		f9 (fs1)	FP Saved Register
x10 (a0)		f10 (fa0)	FP Argument / Return Value
x11 (a1)		f11 (fa1)	FP Argument / Return Value
x12 (a2)		f12 (fa2)	FP Argument
x13 (a3)		f13 (fa3)	FP Argument
x14 (a4)		f14 (fa4)	FP Argument
x15 (a5)		f15 (fa5)	FP Argument
pc	Program Counter		

Figure 1.3: Programmers Model for RV32C/RV32CF (Compressed Instruction)

RVC	XRn	FRn
000	x8 (s0)	f8 (fs0)
001	x9 (s1)	f9 (fs1)
010	x10 (a0)	f10 (fa0)
011	x11 (a1)	f11 (fa1)
100	x12 (a2)	f12 (fa2)
101	x13 (a3)	f13 (fa3)
110	x14 (a4)	f14 (fa4)
111	x15 (a5)	f15 (fa5)

Figure 1.4: Register Number Indication for RV32C/RV32CF (Compressed Instruction)

1.4 Configurable Options

The several specifications of the mmRISC-1 core are configurable as shown in Table 1.2. Each configuration is specified by a parameter definition (`define) or the logic level of each input signal of the mmRISC-1 core.

Where	Class	Name	Description	Default
defines_core.v	`define	`RISCV_ISA_RV32F	Single Floating Point ISA	enabled
defines_core.v	`define	`RISCV_ISA_RV32A	Atomic Access ISA	enabled
defines_core.v	`define	`JTAG_IDCODE	JTAG ID Code	32'h16d6d001 Version = 0x1 PartNo.=0x6d6d (ASCII "mm") ManuID = 0
defines_core.v	`define	`MVENDORID	CSR MVENDORID	32'h00000000
defines_core.v	`define	`MARCHID	CSR MARCHID	32'h6d6d3031
defines_core.v	`define	`MIMPID	CSR MIMPID	32'h00000010
defines_core.v	`define	`TRG_CH_BUS	Count of Type 2 Trigger ch (Type 3 Trigger is 1ch)	4
defines_core.v	`define	`UNAVAILABLE _WHEN_STBY	During STBY, DM_STATUS shows all harts are unavailable	disabled (comment out)
defines_chip.v	`define	`HART_COUNT	Hart Counts in mmRISC-1	1
defines_chip.v	`define	`USE_FORCE_HALT _ON_RESET	Use Halt-On-Reset feature controlled by hardware input signal level	enabled
mmRISC.v	input signal	RESET_VECTOR [0:`HART_COUNT-1]	Reset Vector Address for each Hart	32'h90000000
mmRISC.v	input signal	DEBUG_SECURE	Debug is secured or not	Input from Top Layer
mmRISC.v	input signal	DEBUG_SECURE _CODE_0 [31:0]	Debug Secure Code 0 to be matched with CSR AUTHDATA	32' h12345678
mmRISC.v	input signal	DEBUG_SECURE _CODE_1 [31:0]	Debug Secure Code 1 to be matched with CSR AUTHDATA	32' hbeefcafe

Table 1.2: Configurable Options

Note : The configurable option of ``ENABLE_CJTAG'' is no longer used. You can switch the debug interface between 4-wire JTAG or 2-wire cJTAG by setting the signal level of "enable_cjtag" chip_top_wrap.v at SoC top level RTL. This signal is connected to GPIO2[6] corresponding to SW6 on DE10-Lite board. Therefore from this version onward, there are only two FPGA configurations; one is RV32IMAFc_JTAGcJTAG, which unifies previous RV32IMAFc_JTAG and RV32IMAFc_cJTAG; the other is RV32IMAC_JTAGcJTAG, which unifies previous RV32IMAC_JTAG and RV32IMAFc_cJTAG.

1.5 RTL Files and Structure

The RTL files of the mmRISC-1 are shown in Table 1.3, and the RTL structure in the mmRISC-1 is shown in Figure 1.5. The top layer of the mmRISC-1 (mmRISC.V) has two large domains, one is a multiple CPU (Hart) block (cpu_top.v) and the other is a Debug block (debug_top.v). As for the periodic interrupt timer MTIME, it is defined in the CSR but it is outside of the CPU core because it is connected to the system bus as a memory mapped peripheral.

Basically, the debug interface is 4-wire JTAG, but you can use 2-wire cJTAG (Compact JTAG) instead. If you use cJTAG, please add a “cJTAG to JTAG converter” (CJTAG_2_JTAG.v) in the same layer as the mmRISC. The conversion logic is described in the SoC chapter of this document in detail.

File Name	Description	Note
defines_core.v	Definitions and Configurations for Core	
defines_chip.v	Definitions and Configurations for Chip	
mmRISC.v	Top Layer of mmRISC-1	
bus_m_ahb.v	Bridge from Internal Bus to AHB Lite	
csr_mtime.v	Memory Mapped CSR MTIME	
cpu_top.v	Top Layer of CPU (Hart)	
cpu_fetch.v	Instruction Fetch Unit	
cpu_pipeline.v	Pipeline and Decode Control	
cpu_datapath.v	Datapath Unit	
cpu_csr.v	Machine Mode CSR	
cpu_csr_int.v	Interrupt Controller and its CSR	
cpu_csr_dbg.v	Debug CSR	
cpu_debug.v	Debug Support Logic	
cpu_fpu32.v	Floating Point Unit	
debug_top.v	Debugger Top	
debug_dtm_jtag.v	Debug Transport Module with JTAG I/F	
debug_cdc.v	Clock Domain Crossing	
debug_dm.v	Debug Module	
cjtag_2_jtag.v	Converter from cJTAG to JTAG	optional

Table 1.3: RTL Files

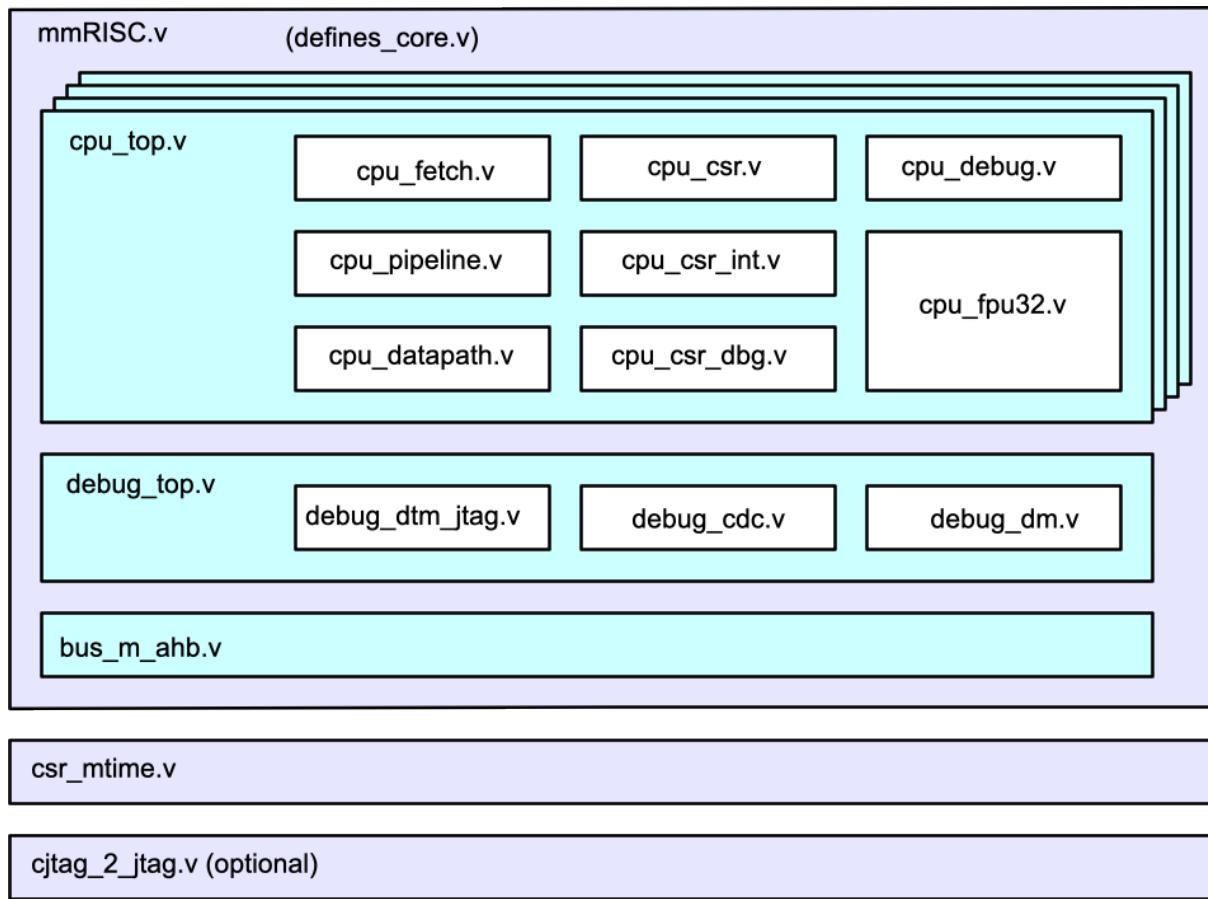


Figure 1.5: RTL Files

1.6 Input / Output Signals of mmRISC-1

Input / Output signals of mmRISC-1 (mmRISC.v) are shown in Table 1.4 to Table 1.12.

Group	Direction	Width	Name	Description	Note
System	input		RES_ORG	Origin of Reset e.g. Power-on-Reset or Reset from External	
System	output		RES_SYS	System Reset Output It is logical OR among RES_ORG, SRSTn and a reset from debugger.	
System	input		CLK	System Clock	
System	input		STBY	System Stand-by It is used to show harts are available or not. If STBY is asserted, each hart is treated as unavailable.	
System	input		STBY_REQ	CPU Stand-By Request It is used to flush pipeline and halt execution before entering system in stand-by low power mode.	
System	output		STBY_ACK	CPU Stand-By Acknowledge It is used to inform the CPU has halted completely, then the system can stop the system clock.	

Table 1.4: Input / Output Signals of mmRISC-1 (System)

Group	Direction	Width	Name	Description	Note
System	input		SRSTn_IN	System Reset Input except for Debug Block	
System	output		SRSTn_OUT	System Reset Output from Debug Blocks	
JTAG	input		TRSTn	JTAG Tap Reset	
JTAG	input		TCK	JTAG Clock	
JTAG	input		TMS	JTAG Mode Select	
JTAG	input		TDI	JTAG Data Input	
JTAG	output		TDO_E	JTAG Data Output Enable (for 3-state output)	
JTAG	output		TDO_D	JTAG Data Output Data (for 3-state output)	
JTAG	output		RTCK	JTAG Return Clock	
JTAG	input	[31:0]	JTAG_DR_USER_IN	JTAG DR User Register Input (to capture)	
JTAG	output	[31:0]	JTAG_DR_USER_OUT	JTAG DR User Register Output (updated)	
Debug	input		FORCE_HALT_ON_RESET_REQ	Request to force CPU in Halt State when Reset	
Debug	output		FORCE_HALT_ON_RESET_ACK	Acknowledge of Above Request	

Table 1.5: Input / Output Signals of mmRISC-1 (JTAG)

Group	Direction	Width	Name	Description	Note
Config	input	[31:0]	RESET_VECTOR[0:HART_COUNT-1]	Reset Vector Configuration for each Hart	
Config	input		DEBUG_SECURE	Debug is secured or not	
Config	input	[31:0]	DEBUG_SECURE_CODE_0	Debug Secure Code 0 to be matched with CSR AUTHDATA	
Config	input	[31:0]	DEBUG_SECURE_CODE_1	Debug Secure Code 1 to be matched with CSR AUTHDATA	

Table 1.6: Input / Output Signals of mmRISC-1 (Configuration)

Group	Direction	Width	Name	Description	Note
Instruction Bus	output		CPUI_M_SEL [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Select	
Instruction Bus	output	[1:0]	CPUI_M_HTRANS [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Transfer	
Instruction Bus	output		CPUI_M_HWRITE [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Write	
Instruction Bus	output		CPUI_M_HMASTERLOCK [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Master Lock	
Instruction Bus	output	[2:0]	CPUI_M_HSIZE [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Size	
Instruction Bus	output	[2:0]	CPUI_M_HBURST [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Burst	
Instruction Bus	output	[3:0]	CPUI_M_HPROT [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Protection	
Instruction Bus	output	[31:0]	CPUI_M_HADDR [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Address	
Instruction Bus	output	[31:0]	CPUI_M_HWDATA [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Write Data	
Instruction Bus	output		CPUI_M_HREADY [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Ready Output	
Instruction Bus	input		CPUI_M_HREADYOUT [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Ready Input	
Instruction Bus	input	[31:0]	CPUI_M_RDATA [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Read Data	
Instruction Bus	input		CPUI_M_HRESP [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Response	

Table 1.7: Input / Output Signals of mmRISC-1 (Instruction Bus)

Group	Direction	Width	Name	Description	Note
Data Bus	output		CPUD_M_SEL [0:`HART_COUNT-1]	Data Bus; AHB-Lite Select	
Data Bus	output	[1:0]	CPUD_M_HTRANS [0:`HART_COUNT-1]	Data Bus; AHB-Lite Transfer	
Data Bus	output		CPUD_M_HWRITE [0:`HART_COUNT-1]	Data Bus; AHB-Lite Write	
Data Bus	output		CPUD_M_HMASTERLOCK [0:`HART_COUNT-1]	Data Bus; AHB-Lite Master Lock	
Data Bus	output	[2:0]	CPUD_M_HSIZE [0:`HART_COUNT-1]	Data Bus; AHB-Lite Size	
Data Bus	output	[2:0]	CPUD_M_HBURST [0:`HART_COUNT-1]	Data Bus; AHB-Lite Burst	
Data Bus	output	[3:0]	CPUD_M_HPROT [0:`HART_COUNT-1]	Data Bus; AHB-Lite Protection	
Data Bus	output	[31:0]	CPUD_M_HADDR [0:`HART_COUNT-1]	Data Bus; AHB-Lite Address	
Data Bus	output	[31:0]	CPUD_M_HWDATA [0:`HART_COUNT-1]	Data Bus; AHB-Lite Write Data	
Data Bus	output		CPUD_M_HREADY [0:`HART_COUNT-1]	Data Bus; AHB-Lite Ready Output	
Data Bus	input		CPUD_M_HREADYOUT [0:`HART_COUNT-1]	Data Bus; AHB-Lite Ready Input	
Data Bus	input	[31:0]	CPUD_M_RDATA [0:`HART_COUNT-1]	Data Bus; AHB-Lite Read Data	
Data Bus	input		CPUD_M_HRESP [0:`HART_COUNT-1]	Data Bus; AHB-Lite Response	

Table 1.8: Input / Output Signals of mmRISC-1 (Data Bus)

Group	Direction	Width	Name	Description	Note
Atomic	input		CPUM_S_SEL [0:`HART_COUNT-1]	Atomic Monitor Bus; AHB-Lite Select	
Atomic	input	[1:0]	CPUM_S_HTRANS [0:`HART_COUNT-1]	Atomic Monitor Bus; AHB-Lite Transfer	
Atomic	input		CPUM_S_HWRITE [0:`HART_COUNT-1]	Atomic Monitor Bus; AHB-Lite Write	
Atomic	input	[31:0]	CPUM_S_HADDR [0:`HART_COUNT-1]	Atomic Monitor Bus; AHB-Lite Address	
Atomic	input		CPUM_S_HREADY [0:`HART_COUNT-1]	Atomic Monitor Bus; AHB-Lite Ready Output	
Atomic	input		CPUM_S_HREADYOUT [0:`HART_COUNT-1]	Atomic Monitor Bus; AHB-Lite Ready Input	

**Table 1.9: Input / Output Signals of mmRISC-1
(Atomic LR/SC Monitor)¹**

Group	Direction	Width	Name	Description	Note
Sys Bus	output		DBGD_M_SEL	Debug System Bus; AHB-Lite Select	
Sys Bus	output	[1:0]	DBGD_M_HTRANS	Debug System Bus; AHB-Lite Transfer	
Sys Bus	output		DBGD_M_HWRITE	Debug System Bus; AHB-Lite Write	
Sys Bus	output		DBGD_M_HMASTERLOCK	Debug System Bus; AHB-Lite Master Lock	
Sys Bus	output	[2:0]	DBGD_M_HSIZE	Debug System Bus; AHB-Lite Size	
Sys Bus	output	[2:0]	DBGD_M_HBURST	Debug System Bus; AHB-Lite Burst	
Sys Bus	output	[3:0]	DBGD_M_HPROT	Debug System Bus; AHB-Lite Protection	
Sys Bus	output	[31:0]	DBGD_M_HADDR	Debug System Bus; AHB-Lite Address	
Sys Bus	output	[31:0]	DBGD_M_HWDATA	Debug System Bus; AHB-Lite Write Data	
Sys Bus	output		DBGD_M_HREADY	Debug System Bus; AHB-Lite Ready Output	
Sys Bus	input		DBGD_M_HREADYOUT	Debug System Bus; AHB-Lite Ready Input	
Sys Bus	input	[31:0]	DBGD_M_RDATA	Debug System Bus; AHB-Lite Read Data	
Sys Bus	input		DBGD_M_HRESP	Debug System Bus; AHB-Lite Response	

**Table 1.10: Input / Output Signals of mmRISC-1
(Debug System Access Bus)**

Group	Direction	Width	Name	Description	Note
Interrupt	input		IRQ_EXT	External Interrupt	
Interrupt	input		IRQ_MSOF	Machine Software Interrupt	
Interrupt	input		IRQ_MTIME	Machine Timer Interrupt	
Interrupt	input	[63:0]	IRQ	Interrupt Request (towards Interrupt Controller)	

Table 1.11: Input / Output Signals of mmRISC-1 (Interrupt)

A signal called DBG_STOP_TIMER in Table 1.12 can be used to stop the Tick count in the MTIME and to stop the counter of the Watch Dog Timer (WDT) in SoC during debug mode, that is during the halt state according to the break operation.

Group	Direction	Width	Name	Description	Note
MTIME	input	[31:0]	MTIME	Machine Timer Counter LSB	
MTIME	input	[31:0]	MTIMEH	Machine Timer Counter MSB	
MTIME	output		DBG_STOP_TIMER	Stop Timer due to entering Debug Mode	

Table 1.12: Input / Output Signals of mmRISC-1 (MTIME)

¹ Available when `RISCV_ISA_RV32A` is defined.

1.7 Clock & Reset

The mmRISC-1 has two clock domains; one is CLK (System Clock) and the other is TCK (JTAG Clock). These clocks are asynchronous, so clock domain crossing logic is used in DTM (Debug Transport Module).

The Reset Structure of mmRISC-1 is slightly complicated as shown in Figure 1.6. RES_ORG is the origin of chip reset such as power-on-reset or reset from external pin. RES_DBG, which is controlled by DTMCS, is a reset signal only for debug logic. DMCONTROL can control both RES_SYS and HART_RESET[]. RES_SYS is a system reset for the whole chip and peripherals. HART_RESET[] are connected to each CPU (Hart) and can independently reset each hart. In each CPU (Hart), one of the HART_RESET signals is renamed to RES_CPU, and the RES_CPU resets the whole area in the CPU block. In the chip top layer, SRSTn is a bi-directional inout signal. The SRSTn input is used to reset the whole chip except for the debug block, and the SRSTn output is generated from the debug block.

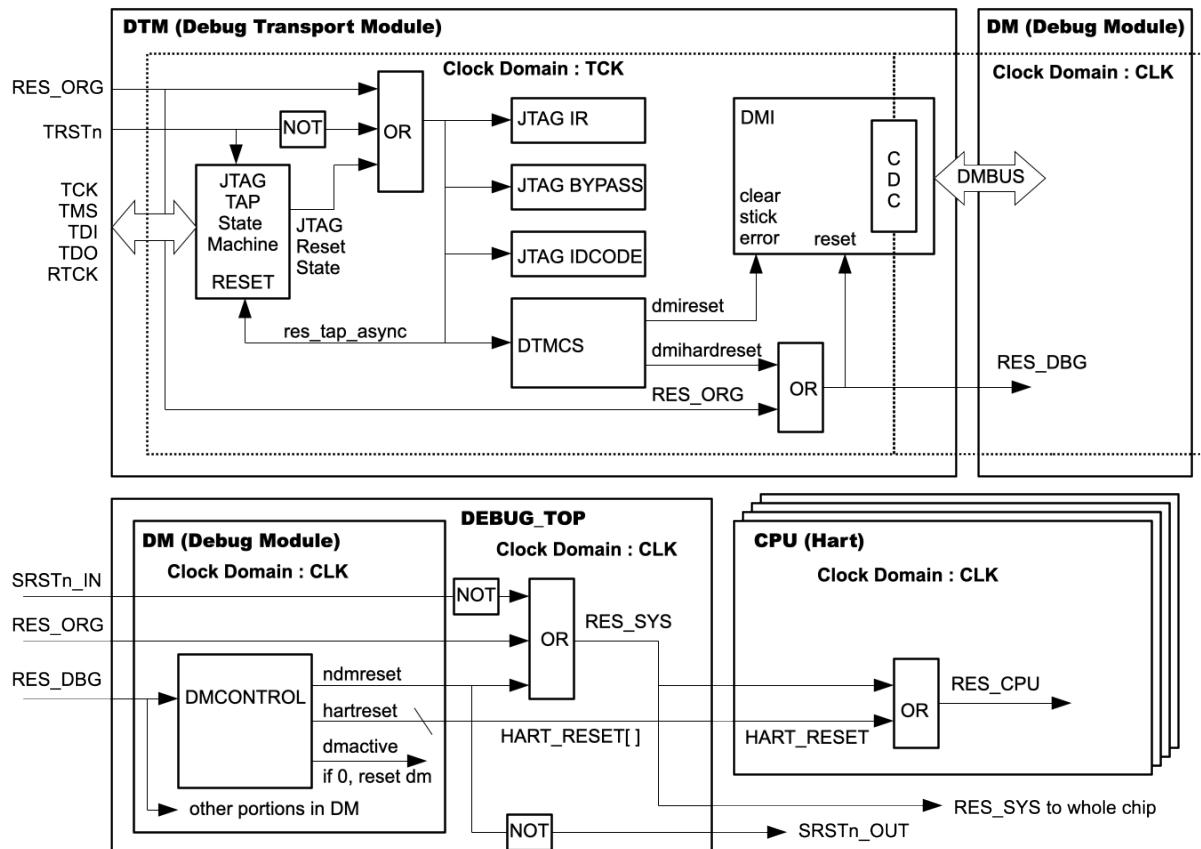


Figure 1.6: Reset Structure

1.8 Exceptions

Privileged mode of mmRISC-1 is only Machine-Mode (M-Mode). Exceptions are divided into two categories; one is synchronous exceptions which are invoked by instructions, and the other is asynchronous exceptions which are invoked by interrupt inputs.

Whenever an exception occurs, hardware will automatically save and restore important registers. The following steps are completed by hardware before jumping to the vector which corresponds to each exception.

1. Save PC to MEPC.
2. Save Privilege level to MSTATUS. (Fixed to 2 due to only M-Mode is supported.)
3. Save MSTATUS.mie to MSTATUS.mpie.
4. Disable interrupts by setting MSTATUS.mie = 0.
5. MVAL is set to specific data corresponding to each exception.
6. MCAUSE is set to cause information corresponding to each exception.
7. Set PC to corresponding exception's Vector Address associated to MTVEC (Direct Mode or Vectored Mode).

At this point, software receives its responsibility to handle proper exception process. After the process is finished, the MRET instruction should be executed. The MRET executes following steps.

1. Restore MSTATUS.mpp to privilege level. (Fixed to 2 due to only M-Mode is supported.)
2. Restore MSTATUS.mpie to MSTATUS.mie.
3. Restore MEPC to PC.

Supported exceptions and corresponding MCAUSE, Vector Address and MVAL are summarized in Table 1.13. If an instruction invokes multiple synchronous exceptions simultaneously, one of those is selected by priorities shown in Table 1.14.

Group	Cause and Vector					
	MCAUSE		Vector Offset		Description	MVAL
	Interrupt	Exception Code	Direct Mode	Vectored Mode		
Asynchronous	1	0x03	base	base + 0x000c	Machine Software Interrupt (priority is controlled by software)	0x00000000
	1	0x07	base	base + 0x001c	Machine Timer Interrupt (priority is controlled by software)	0x00000000
	1	0x0b	base	base + 0x002c	Machine External Interrupt (priority is controlled by software)	0x00000000
	1	0x10	base	base + 0x0040	Machine IRQ00 (priority is controlled by hardware)	0x00000000
	1	0x11	base	base + 0x0044	Machine IRQ01 (priority is controlled by hardware)	0x00000000
	1
	1	0x4f	base	base + 0x013c	Machine IRQ63 (priority is controlled by hardware)	0x00000000
Synchronous	0	0x00	base	base	Instruction Address Misaligned	Never occurs
	0	0x01	base	base	Instruction Access Fault	Fault Instruction Address
	0	0x02	base	base	Illegal Instruction	0x00000000
	0	0x03	base	base	Breakpoint (Trigger, EBREAK, CBREAK)	Next PC of executed
	0	0x04	base	base	Load Address Misaligned	Bus Error Address
	0	0x05	base	base	Load Access Fault	Bus Error Address
	0	0x06	base	base	Store/AMO(Atomic Memory Operation) Address Misaligned	Bus Error Address
	0	0x07	base	base	Store/AMO Access Fault	Bus Error Address
	0	0xb	base	base	Environment Call from M-mode	0x00000000

Table 1.13: Exceptions

Synchronous Exception Priority		
Priority	Exception Code	Description
Highest	0x03	Instruction Address Break Point
Higher	0x01	Instruction Access Fault
	0x02	Illegal Instruction
	0x00	Instruction Address Misaligned
	0x0b	Environment Call from M-mode
	0x03	Environment Break
Lower	0x03	Load/Store/AMO Address Break Point
	0x06	Store/AMO(Atomic Memory Operation) Address Misaligned
	0x04	Load Address Misaligned
	0x07	Store/AMO Access Fault
Lowest	0x05	Load Access Fault

Table 1.14: Priority of Synchronous Exception

1.9 Interrupts and INTC (Interrupt Controller)

The mmRISC-1 supports the following four interrupt request inputs.

IRQ_EXT External Interrupt

IRQ_MSOFT Machine Software Interrupt **IRQ_MTIME** Machine Timer Interrupt

IRQ[63:0] Interrupt Request (towards Interrupt Controller)

Priority among above interrupts is defined as $\text{IRQ_EXT} > \text{IRQ_SOFT} > \text{IRQ_MTIME} > \text{IRQ}[63:0]$. 64 IRQ inputs are controlled by INTC (Interrupt Controller) shown in Figure 1.7. The INTC can be configured by dedicated CSRs. INTCFGSENSE0/1 select each IRQ sense way from level sense or rising edge sense. INTPENDING0/1 show each IRQ input logic level for level sense, or pending status for rising edge sense. For rising edge IRQ, pending status can be cleared by writing 1 to each corresponding bit. INTCFGENABLE0/1 enable or disable each interrupt.

INTCFGPRIORITY0/1/2/3 configure priority level of each IRQ. The priority level is expressed in 4bits, 4' b0000 is lowest and 4' b1111 is highest. In priority tournament block, one IRQ having the highest priority level is selected from requesting IRQ, and finally if the priority is larger than MINTCURLVL, the selected IRQ is outputted. When CPU accepts the IRQ request, MINTCURLVL is automatically transferred to MINTPRELVL, and priority level of accepted IRQ is also automatically transferred to MINTCURLVL. In last phase of interrupt software handler before MRET, software should copy MINTPRELVL to MINTCURLVL. Thus IRQ allows nested interrupts.

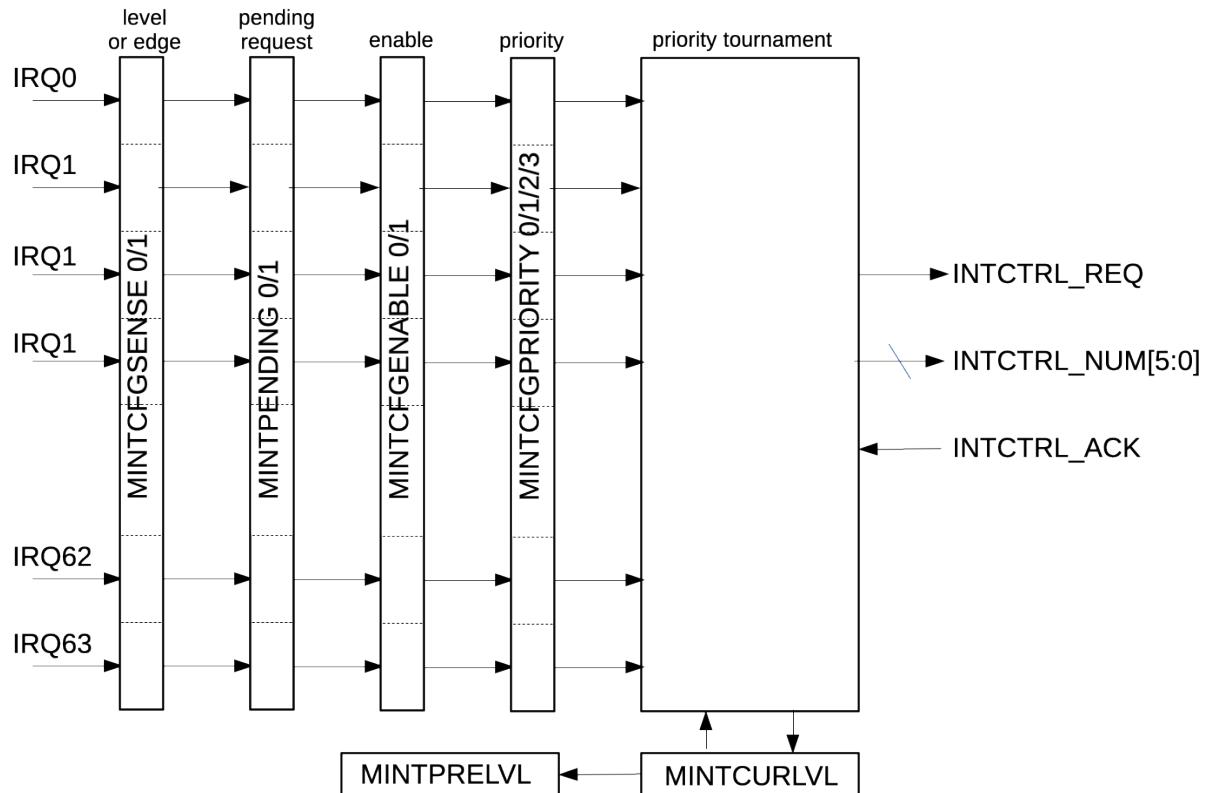


Figure 1.7: INTC (Interrupt Controller)

An example of a startup routine that includes exception handlers and interrupt entry points is shown in Listing 1.1. Also, an example of interrupt initialization routines and

interrupt handler routines written in C language, which are combined with the startup routine, are shown in Listing 1.2. The sample routines support nested interrupts.

The sample routine depends on the GNU C compiler to save and restore the XRn and FRn registers by using the keyword "`__attribute__((interrupt))`" to define the interrupt handler routines.

In this example, multiple interrupts with different priorities are gathered into one entry point and one handler routine "`INT_Handler_IRQ()`". This example increases the interrupt latency due to the conditional judgment in the handler routine.

If you want to improve the latency of each interrupt, it is recommended to prepare pairs of dedicated entry point and corresponding handler routine for each interrupt as shown in 1.3. In the case, the interrupt nesting is not supported.

```

.globl _start
.globl main
.globl INT_Handler_EXT
.globl INT_Handler_MSOFST
.globl INT_Handler_MTIME
.globl INT_Handler_IRQ
-----
// Reset Vector
-----
.text
.section ".vectors"
.align 4
_reset:
    j    _start
_vector_base:
    j    _trap_exception      // 00:Exception
    j    _trap_illegal        // 01:Supervisor Software Interrupt
    j    _trap_illegal        // 02:Reserved
    j    INT_Handler_MSOFST // 03:Machine Software Interrupt
    j    _trap_illegal        // 04:Reserved
    j    _trap_illegal        // 05:Reserved
    j    _trap_illegal        // 06:Reserved
    j    INT_Handler_MTIME  // 07:Machine Timer Interrupt
    j    _trap_illegal        // 08:Reserved
    j    _trap_illegal        // 09:Reserved
    j    _trap_illegal        // 10:Reserved
    j    INT_Handler_EXT    // 11:Machine External Interrupt
    j    _trap_illegal        // 12:Reserved
    j    _trap_illegal        // 13:Reserved
    j    _trap_illegal        // 14:Reserved
    j    _trap_illegal        // 15:Reserved
    j    INT_Handler_IRQ    // 16:IRQ00
    j    INT_Handler_IRQ    // 17:IRQ01
    j    INT_Handler_IRQ    // 18:IRQ02
    j    INT_Handler_IRQ    // 19:IRQ03
    ...
    j    INT_Handler_IRQ    // 76:IRQ60
    j    INT_Handler_IRQ    // 77:IRQ61
    j    INT_Handler_IRQ    // 78:IRQ62
    j    INT_Handler_IRQ    // 79:IRQ63
-----
// Trap Entry
-----
.text
.section ".trap"
.align 4
//
_trap_exception:
    // Do Nothing
    mret
//
_trap_illegal:
    // Forever Loop as a Trap
    j    .
-----
// Startup
-----

```

```

.text
.section ".startup"
_start:
//
// Init GP and SP
la gp, __GLOBAL_PTR__
la sp, __STACK_TOP__
//
// Copy Initial Data
la a0, __ROM_INIT_BGN__
la a1, __RAM_INIT_BGN__
la a2, __RAM_INIT_END__
bgeu a1, a2, 2f
1:
lw t0, (a0)
sw t0, (a1)
addi a0, a0, 4
addi a1, a1, 4
bltu a1, a2, 1b
2:
//
// Clear BSS
la a0, __BSS_BGN__
la a1, __BSS_END__
bgeu a0, a1, 2f
1:
sw zero, (a0)
addi a0, a0, 4
bltu a0, a1, 1b
2:
//
// Setup MTVEC
la t0, _vector_base
ori t0, t0, 0x01 // vectored
csrw mtvec, t0
//
// Goto Main
call main
//
// Forever Loop
3:
j 3b

```

Listing 1.1: An Example of Startup Routine including Exception Handlers and Interrupt Entry Points written in Assembler language

```

#include <stdint.h>
#include "common.h"
#include "csr.h"
#include "gpio.h"
#include "system.h"
#include "uart.h"
#include "interrupt.h"

//-----
// Interrupt Initialization
//-----
void INT_Init(void)
{
    uint32_t irq;
    uint32_t level;
    //
    // Configure IRQ00 as UART RXD Interrupt
    IRQ_Config(0, 1, 1, 1);
    //
    // Enable MTIME to make periodic interrupt
    MTIME_Init(1, 10, 1, 0, 166666); // 100ms
    //
    // Enable MTIME and IRQ Interrupts
    INT_Config(0, 1, 0, 1, 0x0);
}

//-----
// IRQ Configuration
//-----

```

```

void IRQ_Config
(
    uint32_t irqnum,      // IRQ Number : 0 ~ 63
    uint32_t enable,      // IRQ Enable : 0=Disable, 1=Enable
    uint32_t sense,       // IRQ Sense : 0=Level , 1=Edge
    uint32_t level        // IRQ Level : 0 ~ 15
)
{
    uint32_t pos1, pos4;
    uint32_t data;
    //
    enable = enable & 0x01;
    sense = sense & 0x01;
    level = level & 0x0f;
    //
    // Set Priority
    if (irqnum < 8)
    {
        pos4 = (irqnum - 0) * 4;
        //
        data = read_csr(MINTCFG_PRIORITY0);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFG_PRIORITY0, data);
    }
    else if (irqnum < 16)
    {
        pos4 = (irqnum - 8) * 4;
        //
        data = read_csr(MINTCFG_PRIORITY1);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFG_PRIORITY1, data);
    }
    else if (irqnum < 24)
    {
        pos4 = (irqnum - 16) * 4;
        //
        data = read_csr(MINTCFG_PRIORITY2);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFG_PRIORITY2, data);
    }
    else if (irqnum < 32)
    {
        pos4 = (irqnum - 24) * 4;
        //
        data = read_csr(MINTCFG_PRIORITY3);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFG_PRIORITY3, data);
    }
    else if (irqnum < 40)
    {
        pos4 = (irqnum - 32) * 4;
        //
        data = read_csr(MINTCFG_PRIORITY4);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFG_PRIORITY4, data);
    }
    else if (irqnum < 48)
    {
        pos4 = (irqnum - 40) * 4;
        //
        data = read_csr(MINTCFG_PRIORITY5);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFG_PRIORITY5, data);
    }
    else if (irqnum < 56)
    {
        pos4 = (irqnum - 48) * 4;
        //
        data = read_csr(MINTCFG_PRIORITY6);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFG_PRIORITY6, data);
    }
    else if (irqnum < 64)
    {
        pos4 = (irqnum - 56) * 4;
        //
    }
}

```

```

        data = read_csr(MINTCFGPRIOITY7);
        data = (data & ~(0x0f << pos4)) | (level << pos4);
        write_csr(MINTCFGPRIOITY7, data);
    }
    //
    // Set Sense and Enable
    if (irqnum < 32)
    {
        pos1 = (irqnum - 0);
        //
        data = read_csr(MINTCFGSENSE0);
        data = (data & ~(0x01 << pos1)) | (sense << pos1);
        write_csr(MINTCFGSENSE0, data);
        //
        data = read_csr(MINTCFGENABLE0);
        data = (data & ~(0x01 << pos1)) | (enable << pos1);
        write_csr(MINTCFGENABLE0, data);
    }
    else if (irqnum < 64)
    {
        pos1 = (irqnum - 32);
        //
        data = read_csr(MINTCFGSENSE1);
        data = (data & ~(0x01 << pos1)) | (sense << pos1);
        write_csr(MINTCFGSENSE1, data);
        //
        data = read_csr(MINTCFGENABLE1);
        data = (data & ~(0x01 << pos1)) | (enable << pos1);
        write_csr(MINTCFGENABLE1, data);
    }
}

//-----
// Interrupt Configuration
//-----
void INT_Config
(
    uint32_t ena_intext,    // Enable External Interrupt
    uint32_t ena_intmtime, // Enable MTIME Interrupt
    uint32_t ena_intmsoft, // Enable SOFTWARE Interrupt
    uint32_t ena_irq,      // Enable IRQ Interrupts
    uint32_t cur_irqlvl   // Current IRQ Level
)
{
    uint32_t data;
    //
    ena_inttext = ena_inttext & 0x01;
    ena_intmtime = ena_intmtime & 0x01;
    ena_intmsoft = ena_intmsoft & 0x01;
    ena_irq = ena_irq & 0x01;
    cur_irqlvl = cur_irqlvl & 0x0f;
    //
    // Interrupt Current Level
    write_csr(MINTCURLVL, cur_irqlvl);
    //
    // Enable Interrupt
    data = read_csr(MIE);
    data = (data & ~((1<<11) | (1<<7) | (1<<3)))
           | (ena_inttext << 11) | (ena_intmtime << 7) | (ena_intmsoft << 3);
    write_csr(MIE, data);
    //
    data = read_csr(MSTATUS);
    data = (data & ~(0x01 << 3))
           | ((ena_inttext | ena_intmtime | ena_intmsoft | ena_irq) << 3);
    write_csr(MSTATUS, data);
}

//-----
// Interrupt Generate
//-----
void INT_Generate
(
    uint32_t inttext,
    uint32_t intsoft,
    uint64_t irq
)

```

```

{
    mem_wr32(INTGEN_IRQ_EXT, inttext & 0x01);
    mem_wr32(MSOFTIRQ, intsoft & 0x01);
    mem_wr32(INTGEN IRQ1, (uint32_t)(irq >> 32));
    mem_wr32(INTGEN IRQ0, (uint32_t)(irq & 0xffffffffUL));
}

//-----
// MTIME Initialization
//-----
void MTIME_Init
(
    uint32_t enable,
    uint32_t div_plus_one,
    uint64_t intena,
    uint64_t mtime_count,
    uint64_t mtime_compa
)
{
    uint32_t div;
    //
    enable = enable & 0x01;
    div = (div_plus_one > 0)? div_plus_one - 1 : div_plus_one;
    div = div & 0x3f;
    intena = intena & 0x01;
    //
    mem_wr32(MTIME_DIV, div);
    mem_wr32(MTIME, (uint32_t)(mtime_count & 0xffffffffUL));
    mem_wr32(MTIMEH, (uint32_t)(mtime_count >> 32));
    mem_wr32(MTIMECMP, (uint32_t)(mtime_compa & 0xffffffffUL));
    mem_wr32(MTIMECMPPH, (uint32_t)(mtime_compa >> 32));
    mem_wr32(MTIME_CTRL, (intena << 2) | (enable << 0));
}

//-----
// Interrupt Handler External
//-----
__attribute__((interrupt)) void INT_Handler_EXT(void)
{
    // Negate IRQ_EXT
    mem_wr32(INTGEN_IRQ_EXT, 0x00000000);
    //
    // Write your own code
    ...
}

//-----
// Interrupt Handler MTIME
//-----
__attribute__((interrupt)) void INT_Handler_MTIME(void)
{
    // Clear Interrupt Flag in MTIME
    mem_wr32(MTIME_CTRL, mem_rd32(MTIME_CTRL));
    //
    // Write your own code
    ...
}

//-----
// Interrupt Handler MSOFT
//-----
__attribute__((interrupt)) void INT_Handler_MSOFST(void)
{
    // Negate MSOFT
    mem_wr32(MSOFTIRQ, 0x00000000);
    //
    // Write your own code
    ...
}

//-----
// Interrupt Handler IRQ
//-----
__attribute__((interrupt)) void INT_Handler_IRQ(void)
{
    uint32_t irq_level;
}

```

```

uint64_t irq_pend0;
uint64_t irq_pend1;
uint64_t irq_pend;
uint32_t prev_mepc;
uint32_t prev_mstatus;
uint32_t prev_mintprelvl;
//
// Get Pending Status
irq_level = read_csr(MINTCURLVL);
irq_pend0 = (uint64_t)read_csr(MINTPENDING0);
irq_pend1 = (uint64_t)read_csr(MINTPENDING1);
irq_pend = (irq_pend1 << 32) + (irq_pend0 << 0);
//
// Save Previous Status
prev_mintprelvl = read_csr(MINTPRELVL);
prev_mepc = read_csr(MEPC);
prev_mstatus = read_csr(MSTATUS);
//
// Enable global interrupt (set mie)
write_csr(MSTATUS, prev_mstatus | 0x08);
//
//-----
// Dispatch to each Priority Handler
switch(irq_level)
{
    // Group Priority Level 1
    case 1 :
    {
        // Write your own code
        // IRQ00
        if (irq_pend0 & 0x00000001)
        {
            INT_UART_Handler();
            write_csr(MINTPENDING0, 0x00000001); // Clear Pending Edge
        }
        break;
    }
    // Group Priority 2-15
    case 2 :
    case 3 :
    case 4 :
    case 5 :
    case 6 :
    case 7 :
    case 8 :
    case 9 :
    case 10 :
    case 11 :
    case 12 :
    case 13 :
    case 14 :
    case 15 :
    {
        // Write your own code
        break;
    }
    // Never reach here
    default :
    {
        break;
    }
}
//-----
//
// Disable global interrupt, Save Previous Status
write_csr(MSTATUS, prev_mstatus); // mie=0
write_csr(MEPC, prev_mepc);
write_csr(MINTCURLVL, prev_mintprelvl);
}

```

Listing 1.2: An Example of Interrupt Initializations and Interrupt Handlers written in C language

```

===== Assembler Part =====
...

```

```
_vector_base:  
...  
    j      INT_Handler_IRQ00      // 16:IRQ00  
    j      INT_Handler_IRQ01      // 17:IRQ01  
    j      INT_Handler_IRQ02      // 18:IRQ02  
    j      INT_Handler_IRQ03      // 19:IRQ03  
...  
  
===== C language Part =====  
...  
__attribute__((interrupt)) void INT_Handler_IRQ00(void)  
{  
    // Write your own code  
}  
__attribute__((interrupt)) void INT_Handler_IRQ01(void)  
{  
    // Write your own code  
}  
__attribute__((interrupt)) void INT_Handler_IRQ02(void)  
{  
    // Write your own code  
}  
__attribute__((interrupt)) void INT_Handler_IRQ03(void)  
{  
    // Write your own code  
}  
...
```

Listing 1.3: An Example of Interrupt related routines (assembler and C) involving pairs of dedicated entry point and corresponding handler routine in order to improve the interrupt latency

1.10 Memory Model

Memory Model of mmRISC-1 is based on the Little Endian order as shown in Figure 1.8. As for instruction, both 16bit width code and 32bit width code should be aligned to 2-byte boundary. 32bit width code does not require its alignment to 4-byte boundary even if RV32C is not enabled.

As for data, 16bit data should be aligned to 2-byte boundary, and 32bit data should be aligned to 4- byte boundary. Unaligned data access invokes a memory alignment exception.

The mmRISC-1 uses strong memory access ordering. The sequence and the number of memory accesses are matched to the corresponding sequence of instructions executed. FENCE instruction is executed as NOP, FENCE.I instruction flushes the instruction fetch queue.

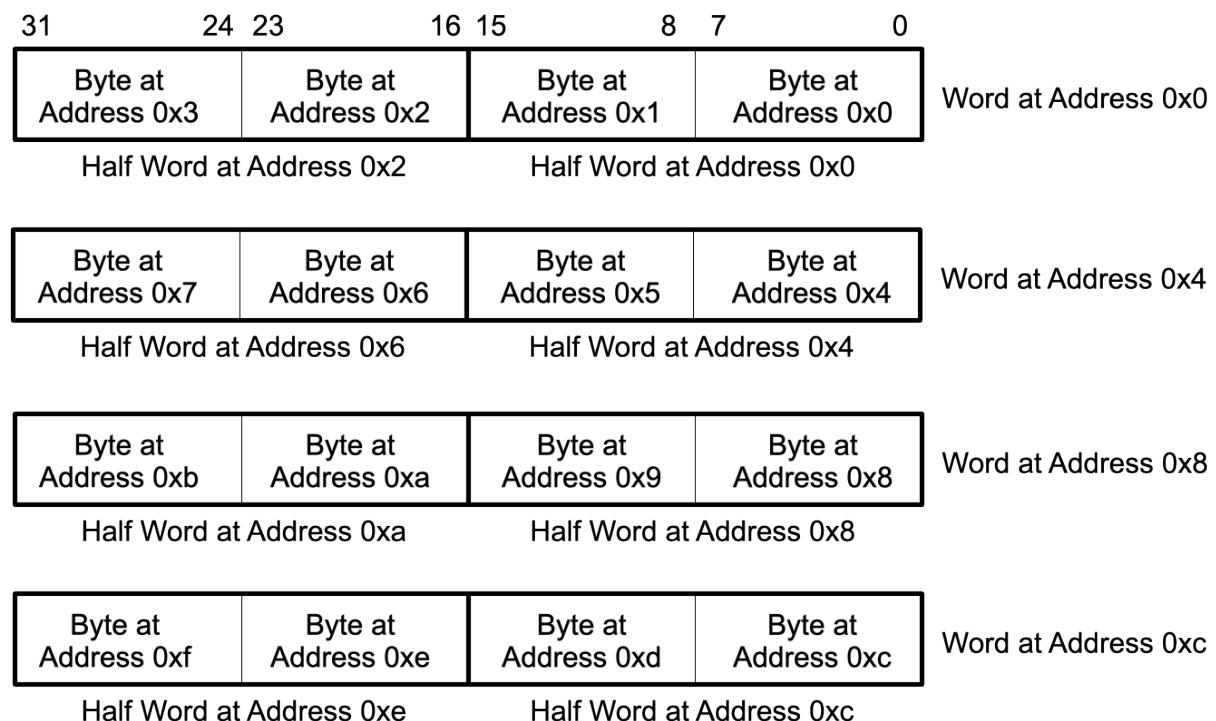


Figure 1.8: Little Endian Memory Organization

1.11 CSR Map List

Machine Mode CSR		
Address	Name	Description
0xf11	MVENDORID	Machine Vendor ID
0xf12	MARCHID	Machine Architecture ID
0xf13	MIMPID	Machine Implementation ID
0x300	MSTATUS	Machine Status
0x301	MISA	Machine Instruction Set Architecture
0x304	MIE	Machine Interrupt Enable
0x344	MIP	Machine Interrupt Pending
0x305	MTVEC	Machine Trap Vector Base Address
0x340	MSCRATCH	Machine Scratch
0x341	MEPC	Machine Exception Program Counter
0x342	MCAUSE	Machine Cause
0x343	MTVAL	Machine Trap Value
0xb00	MCYCLE	Machine Cycle Counter Lower Side
0xb80	MCYCLES	Machine Cycle Counter Upper Side
0xb02	MINSTRET	Machine Instruction Retired Counter Lower Side
0xb82	MINSTRETH	Machine Instruction Retired Counter Upper Side
0x320	MCOUNTINHIBIT	Machine Counter Inhibit
0xc00	CYCLE	Read-Only Mirror of MCYCLE
0xc01	TIME	Read-Only Mirror of MTIME (Memory Mapped MTIME)
0xc02	INSTRET	Read-Only Mirror of MINSTRET
0xc80	CYCLES	Read-Only Mirror of MCYCLES
0xc81	TIMEH	Read-Only Mirror of MTIMEH (Memory Mapped MTIMEH)
0xc82	INSTRETH	Read-Only Mirror of MINSTRETH

Table 1.15: Machine Mode CSR

Interrupt Controller CSR		
Address	Name	Description
0xbf0	MINTCURLVL	Machine Interrupt Current Level
0xbf1	MINTPRELVL	Machine Interrupt Previous Level
0xbf2	MINTCFGENABLE0	Machine Interrupt Configuration Enable 0 (x = 0)
0xbf3	MINTCFGENABLE1	Machine Interrupt Configuration Enable 1 (x = 1)
0xbf4	MINTCFGSENSE0	Machine Interrupt Configuration Sense 0 (x = 0)
0xbf5	MINTCFGSENSE1	Machine Interrupt Configuration Sense 1 (x = 1)
0xbf6	MINTPENDING0	Machine Interrupt Pending 0 (x = 0)
0xbf7	MINTPENDING1	Machine Interrupt Pending 1 (x = 1)
0xbf8	MINTCFGPRIORITY0	Machine Interrupt Configuration Priority 0 (x = 0)
0xbf9	MINTCFGPRIORITY1	Machine Interrupt Configuration Priority 1 (x = 1)
0xbfa	MINTCFGPRIORITY2	Machine Interrupt Configuration Priority 2 (x = 2)
0xbfb	MINTCFGPRIORITY3	Machine Interrupt Configuration Priority 3 (x = 3)
0xbfc	MINTCFGPRIORITY4	Machine Interrupt Configuration Priority 4 (x = 4)
0xbfd	MINTCFGPRIORITY5	Machine Interrupt Configuration Priority 5 (x = 5)
0xbfe	MINTCFGPRIORITY6	Machine Interrupt Configuration Priority 6 (x = 6)
0xbff	MINTCFGPRIORITY7	Machine Interrupt Configuration Priority 7 (x = 7)

Table 1.16: INTC (Interrupt Controller) CSR

FPU CSR		
Address	Name	Description
0x003	FCSR	Floating Point Control and Status Register
0x002	FRM	Floating Point Dynamic Rounding Mode
0x001	FFLAGS	Floating Point Acquired Exception Flags
0xbe0	FCONV	Floating Point Convergence Loop Count for FDIV/FSQRT Goldschmidt's Algorithm

Table 1.17: FPU CSR

Debug CSR		
Address	Name	Description
0x7b0	DCSR	Debug Control and Status
0x7b1	DPC	Debug PC

Table 1.18: Debugger CSR

Debug Trigger Module CSR		
Address	Name	Description
0x7a0	TSELECT	Trigger Select
0x7a1	TDATA1	Trigger Data 1
0x7a1	MCONTROL (TDATA1; Type 2)	Match Control (Trigger Data of Type 2)
0x7a1	ICOUNT (TDATA1; Type 3)	Instruction Count (Trigger Data of Type 3)
0x7a2	TDATA2	Trigger Specific Data
0x7a4	TINFO	Trigger Info

Table 1.19: Debug Trigger Module CSR

1.12 Other Register Map List except for CSR

Memory Mapped Peripheral Base Address = 0x49000000		
MTIME		
Offset Addr	Name	Description
0x0000	MTIME_CTRL	MTIME Control Register
0x0004	MTIME_DIV	MTIME Clock Divider
0x0008	MTIME	MTIME Timer Ticks (lower 32bits)
0x000c	MTIMEH	MTIME Timer Ticks (higher 32bits)
0x0010	MTIMECMP	MTIME Timer Ticks Comparator (lower 32bits)
0x0014	MTIMECMPPH	MTIME Timer Ticks Comparator (higher 32bits)
0x0018	MSOFTIRQ	Software Interrupt Request

Table 1.20: Machine Mode MTIME Registers (Memory Mapped Registers; So far, recommended Base Address is 0x49000000)

DTM (Debug Transport Module) JTAG Tap Registers		
Address	Name	Description
0x00	BYPASS	JTAG recommends this encoding
0x01	IDCODE	JTAG recommends this encoding
0x10	DTMCS	For Debugging
0x11	DMI	For Debugging
0x12	Reserved BYPASS	For Debugging
0x13	Reserved BYPASS	For Debugging
0x14	Reserved BYPASS	For Debugging
0x15	Reserved BYPASS	For Debugging
0x16	Reserved BYPASS	For Debugging
0x17	Reserved BYPASS	For Debugging
0x1f	BYPASS	JTAG Requires this encoding

Table 1.21: DTM (Debug Transport Module) JTAG Tap Registers; Address is 5bit IR

DM (Debug Module) Registers		
Address	Name	Description
0x11	DMSTATUS	Debug Module Status
0x10	DMCONTROL	Debug Module Control
0x12	HARTINFO	Hart Info
0x14	HAWINDOWSEL	Hart Array Window Select
0x15	HAWINDOW	Hart Array Window
0x30	AUTHDATA	Authentication Data
0x40	HALTSMU0	Halt Summary 0
0x13	HALTSMU1	Halt Summary 1
0x34	HALTSMU2	Halt Summary 2
0x35	HALTSMU3	Halt Summary 3

Table 1.22: DM (Debug Module) Registers; Address comes from DMI bus

DM (Debug Module) Registers : Abstract Command		
Address	Name	Description
0x04	DATA0	Abstract Data 0 ; arg0 / return value
0x05	DATA1	Abstract Data 1 ; arg1
0x17	COMMAND	Abstract Command : Access Register
0x17	COMMAND	Abstract Command : Access Memory
0x16	ABSTRACTCS	Abstract Control and Status

Table 1.23: DM (Debug Module) Abstract Command Registers; Address comes from DMI bus

DM (Debug Module) Registers : System Bus Access		
Address	Name	Description
0x38	SBCS	System Bus Access Control and Status
0x39	SBADDRESS0	System Bus Address 31:0
0x3c	SBDATA0	System Bus Data 31:0

Table 1.24: DM (Debug Module) System Bus Access Registers; Address comes from DMI bus

1.13 Machine Mode CSR

Address	Name	Description			
0xf11	MVENDORID	Machine Vendor ID			
Bit	Field	Initial	Access	Description	Note
31:0	vendorid	32'h0	R	JEDEC manufacturer ID; bit [31:7]: bank, bit [6:0]: offset. Configurable by `MVENDORID`.	

Table 1.25: MVENDORID

Address	Name	Description			
0xf12	MARCHID	Machine Architecture ID			
Bit	Field	Initial	Access	Description	Note
31:0	archid	32'h6d6d 3031	R	Base microarchitecture of the hart. Configurable by `MARCHID`	"mm01" in ASCII code

Table 1.26: MARCHID

Address	Name	Description			
0xf13	MIMPID	Machine Implementation ID			
Bit	Field	Initial	Access	Description	Note
31:0	impid	32'h10	R	Version of the processor implementation Configurable by `MIMPID`	

Table 1.27: MIMPID

Address	Name	Description			
0x300	MSTATUS	Machine Status			
Bit	Field	Initial	Access	Description	Note
31:15	0	17'h0	R	Always 0.	
14:13	fs	*	R/W	Status of Floating Point Unit. 0: Off (No FPU.) 1: Initial (FPU is in initial state just after reset.) 2: Clean (Privileged code should set to clean after saving context of f0-f31 and fcsr.) 3: Dirty (When an instruction writing to f0-f31 or fcsr is executed, hardware sets it to dirty.) *: 2'b00(off) if without Floating Point ISA, 2'b01(initial) if with Floating Point ISA.	
12:11	mpp	2'b10	R	Previous Privileged Mode, fixed to 2 (Machine Mode).	
10:8	0	3'b000	R	Always 0.	
7	mpie	1'b0	R/W	Previous Global Interrupt Enable. When Interrupt on Exception happens, mie is copied to mpie, and mie is cleared.	
6:4	0	3'b000	R	Always 0.	
3	mie	1'b0	R/W	Global Interrupt Enable. 0: Interrupt Disable 1: Interrupt Enable When Interrupt on Exception happens, mie is copied to mpie, and mie is cleared. When MRET instruction executed, mpie is restored to mie.	
2:0	0	3'b000	R	Always 0.	

Table 1.28: MSTATUS

Address	Name	Description			
0x301	MISA	Machine Instruction Set Architecture			
Bit	Field	Initial	Access	Description	Note
31:30	mxl	2'b01	R	XLEN = 32	
29:26	0	4'b0000	R	Always 0.	
25	z	1'b0	R	Reserved	
24	y	1'b0	R	Reserved	
23	x	1'b0	R	Non-standard extensions present	
22	w	1'b0	R	Reserved	
21	v	1'b0	R	Vector extension	
20	u	1'b0	R	User mode implemented	
19	t	1'b0	R	Transactional Memory extension	
18	s	1'b0	R	Supervisor mode implemented	
17	r	1'b0	R	Reserved	
16	q	1'b0	R	Quad-precision floating-point extension	
15	p	1'b0	R	Packed-SIMD extension	
14	o	1'b0	R	Reserved	
13	n	1'b1	R	User-level interrupts supported	
12	m	1'b1	R	Integer Multiply/Divide extension	
11	l	1'b0	R	Decimal Floating-Point extension	
10	k	1'b0	R	Reserved	
9	j	1'b0	R	Dynamically Translated Languages extension	
8	i	1'b1	R	RV32I/64I/128I base ISA	
7	h	1'b0	R	Hypervisor extension	
6	g	1'b0	R	Additional standard extensions present	
5	f	Preset	R	Single-precision floating-point extension Set 1 if 'RISCV_ISA_RV32F specified.	
4	e	1'b0	R	RV32E base ISA	
3	d	1'b0	R	Double-precision floating-point extension	
2	c	1'b1	R	Compressed extension	
1	b	1'b0	R	Bit-Manipulation extension	
0	a	Preset	R	Atomic extension Set 1 if 'RISCV_ISA_RV32Aspecified.	

Table 1.29: MISA

Address	Name	Description			
0x304	MIE	Machine Interrupt Enable			
Bit	Field	Initial	Access	Description	Note
31:12	0	20'h0	R	Always 0.	
11	meie	1'b0	R/W	Machine External Interrupt Enable	
10:8	0	3'b000	R	Always 0.	
7	mtie	1'b0	R/W	Machine Timer Interrupt Enable	
6:4	0	3'b000	R	Always 0.	
3	msie	1'b0	R/W	Machine Software Interrupt Enable	
2:0	0	3'b000	R	Always 0.	

Table 1.30: MIE

Address	Name	Description			
0x344	MIP	Machine Interrupt Pending			
Bit	Field	Initial	Access	Description	Note
31:12	0	20'h0	R	Always 0.	
11	meip	1'b0	R	Machine External Interrupt Pending	
10:8	0	3'b000	R	Always 0.	
7	mtip	1'b0	R	Machine Timer Interrupt Pending	
6:4	0	3'b000	R	Always 0.	
3	msip	1'b0	R	Machine Software Interrupt Pending	
2:0	0	3'b000	R	Always 0.	

Table 1.31: MIP

Address	Name	Description			
0x305	MTVEC	Machine Trap Vector Base Address			
Bit	Field	Initial	Access	Description	Note
31:2	base	30'h0	R/W	Vector Base Address (upper 30bits)	
1:0	mode	2'b00	R/W	Vector Mode 0: Direct (All exception set pc to base.) 1: Vectored (Asynchronous interrupts set pc to (base + 4 x cause).)	

Table 1.32: MTVEC

Address	Name	Description			
0x340	MSCRATCH	Machine Scratch			
Bit	Field	Initial	Access	Description	Note
31:0	mscratch	32'h0	R/W	Any Scratch Read & Write Data for Machine Mode.	

Table 1.33: MSCRATCH

Address	Name	Description			
0x341	MEPC	Machine Exception Program Counter			
Bit	Field	Initial	Access	Description	Note
31:0	mepc	32'h0	R/W	When a trap is taken into M-mode, mepc is written with the virtual address of the instruction that was interrupted or that encountered the exception. Otherwise, mepc is never written by the implementation, though it may be explicitly written by software.	

Table 1.34: MEPC

Address	Name	Description			
0x342	MCAUSE	Machine Cause			
Bit	Field	Initial	Access	Description	Note
31	interrupt	1'b0	R/W	Set if the trap was caused by an interrupt.	
30:0	exception code	31'h0	R/W	Contains a code identifying the last exception.	

Table 1.35: MCAUSE

Address	Name	Description			
0x343	MTVAL	Machine Trap Value			
Bit	Field	Initial	Access	Description	Note
31:0	mtval	32'h0	R	When a trap is taken into M-mode, mtval is either set to zero or written with exception-specific information to assist software in handling the trap. Otherwise, mtval is never written by the implementation, though it may be explicitly written by software. The hardware platform will specify which exceptions must set mtval informatively and which may unconditionally set it to zero. Instruction Address Misaligned / Access Fault → Access Address Load/Store Address Misaligned / Access Fault → Access Address BreakPoint → Breakpoint Address Illegal Instruction → The Instruction Code Environment Call → 0x00000000 Interrupt → 0x00000000	

Table 1.36: MTVAL

Address	Name	Description			
0xb00	MCYCLE	Machine Cycle Counter Lower Side			
0xb80	MCYCLEH	Machine Cycle Counter Upper Side			
Bit	Field	Initial	Access	Description	Note
31:0	cycle	32'h0	R/W	Cycle Counter (Lower/Upper) [Note on Writing] Writing mcycle stores the value to a write buffer. Writing mcycleh stores the value to mcycleh, and stores the write buffer value to mcycle, simultaneously. [Note on Reading] Reading mcycle captures {mcycleh:mcycle} into a 64bit capture buffer and outputs lower 32bit of the buffer as read data. Reading mcycleh outputs higher 32bit of the buffer as read data.	

Table 1.37: MCYCLE / MCYCLEH

Address	Name	Description			
0xb02	MINSTRET	Machine Instruction Retired Counter Lower Side			
0xb82	MINSTRETH	Machine Instruction Retired Counter Upper Side			
Bit	Field	Initial	Access	Description	Note
31:0	instret	32'h0	R/W	Instruction Retired Counter (Lower/Upper) [Note] Reading/Writing these registers follow same manner as mcycle/mcycleh.	

Table 1.38: MINSTRET / MINSTRETH

Address	Name	Description			
0x320	MCOUNTINHIBIT	Machine Counter Inhibit			
Bit	Field	Initial	Access	Description	Note
31:3	0	29'h0	R	Always 0.	
2	ir	1'b0	R/W	0 : MINSTRET/MINSTRETH increments as usual. 1 : MINSTRET/MINSTRETH does not increment.	
1	0	1'b0	R	Always 0.	
0	cy	1'b0	R/W	0 : MCYCLE/MCYCLEH increments as usual. 1 : MCYCLE/MCYCLEH does not increment.	

Table 1.39: MCOUNTINHIBIT

Address	Name	Description
0xc00	CYCLE	Read-Only Mirror of MCYCLE
0xc01	TIME	Read-Only Mirror of MTIME (Memory Mapped MTIME)
0xc02	INSTRET	Read-Only Mirror of MINSTRET
0xc80	CYCLEH	Read-Only Mirror of MCYCLEH
0xc81	TIMEH	Read-Only Mirror of MTIMEH (Memory Mapped MTIMEH)
0xc82	INSTRETH	Read-Only Mirror of MINSTRETH

[NOTE] Reading these registers follow same manner as mcycle/mcycleh.

Table 1.40: Read-Only Mirror Registers

1.14 Machine Mode MTIME (Memory Mapped)

Machine Mode MTIME, which is a periodic interrupt timer, is defined as a memory mapped peripheral. The timer counter is 64bit length; its high side is MTIMEH and its lower side is MTIME. The count increment clock source can be selected from the internal system clock or the external clock of the IP. The timer comparator is also 64bit length; its high side is MTIMECMPH and its lower side is MTIMECMP. When the 64bit timer counter matches the timer comparator, an interrupt request is generated. MTIME_CTRL has the timer enable bit, the clock select bit, the interrupt enable bit and the interrupt status bit. MTIME_DIV is the pre-scalar configuration of the count clock. MSOFTIRQ can be used to generate IRQ_SOFT interrupt request.

Memory Mapped Peripheral Base Address = 0x49000000					
Offset Address	Name	Description			
0x0000	MTIME_CTRL	MTIME Control Register			
Bit	Field	Initial	Access	Description	Note
31:4	0	28'h0	R	Always 0.	
3	ints	1'b0	R/W1C	Interrupt Status 0: No Interrupt 1: Interrupt Pending Write 1 to clear this bit.	
2	inte	1'b0	R/W	Interrupt Enable 0: Disable 1: Enable	
1	clksrc	1'b0	R/W	Clock Source 0: Internal Clock (Clock Input of this IP) 1: External Clock (slower than Internal Clock)	
0	enable	1'b0	R/W	Time Enable 0: Disable 1: Enable	

Table 1.41: MTIME_CTRL

Memory Mapped Peripheral Base Address = 0x49000000					
Offset Address	Name	Description			
0x0004	MTIME_DIV	MTIME Clock Divider			
Bit	Field	Initial	Access	Description	Note
31:10	0	22'h0	R	Always 0.	
9:0	div	10'h0	R/W	Timer Divider A Tick occurs every DIV+1 clocks.	

Table 1.42: MTIME_DIV

Memory Mapped Peripheral Base Address = 0x49000000					
Offset Address	Name	Description			
0x0008	MTIME	MTIME Timer Ticks (lower 32bits)			
0x000c	MTIMEH	MTIME Timer Ticks (higher 32bits)			
Bit	Field	Initial	Access	Description	Note
31:0	mtime	32'h0	R/W	Timer Ticks (lower 32bits / higher 32bits) Writing MTIME stores the value to a write buffer. Writing MTIMEH stores the value to MTIMEH, and stores the write buffer value to MTIME, simultaneously. Reading MTIME captures {MTIMEH:MTIME} into a 64bit capture buffer and outputs lower 32bit of the buffer as read data. Reading MTIMEH outputs higher 32bit of the buffer as read data.	

Table 1.43: MTIME / MTIMEH

Memory Mapped Peripheral Base Address = 0x49000000					
Offset Address	Name	Description			
0x0010	MTIMECMP	MTIME Timer Ticks Comparator (lower 32bits)			
0x0014	MTIMECMPPH	MTIME Timer Ticks Comparator (higher 32bits)			
Bit	Field	Initial	Access	Description	
31:0	mtimecmp	32'h0	R/W	<p>Timer Ticks Comparator (lower 32bits / higher 32bits)</p> <p>Writing MTIMECMP stores the value to a write buffer. Writing MTIMECMPPH stores the value to MTIMECMPPH, and stores the write buffer value to MTIMECMP, simultaneously.</p> <p>Reading MTIMECMP captures (MTIMECMPPH:MTIMECMP) into a 64bit capture buffer and outputs lower 32bit of the buffer as read data.</p> <p>Reading MTIMECMPPH outputs higher 32bit of the buffer as read data.</p>	Note

Table 1.44: MTIMECMP / MTIMECMPPH

Memory Mapped Peripheral Base Address = 0x49000000					
Offset Address	Name	Description			
0x0018	MSOFTIRQ	Software Interrupt Request			
Bit	Field	Initial	Access	Description	
31:1	0	31'h0	R	Always 0.	Note
0	msip	1'b0	R/W	<p>Machine Software Interrupt Pending (Request)</p> <p>0: Negate Software Interrupt</p> <p>1: Assert Software Interrupt</p>	

Table 1.45: MSOFTIRQ

1.15 INTC (Interrupt Controller) CSR

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
31:4	0	28'h0	R	Always 0.	
3:0	intcurlevel	4'b1111	R/W	Interrupt Current Level.	

Table 1.46: MINTCURLVL

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
31:4	0	28'h0	R	Always 0.	
3:0	intprelevel	4'b1111	R/W	Interrupt Previous Level.	

Table 1.47: MINTPRELVL

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
b (31~0)	enable	1'b0	R/W	Enable for IRQ[x*32+b] 0 : Disable 1 : Enable	

Table 1.48: MINTCFGENABLE 0/1

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
b (31~0)	sense	1'b0	R/W	Sense for IRQ[x*32+b] 0 : Level Sense 1 : Edge Sense	

Table 1.49: MINTCFGSENSE 0/1

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
b (31~0)	pending	1'b0	*	Pending State for IRQ[x*32+b] 0 : No Pending IRQ 1 : Pending IRQ * : if Level sense, R; if Edge Sense, R/W1C	

Table 1.50: MINTPENDING 0/1

Address	Name	Description			Note
Bit	Field	Initial	Access	Description	
31:28	priority[x*8+7]	4'b0000	R/W	Priority Level for IRQ[x*8+7]	
27:24	priority[x*8+6]	4'b0000	R/W	Priority Level for IRQ[x*8+6]	
23:20	priority[x*8+5]	4'b0000	R/W	Priority Level for IRQ[x*8+5]	
19:16	priority[x*8+4]	4'b0000	R/W	Priority Level for IRQ[x*8+4]	
15:12	priority[x*8+3]	4'b0000	R/W	Priority Level for IRQ[x*8+3]	
11:8	priority[x*8+2]	4'b0000	R/W	Priority Level for IRQ[x*8+2]	
7:4	priority[x*8+1]	4'b0000	R/W	Priority Level for IRQ[x*8+1]	
3:0	priority[x*8+0]	4'b0000	R/W	Priority Level for IRQ[x*8+0]	

Table 1.51: MINTCFGPRORITY 0/1/2/3/4/5/6/7

1.16 Floating Point CSR

The Floating Point Unit of mmRISC-1 uses Goldschmidt's Algorithm for FDIV.S and FSQRT.S instructions. A CSR FCONV, which is a dedicated CSR of mmRISC-1, configures the convergence loop count for each FDIV and FSQRT. The larger loop counts generate more precise results.

Address	Name	Description			
0x003	FCSR	Floating Point Control and Status Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:5	frm	3'b000	R/W	Floating Point Dynamic Rounding Mode FRM Mnemonic Meaning 000 RNE Round to Nearest, tied to even 001 RTZ Round to Zero 010 RDN Round Down towards minus infinite 011 RUP Round Up towards plus infinite 100 RMM Round to Nearest, ties to Max Magnitude 101 Reserved 110 Reserved 111 DYN In instruction's rm field, selects dynamic rounding mode; In Rounding Mode register, Invalid.	
4:0	fflags	5'b00000	R/W	Floating Point Acquired Exception Flags bit4 : NV Invalid Operation bit3 : DZ Divide by Zero bit2 : OF Overflow bit1 : UF Underflow bit0 : NX Inexact	

Table 1.52: FCSR

Address	Name	Description			
0x002	FRM	Floating Point Dynamic Rounding Mode			
Bit	Field	Initial	Access	Description	Note
31:3	0	29'h0	R	Always 0.	
2:0	frm	3'b000	R/W	Floating Point Dynamic Rounding Mode FRM Mnemonic Meaning 000 RNE Round to Nearest, tied to even 001 RTZ Round to Zero 010 RDN Round Down towards minus infinite 011 RUP Round Up towards plus infinite 100 RMM Round to Nearest, ties to Max Magnitude 101 Reserved 110 Reserved 111 DYN In instruction's rm field, selects dynamic rounding mode; In Rounding Mode register, Invalid.	

Table 1.53: FRM

Address	Name	Description			
0x001	FFLAGS	Floating Point Acquired Exception Flags			
Bit	Field	Initial	Access	Description	Note
31:5	0	27'h0	R	Always 0.	
4:0	fflags	5'b00000	R/W	Floating Point Acquired Exception Flags bit4 : NV Invalid Operation bit3 : DZ Divide by Zero bit2 : OF Overflow bit1 : UF Underflow bit0 : NX Inexact	

Table 1.54: FFLAGS

Address	Name	Description			
0xbe0	FCONV	Floating Point Convergence Loop Count for FDIV/FSQRT Goldschmidt's Algorithm			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:4	fsqrtconv	4'b0100	R/W	FSQRT Convergence Loop Count (default 4) 0: 16-loops 1: 1-loop 2: 2-loops ... 15: 15-loops	
3:0	fdivconv	4'b0100	R/W	FDIV Convergence Loop Count (default 4) 0: 16-loops 1: 1-loop 2: 2-loops ... 15: 15-loops	

Table 1.55: FCONV

1.17 Debug CSR

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
31:28	xdebugver	4'b0100	R	Always 4. 4: External debug support exists as it is described in RISC-V External Debug Support Version 0.13.2.	
27:16	0	12'h0	R	Always 0.	
15	ebreakm	1'b0	R/W	0: ebreak instructions in M-mode behave as described in the Privileged Spec. 1: ebreak instructions in M-mode enter Debug Mode.	
14	0	1'b0	R	Always 0.	
13	ebreaks	1'b0	R	Always 0.	
12	ebreaku	1'b0	R	Always 0.	
11	stepie	1'b0	R/W	0: Interrupts are disabled during single stepping. 1: Interrupts are enabled during single stepping. The debugger must not change the value of this bit while the hart is running.	
10	stopcount	1'b0	R/W	0: Increment counters as usual. 1: Don't increment any counters while in Debug Mode or on ebreak instructions that cause entry into Debug Mode. These counters include the cycle and instret CSRs. This is preferred for most debugging scenarios.	
9	stoptime	1'b0	R/W	0: Increment timers as usual. 1: Don't increment any hart-local timers while in Debug Mode.	
8:6	cause	3'b000	R	Explains why Debug Mode was entered. When there are multiple reasons to enter Debug Mode in a single cycle, hardware should set cause to the cause with the highest priority. 1: An ebreak instruction was executed. (priority 3) 2: The Trigger Module caused a breakpoint exception. (priority 4, highest) 3: The debugger requested entry to Debug Mode using haltreq. (priority 1) 4: The hart single stepped because step was set. (priority 0, lowest) 5: The hart halted directly out of reset due to resethaltreq. It is also acceptable to report 3 when this happens. (priority 2) Other values are reserved for future use.	
5	0	1'b0	R	Always 0.	
4	mprvn	1'b0	R	Always 0. 0: MPRV in mstatus is ignored in Debug Mode.	
3	n mip	1'b0	R	Always 0. There is no Non-Maskable-Interrupt (NMI).	
2	step	1'b0	R/W	When set and not in Debug Mode, the hart will only execute a single instruction and then enter Debug Mode. If the instruction does not complete due to an exception, the hart will immediately enter Debug Mode before executing the trap handler, with appropriate exception registers set. The debugger must not change the value of this bit while the hart is running.	
1:0	prv	2'b11	R	Always 3. Contains the privilege level the hart was operating in when Debug Mode was entered. Always M-Mode = 3.	

Table 1.56: DCSR

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
31:0	dpc	-	R/W	Upon entry to debug mode, dpc is updated with the virtual address of the next instruction to be executed. When resuming, the hart's PC is updated to the virtual address stored in dpc. A debugger may write dpc to change where the hart resumes. (1) ebreak Address of the ebreak instruction. (2) Single Step Address of the instruction that would be executed next if no debugging was going on. i.e. pc + 4 for 32-bit instructions that don't change program flow, the destination PC on taken jumps/branches, etc. (3) Trigger Module If timing in Match Control Register of Trigger Module is 0, the address of the instruction which caused the trigger to fire. If timing is 1, the address of the next instruction to be executed at the time that debug mode was entered. (4) Halt Request Address of the next instruction to be executed at the time that debug mode was entered.	

Table 1.57: DPC

1.18 Debug Trigger Module CSR

Address	Name	Description			Note
0x7a0	TSELECT	Trigger Select			
Bit	Field	Initial	Access	Description	
31:0	index	32'h0	R/W	This register determines which trigger is accessible through the other trigger registers. The set of accessible triggers must start at 0, and be contiguous. Index Number 0 to `TRG_CH_BUS - 1 : TYPE 2 Trigger (mcontrol; trigger by address and data match) Index Number `TRG_CH_BUS : TYPE 3 Trigger (icount; trigger by instruction count)	

Table 1.58: TSELECT

Address	Name	Description			Note
0x7a1	TDATA1	Trigger Data 1			
Bit	Field	Initial	Access	Description	
31:28	type	4'b0010 or 4'b0011	R	Trigger Type depends on Index Number. Write Value is ignored. Read Value is 4'h2 or 4'h3 according to Index Number as follows. Index Number 0 to `TRG_CH_BUS - 1 : TYPE 2 Trigger (mcontrol; trigger by address and data match) Index Number `TRG_CH_BUS : TYPE 3 Trigger (icount; trigger by instruction count)	
27	dmode	1'b0	R/W	0: Both Debug and M-mode can write the tdata registers at the selected tselect. 1: Only Debug Mode can write the tdata registers at the selected tselect. Writes from other modes are ignored. In original spec, this bit is only writable from Debug Mode, but this core allows writing this bit from any mode.	
26:0	data	-	R/W	Trigger Specific Data.	

Table 1.59: TDATA1

Address	Name	Description			Note
0x7a2	TDATA2	Trigger Specific Data			
Bit	Field	Initial	Access	Description	
31:0	data	32'h0	R/W	Trigger-specific data associated to mcontrol.	

Table 1.60: TDATA2

Address	Name	Description			Note
0x7a4	TINFO	Trigger Info			
Bit	Field	Initial	Access	Description	
31:16	0	16'h0	R	Always 0.	
15:0	info	16'h0	R	If tselect < `TRG_CH_BUS, always 4 (Only Type 2). If tselect == `TRG_CH_BUS, always 8 (Only Type 3). One bit for each possible type enumerated in tdata1. Bit N corresponds to type N. If the bit is set, then that type is supported by the currently selected trigger. If the currently selected trigger doesn't exist, this field contains 1.	

Table 1.61: TINFO

Address	Name	Description			
0x7a1	MCONTROL (TDATA1; Type 2)	Match Control (Trigger Data of Type 2)			
Bit	Field	Initial	Access	Description	Note
31:28	type	4'b0010	R	Trigger Type Index Number 0 to `TRG_CH_BUS - 1 are TYPE 2 Trigger. (mcontrol; trigger by address and data match)	
27	dmode	1'b0	R/W	0: Both Debug and M-mode can write the tdata registers at the selected tselect. 1: Only Debug Mode can write the tdata registers at the selected tselect. Writes from other modes are ignored. In original spec, this bit is only writable from Debug Mode, but this core allows writing this bit from any mode.	
26:21	maskmax	6'h20	R	Always 6'h20 (32). Specifies the largest naturally aligned powers-of-two (NAPOT) range supported by the hardware when match is 1. The value is the logarithm base 2 of the number of bytes in that range. A value of 0 indicates that only exact value matches are supported (one byte range). A value of 32 corresponds to the maximum NAPOT range, which is 2^32 bytes in size.	
20	hit	1'b0	R/W	The hardware sets it when this trigger matches. The trigger's user can set or clear it at any time. It is used to determine which trigger(s) matched.	
19	select	1'b0	R/W	0: Perform a match on the access address. 1: Perform a match on the data value loaded or stored, or the instruction executed.	
18	timing	1'b0	R/W	The timing bit is effective for instruction address and instruction code trigger. Data access trigger is asynchronous to instruction execution sequence. 0: The action for this trigger will be taken just before the instruction that triggered it is executed, but after all preceding instructions are committed. 1: The action for this trigger will be taken after the instruction that triggered it is executed.	
17:16	sizelo	1'b0	R/W	This field contains the 2 bits of size. 0: The trigger will attempt to match against an access of any size. The behavior is only well-defined if select = 0, or if the access size is XLEN (=32). 1: The trigger will only match against 8-bit memory accesses. 2: The trigger will only match against 16-bit memory accesses or execution of 16-bit instructions. 3: The trigger will only match against 32-bit memory accesses or execution of 32-bit instructions.	
15:12	action	4'b0000	R/W	The action to take when the trigger fires. 0: Raise a breakpoint exception. 1: Enter Debug Mode. (Only supported when dmode is 1.) other: Reserved for future use.	
11	chain	1'b0	R/W	0: When this trigger matches, the configured action is taken. 1: While this trigger does not match, it prevents the trigger with the next index from matching. A trigger chain starts on the first trigger with chain = 1 after a trigger with chain = 0, or simply on the first trigger if that has chain = 1. It ends on the first trigger after that which has chain = 0. This final trigger is part of the chain. The action on all but the final trigger is ignored. The action on that final trigger will be taken if and only if all the triggers in the chain match at the same time.	
10:7	match	4'b0000	R/W	0: Matches when the value equals tdata2. 1: Matches when the top M bits of the value match the top M bits of tdata2. M is XLEN-1 minus the index of the least-significant bit containing 0 in tdata2. 2: Matches when the value is greater than (unsigned) or equal to tdata2. 3: Matches when the value is less than (unsigned) tdata2. 4: Matches when the lower half of the value equals the lower half of tdata2 after the lower half of the value is ANDed with the upper half of tdata2. 5: Matches when the upper half of the value equals the lower half of tdata2 after the upper half of the value is ANDed with the upper half of tdata2. Other values are reserved for future use.	
6	m	1'b0	R/W	When set, enable this trigger in M-mode. This trigger module ignores this bit.	
5	0	1'b0	R	Always 0.	
4	s	1'b0	R/W	When set, enable this trigger in S-mode. This trigger module ignores this bit.	
3	u	1'b0	R/W	When set, enable this trigger in U-mode. This trigger module ignores this bit.	
2	execute	1'b0	R/W	When set, the trigger fires on the address or opcode of an instruction that is executed.	
1	store	1'b0	R/W	When set, the trigger fires on the address or data of a store.	
0	load	1'b0	R/W	When set, the trigger fires on the address or data of a load.	

Table 1.62: MCONTROL

Address	Name	Description			
0x7a1	ICOUNT (TDATA1; Type 3)	Instruction Count (Trigger Data of Type 3)			
Bit	Field	Initial	Access	Description	Note
31:28	type	4'b0010	R	Trigger Type Index Number `TRG_CH_BUS is TYPE 3 Trigger. (icount; trigger by instruction count)	
27	dmode	1'b0	R/W	0: Both Debug and M-mode can write the tdata registers at the selected tselect. 1: Only Debug Mode can write the tdata registers at the selected tselect. Writes from other modes are ignored. In original spec, this bit is only writable from Debug Mode, but this core allows writing this bit from any mode.	
26:25	0	2'b00	R	Always 0.	
24	hit	1'b0	R/W	The hardware sets it when this trigger matches. The trigger's user can set or clear it at any time. It is used to determine which trigger(s) matched.	
23:10	count	14'h0	R/W	When count is decremented to 0, the trigger fires.	
9	m	1'b0	R/W	When set, every instruction completed or exception taken in M-mode decrements count by 1.	
8	0	1'b0	R	Always 0.	
7	s	1'b0	R	Always 0.	
6	u	1'b0	R	Always 0.	
5:0	action	6'h0	R/W	The action to take when the trigger fires. 0: Raise a breakpoint exception. 1: Enter Debug Mode. (Only supported when dmode is 1.) other: Reserved for future use.	

Table 1.63: ICOUNT

1.19 DTM (Debug Transport Module) JTAG Register

The address of the JTAG register is specified by the JTAG tap IR (5bits).

Address	Name	Description			
0x01	IDCODE	JTAG TAP IDCODE This register is selected (in IR) when the TAP state machine is reset. Its definition is exactly as defined in IEEE Std 1149.1-2013. This entire register is read-only.			
Bit	Field	Initial	Access	Description	Note
31:28	Version	4'b0001	R	Identifies the release version of this part.	All fields are configurable by modifying 'JTAG_IDCODE' in defines.v.
27:12	PartNumber	16'h6d6d	R	Identifies the designer's part number of this part.	
11:1	Nanufld	11'h000	R	Identifies the designer/manufacturer of this part. Bits 6:0 must be bits 6:0 of the designer/manufacturer's Identification Code as assigned by JEDEC Standard JEP106. Bits 10:7 contain the modulo-16 count of the number of continuation characters (0x7f) in that same Identification Code.	All fields are configurable by modifying 'JTAG_IDCODE' in defines.v.
0	1	1'b1	R	Always 1.	

Table 1.64: IDCODE

Address	Name	Description			
0x1f	BYPASS	Bypass Register 1-bit register that has no effect. It is used when a debugger does not want to communicate with this TAP. This entire register is read-only.			
Bit	Field	Initial	Access	Description	Note
0	bypass	-	R/W	Bypass register for JTAG	

Table 1.65: BYPASS

Address	Name	Description			
0x10	DTMCS	DTM Control and Status The size of this register will remain constant in future versions so that a debugger can always determine the version of the DTM.			
Bit	Field	Initial	Access	Description	Note
31:18	0	14'h0	R	-	
17	dmihardreset	1'b0	W1	Writing 1 to this bit does a hard reset of the DTM, causing the DTM to forget about any outstanding DMI transactions. In general this should only be used when the Debugger has reason to expect that the outstanding DMI transaction will never complete (e.g. a reset condition caused an inflight DMI transaction to be cancelled).	
16	dmireset	1'b0	W1	Writing 1 to this bit clears the sticky error state and allows the DTM to retry or complete the previous transaction.	
15	0	1'b0	R	-	
14:12	idle	3'b101	R	This is a hint to the debugger of the minimum number of cycles a debugger should spend in Run-Test/Idle after every DMI scan to avoid a 'busy' return code (dmistat of 3). A debugger must still check dmistat when necessary. 0: It is not necessary to enter Run-Test/Idle at all. 1: Enter Run-Test/Idle and leave it immediately. 2: Enter Run-Test/Idle and stay there for 1 cycle before leaving. And so on.	
11:10	dmistat	3'b000	R	0: No error. 1: Reserved. Interpret the same as 2. 2: An operation failed (resulted in op of 2). 3: An operation was attempted while a DMI access was still in progress (resulted in op of 3).	The "op" is a field in DMI
9:4	abits	6'h7	R	The size of address in dmi.	
3:0	version	4'b0001	R	0: Version described in spec version 0.11. 1: Version described in spec version 0.13. 15: Version not described in any available version of this spec.	

Table 1.66: DTMCS

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
40:34	address	7'h0	R/W	Address used for DMI access. In Update-DR this value is used to access the DM over the DMI.	
33:2	data	32'h0	R/W	The data to send to the DM over the DMI during Update-DR, and the data returned from the DM as a result of the previous operation.	
1:0	op	2'b00	R/W	<p>When the debugger writes this field, it has the following meaning:</p> <p>0: Ignore data and address. (nop) Don't send anything over the DMI during Update-DR. This operation should never result in a busy or error response. The address and data reported in the following Capture-DR are undefined.</p> <p>1: Read from address. (read)</p> <p>2: Write data to address. (write)</p> <p>3: Reserved.</p> <p>When the debugger reads this field, it means the following:</p> <p>0: The previous operation completed successfully.</p> <p>1: Reserved.</p> <p>2: A previous operation failed. The data scanned into dmi in this access will be ignored. This status is sticky and can be cleared by writing dmireset in dtmcs. This indicates that the DM itself responded with an error. There are no specified cases in which the DM would respond with an error, and DMI is not required to support returning errors.</p> <p>3: An operation was attempted while a DMI request is still in progress. The data scanned into dmi in this access will be ignored. This status is sticky and can be cleared by writing dmireset in dtmcs. If a debugger sees this status, it needs to give the target more TCK edges between Update-DR and Capture-DR. The simplest way to do that is to add extra transitions in Run-Test/Idle.</p>	

Table 1.67: DMI

1.20 DM (Debug Module) Register

The DM (Debug Module) registers are accessed by the DMI bus from the DTM. The address is specified by the DMI register according to the JTAG access.

Address	Name	Description			
0x11	DMSTATUS	Debug Module Status			
Bit	Field	Initial	Access	Description	Note
31:23	0	9'h0	R	Always 0.	
22	impebreak	1'b0	R	Always 0, because there are no Program Buffers, and no implicit ebreak instruction at the non-existent word immediately after the Program Buffer.	
21:20	0	2'b00	R	Always 0.	
19	allhavereset	-	R	This field is 1 when all currently selected harts have been reset and reset has not been acknowledged for any of them.	
18	anyhavereset	-	R	This field is 1 when at least one currently selected hart has been reset and reset has not been acknowledged for that hart.	
17	allresumeack	-	R	This field is 1 when all currently selected harts have acknowledged their last resume request.	
16	anyresumeack	-	R	This field is 1 when any currently selected hart has acknowledged its last resume request.	
15	allnonexistent	-	R	This field is 1 when all currently selected harts do not exist in this platform.	
14	anynonexistent	-	R	This field is 1 when any currently selected hart does not exist in this platform.	
13	allunavail	-	R	This field is 1 when all currently selected harts are unavailable.	Unavailable when STBY
12	anyunavail	-	R	This field is 1 when any currently selected hart is unavailable.	
11	allrunning	-	R	This field is 1 when all currently selected harts are running.	
10	anyrunning	-	R	This field is 1 when any currently selected hart is running.	
9	allhalted	-	R	This field is 1 when all currently selected harts are halted.	
8	anyhalted	-	R	This field is 1 when any currently selected hart is halted.	
7	authenticated	-	R	0: Authentication is required before using the DM. 1: The authentication check has passed. On components that don't implement authentication, this bit must be preset as 1.	
6	authbusy	-	R	0: The authentication module is ready to process the next read/write to authdata. 1: The authentication module is busy. Accessing authdata results in unspecified behavior. authbusy only becomes set in immediate response to an access to authdata.	
5	hasresethaltreq	1'b1	R	Always 1. This Debug Module supports halt-on-reset functionality controllable by the setresethaltreq and clrresethaltreq bits.	
4	confstrptrvalid	1'b0	R	Always 0. 0: confstrptr0{confstrptr3 hold information which is not relevant to the configuration string. 1: confstrptr0{confstrptr3 hold the address of the configuration string.	
3:0	version	4'b0010	R	Always 2. 2: There is a Debug Module and it conforms to version 0.13.	

Table 1.68: DMSTATUS

Address	Name	Description			
0x10	DMCONTROL	Debug Module Control			
Bit	Field	Initial	Access	Description	Note
31	haltreq	-	W	Writing 0 clears the halt request bit for all currently selected harts. This may cancel outstanding halt requests for those harts. Writing 1 sets the halt request bit for all currently selected harts. Running harts will halt whenever their halt request bit is set. Writes apply to the new value of hartsel and hasel.	
30	resumereq	-	W1	Writing 1 causes the currently selected harts to resume once, if they are halted when the write occurs. It also clears the resume ack bit for those harts. resumereq is ignored if haltreq is set. Writes apply to the new value of hartsel and hasel.	
29	hartreset	1'b0	R/W	This field writes the reset bit for all the currently selected harts. To perform a reset the debugger writes 1, and then writes 0 to deassert the reset signal. While this bit is 1, the debugger must not change which harts are selected. Writes apply to the new value of hartsel and hasel.	
28	ackhavereset	-	W1	0: No effect. 1: Clears havereset for any selected harts. Writes apply to the new value of hartsel and hasel.	
27	0	1'b0	R	Always 0.	
26	hasel	1'b0	R/W	Selects the definition of currently selected harts. 0: There is a single currently selected hart, that is selected by hartsel. 1: There may be multiple currently selected harts; the hart selected by hartsel, plus those selected by the hart array mask register. A debugger which wishes to use the hart array mask register feature should set this bit and read back to see if the functionality is supported.	
25:16	hartsello	10'h0	R/W	The low 10 bits of hartsel: the DM-specific index of the hart to select. This hart is always part of the currently selected harts.	
15:6	hartselhi	10'h0	R/W	The high 10 bits of hartsel: the DM-specific index of the hart to select. This hart is always part of the currently selected harts.	
5:4	0	2'b00	R	Always 0.	
3	setresethaltreq	-	W1	This field writes the halt-on-reset request bit for all currently selected harts, unless clresethaltreq is simultaneously set to 1. When set to 1, each selected hart will halt upon the next deassertion of its reset. The halt-on-reset request bit is not automatically cleared. The debugger must write to clresethaltreq to clear it. Writes apply to the new value of hartsel and hasel. If hasresethaltreq is 0, this field is not implemented.	
2	clresethaltreq	-	W1	This field clears the halt-on-reset request bit for all currently selected harts. Writes apply to the new value of hartsel and hasel.	
1	ndmreset	1'b0	R/W	This bit controls the reset signal from the DM to the rest of the system. The signal should reset every part of the system, including every hart, except for the DM and any logic required to access the DM. To perform a system reset the debugger writes 1, and then writes 0 to deassert the reset.	
0	dmactive	1'b0	R/W	This bit serves as a reset signal for the Debug Module itself. 0: The module's state, including authentication mechanism, takes its reset values (the dmactive bit is the only bit which can be written to something other than its reset value). 1: The module functions normally. No other mechanism should exist that may result in resetting the Debug Module after power up, with the possible (but not recommended) exception of a global reset signal that resets the entire platform. A debugger may pulse this bit low to get the Debug Module into a known state. Implementations may pay attention to this bit to further aid debugging, for example by preventing the Debug Module from being power gated while debugging is active.	

Table 1.69: DMCONTROL

Address	Name	Description			
0x12	HARTINFO	Hart Info			
Bit	Field	Initial	Access	Description	Note
31:24	0	8'h0	R	Always 0.	
23:20	nscratch	4'b0000	R	Always 0. Number of dscratch registers available for the debugger to use during program buffer execution, starting from dscratch0. The debugger can make no assumptions about the contents of these registers between commands.	
19:17	0	3'b000	R	Always 0.	
16	dataaccess	1'b0	R	Always 0. 0: The data registers are shadowed in the hart by CSRs. Each CSR is DXLEN bits in size, and corresponds to a single argument.	No data registers shadowed in CSR.
15:12	datasize	4'b0000	R	Always 0. Number of CSRs dedicated to shadowing the data registers.	
11:0	dataaddress	12'h0	R	Always 0. The number of the first CSR dedicated to shadowing the data registers.	

Table 1.70: HARTINFO

Address	Name	Description			
0x14	HAWINDOWSEL	Hart Array Window Select			
Bit	Field	Initial	Access	Description	Note
31:15	0	17'h0	R	Always 0.	
14:0	hawindowsel	15'h0	R/W	The high bits of this field may be tied to 0, depending on how large the array mask register is. E.g. on a system with 48 harts only bit 0 of this field may actually be writable.	

Table 1.71: HAWINDOWSEL

Address	Name	Description			
0x15	HAWINDOW	Hart Array Window			
Bit	Field	Initial	Access	Description	Note
31:0	maskdata	16'h0	R/W	This register provides R/W access to a 32-bit portion of the hart array mask register. The position of the window is determined by hawindowsel. i.e. bit 0 refers to hart hawindowsel * 32, while bit 31 refers to hart hawindowsel * 32 + 31. Since some bits in the hart array mask register may be constant 0, some bits in this register may be constant 0, depending on the current value of hawindowsel.	

Table 1.72: HAWINDOW

Address	Name	Description			
0x30	AUTHDATA	Authentication Data			
Bit	Field	Initial	Access	Description	Note
31:0	data	32'h0	R/W	This register serves as a 32-bit serial port to/from the authentication module. When authbusy is clear, the debugger can communicate with the authentication module by reading or writing this register. There is no separate mechanism to signal over ow/underflow. Authentication is effective when input signal DEBUG_SECURE = 1'b1. and when this register is equal to input signal DEBUG_SECURE_CODE[31:0], authentication is OK.	

Table 1.73: AUTHDATA

Address	Name	Description			
0x40	HALTSUM0	Halt Summary 0			
Bit	Field	Initial	Access	Description	Note
31:0	haltsum0	depends on each hart's status	R	Each bit in this read-only register indicates whether one specific hart is halted or not. Unavailable/nonexistent harts are not considered to be halted. The LSB reflects the halt status of hart {hartsel[1:9:5], 5'h0}, and the MSB reflects halt status of hart {hartsel[19:5], 5'h1f}. This entire register is read-only.	

Table 1.74: HALTSUM0

Address	Name	Description			
0x13	HALTSUM1	Halt Summary 1			
Bit	Field	Initial	Access	Description	Note
31:0	haltsum1	depends on each hart's status	R	Each bit in this read-only register indicates whether any of a group of harts is halted or not. Unavailable/nonexistent harts are not considered to be halted. This register may not be present in systems with fewer than 33 harts. The LSB reflects the halt status of harts {hartsel[19:10], 10'h0} through {hartsel[19:10], 10'h1f}. The MSB reflects the halt status of harts {hartsel[19:10], 10'h3e0} through {hartsel[19:10], 10'h3ff}. This entire register is read-only.	

Table 1.75: HALTSUM1

Address	Name	Description			
0x34	HALTSUM2	Halt Summary 2			
Bit	Field	Initial	Access	Description	Note
31:0	haltsum2	depends on each hart's status	R	Each bit in this read-only register indicates whether any of a group of harts is halted or not. Unavailable/nonexistent harts are not considered to be halted. This register may not be present in systems with fewer than 1025 harts. The LSB reflects the halt status of harts {hartsel[19:15],15'h0} through {hartsel[19:15],15'h3ff}. The MSB reflects the halt status of harts {hartsel[19:15],15'h7c00} through {hartsel[19:15],15'h7fff}. This entire register is read-only.	

Table 1.76: HALTSUM2

Address	Name	Description			
0x35	HALTSUM3	Halt Summary 3			
Bit	Field	Initial	Access	Description	Note
31:0	haltsum3	depends on each hart's status	R	Each bit in this read-only register indicates whether any of a group of harts is halted or not. Unavailable/nonexistent harts are not considered to be halted. This register may not be present in systems with fewer than 32769 harts. The LSB reflects the halt status of harts 20'h0 through 20'h7fff. The MSB reflects the halt status of harts 20'hf8000 through 20'hfffff. This entire register is read-only.	

Table 1.77: HALTSUM3

1.21 DM (Debug Module) Register : Abstract Command

The DM (Debug Module) registers are accessed by the DMI bus from the DTM. The address is specified by the DMI register according to the JTAG access.

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
0x04	DATA0			Abstract Data 0 ; arg0 / return value	
0x05	DATA1			Abstract Data 1 ; arg1	

Table 1.78: DATA 0/1

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
0x17	COMMAND			Abstract Command : Access Register	
31:24	cmdtype	8'h0	W	This is 0 to indicate Access Register Command.	
23	0	1'b0	W	Always 0.	
22:20	aarsize	3'b000	W	Always 2: Access the lowest 32 bits of the register. If aarsize is not 2, then the access must fail. If a register is accessible, then reads of aarsize equal to the register's actual size must be supported.	
19	aarpostincrement	1'b0	W	Always 0. Not Supported.	
18	postexec	1'b0	W	Always 0. Not Supported.	
17	transfer	1'b0	W	0: Don't do the operation specified by write. 1: Do the operation specified by write.	
16	write	1'b0	W	When transfer is set: 0: Copy data from the specified register into arg0 portion of data. 1: Copy data from arg0 portion of data into the specified register.	
15:0	regno	16'h0	W	Number of the register to access. 0x0000 – 0xffff : CSR 0x1000 – 0x101f: GPR (x0 – x31) 0x1020 – 0x103f: FPR (f0 – f31) 0xc000 – 0xffff: Reserved dpc may be used as an alias for PC. This command is supported on a non-halted hart.	

Table 1.79: COMMAND for Access Register

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
0x17	COMMAND			Abstract Command : Access Memory	
31:24	cmdtype	8'h2	W	This is 2 to indicate Access Memory Command.	
23	aamvirtual	1'b0	W	Always 0. Addresses are physical.	
22:20	aamsize	3'b000	W	0: Access the lowest 8 bits of the memory location. 1: Access the lowest 16 bits of the memory location. 2: Access the lowest 32 bits of the memory location.	
19	aampostincrement	1'b0	W	After a memory access has completed, if this bit is 1, increment arg1 (which contains the address used) by the number of bytes encoded in aamsize.	
18:17	0	2'b00	W	Always 0.	
16	write	1'b0	W	0: Copy data from the memory location specified in arg1 into arg0 portion of data. 1: Copy data from arg0 portion of data into the memory location specified in arg1.	
15:14	target-specific	16'h0	W	Always 0. Not Supported.	
13:0	0	14'h0		Always 0.	

Table 1.80: COMMAND for Access Memory

Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
31:29	0	3'b000	R	Always 0.	
28:24	progbuftype	5'b00000	R	Always 0. Program Buffer type is zero, so it is not supported.	
23:13	0	11'h0	R	Always 0.	
12	busy	1'b0	R	1: An abstract command is currently being executed. This bit is set as soon as command is written, and is not cleared until that command has completed.	
11	0	1'b0	R	Always 0.	
10:8	cmderr	3'b000	R/WC1	Gets set if an abstract command fails. The bits in this field remain set until they are cleared by writing 1 to them. No abstract command is started until the value is reset to 0. This field only contains a valid value if busy is 0. 0 (none): No error. 1 (busy): An abstract command was executing while command, abstractcs was written. This status is only written if cmderr contains 0. 2 (not supported): The requested command is not supported, regardless of whether the hart is running or not. 3 (exception): An exception occurred while executing the command. 4 (halt/resume): The abstract command couldn't execute because the hart wasn't in the required state (running/halted), or unavailable. 5 (bus): The abstract command failed due to a bus error (e.g. alignment, access size, or timeout). 7 (other): The command failed for another reason.	
7:4	0	4'b0000	R	Always 0.	
3:0	datacount	4'b0010	R	Number of data registers that are implemented as part of the abstract command interface. Always 2.	

Table 1.81: ABSTRACTCS

1.22 DM (Debug Module) Register : Abstract Command

The DM (Debug Module) registers are accessed by the DMI bus from the DTM. The address is specified by the DMI register according to the JTAG access.

Address	Name	Description			
0x38	SBCS	System Bus Access Control and Status			
Bit	Field	Initial	Access	Description	Note
31:29	sbversion	3'b001	R	1: The System Bus interface conforms to this version of the spec.	
28:23	0	6'b000000	R	Always 0.	
22	sbbusyerror	1'b0	R/W1C	Set when the debugger attempts to read data while a read is in progress, or when the debugger initiates a new access while one is already in progress (while sbbusy is set). It remains set until it's explicitly cleared by the debugger. While this field is set, no more system bus accesses can be initiated by the Debug Module.	
21	sbbusy	1'b0	R	When 1, indicates the system bus master is busy. (Whether the system bus itself is busy is related, but not the same thing.) This bit goes high immediately when a read or write is requested for any reason, and does not go low until the access is fully completed. Writes to sbcs while sbbusy is high result in undefined behavior. A debugger must not write to sbcs until it reads sbbusy as 0.	
20	sbreadonaddr	1'b0	R/W	When 1, every write to sbaddress0 automatically triggers a system bus read at the new address.	
19:17	sbaccess	3'b000	R/W	Select the access size to use for system bus accesses. 0: 8-bit 1: 16-bit 2: 32-bit If sbaccess has an unsupported value when the DM starts a bus access, the access is not performed and sberror is set to 4.	
16	sbautoincrement	1'b0	R/W	When 1, sbaddress is incremented by the access size (in bytes) selected in sbaccess after every system bus access.	
15	sbreadondata	1'b0	R/W	When 1, every read from sbdata0 automatically triggers a system bus read at the (possibly auto-incremented) address.	
14:12	sberror	3'b000	R/W1C	When the Debug Module's system bus master encounters an error, this field gets set. The bits in this field remain set until they are cleared by writing 1 to them. While this field is non-zero, no more system bus accesses can be initiated by the Debug Module. An implementation may report "Other" (7) for any error condition. 0: There was no bus error. 1: There was a timeout. 2: A bad address was accessed. 3: There was an alignment error. 4: An access of unsupported size was requested. 7: Other.	
11:5	sbasize	7'h20	R	Width of system bus addresses in bits. Always 32 (7'h20).	
4	sbaccess128	1'b0	R	1 when 128-bit system bus accesses are supported. Not supported.	
3	sbaccess64	1'b0	R	1 when 64-bit system bus accesses are supported. Not supported.	
2	sbaccess32	1'b1	R	1 when 32-bit system bus accesses are supported.	
1	sbaccess16	1'b1	R	1 when 16-bit system bus accesses are supported.	
0	sbaccess8	1'b1	R	1 when 8-bit system bus accesses are supported.	

Table 1.82: SBCS

Address	Name	Description			
0x39	SBADDRESS0	System Bus Address 31:0			
Bit	Field	Initial	Access	Description	Note
31:0	address	32'h0	R/W	Physical 32bit address of system bus access.	

Table 1.83: SBADDRESS0

Address	Name	Description			
0x3c	SBDATA0	System Bus Data 31:0			
Bit	Field	Initial	Access	Description	Note
31:0	data	32'h0	R/W	32bit data of system bus access.	

Table 1.84: SBDATA0

1.23 CPU Pipeline Structure

For the mmRISC-1 core, the RV32IMAC instruction pipelines consist of 3-stages to 5-stages as shown in Figure 1.9.

Instructions are pre-fetched and queued in the instruction fetch unit which is expressed as “F”, and sent to the next decode stage “D”. Execution stage is shown by “E”, the data memory access stage is “M” and the write back stage is “W”.

ALU and Integer Multiplication instructions finish at “E” stage, that is, register to register operations are completed in “E” stage.

Integer Division and Remainder instructions use the non-restoring method and take several clocks to complete.

Jump instructions generate the target address in the decode stage, and the target address is sent to both the instruction fetch unit and the instruction bus at the next cycle, then the instruction bus gets the code from the instruction memory at “F” stage which is sent to “D” stage at the next cycle of “F”.

Conditional branch instructions judge whether to take or not to take in “D” stage, and if taken, then follow the same manner as jump instructions. Therefore “D” stage needs register (XRn) forwarding from “E” stage or “W” stage to judge whether to take or not to take.

Data load instructions generate the access address and send it to the data memory bus at “E” stage. Then, the data bus gets the read data from the data memory at “M” stage, and write back the data to the specified register at “W” stage. If any instruction after the load instruction refers to the loading data (register conflict), the next instruction stalls its “D” stage.

Data store instructions generate the access address and data to write at “E” stage, and send them to the data memory in “E” stage, then writing to the data memory is completed in “M” stage.

Detail CPU pipeline logic structure is shown in Figure 1.10.

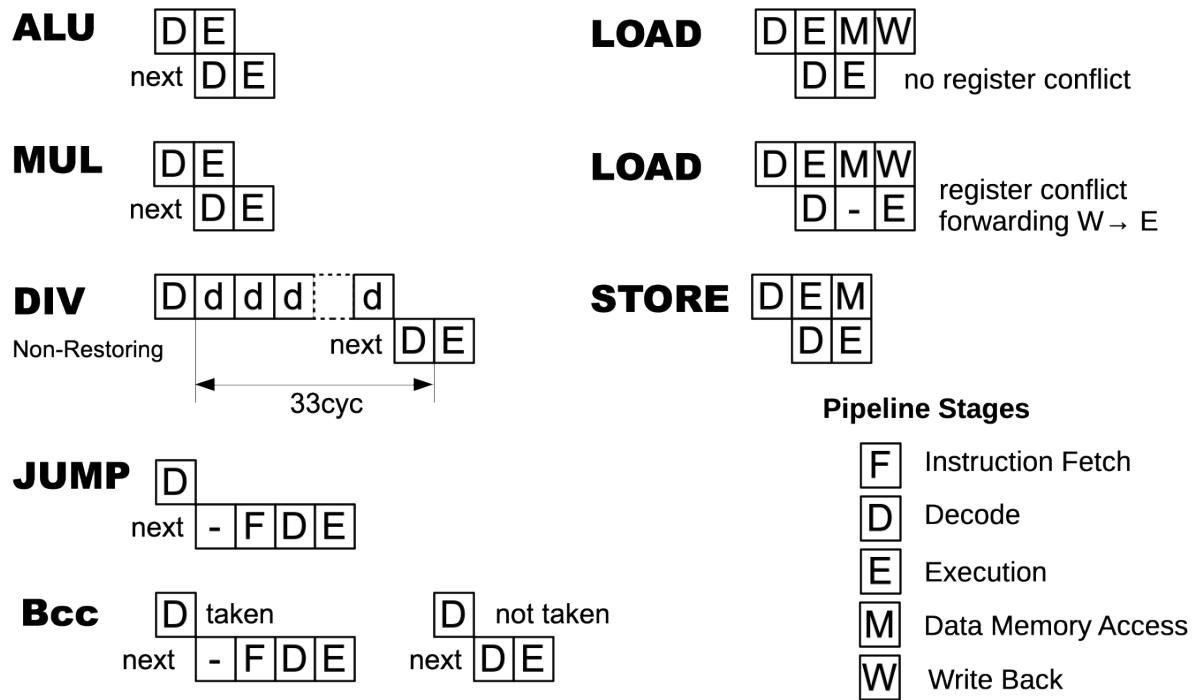


Figure 1.9: CPU Pipeline

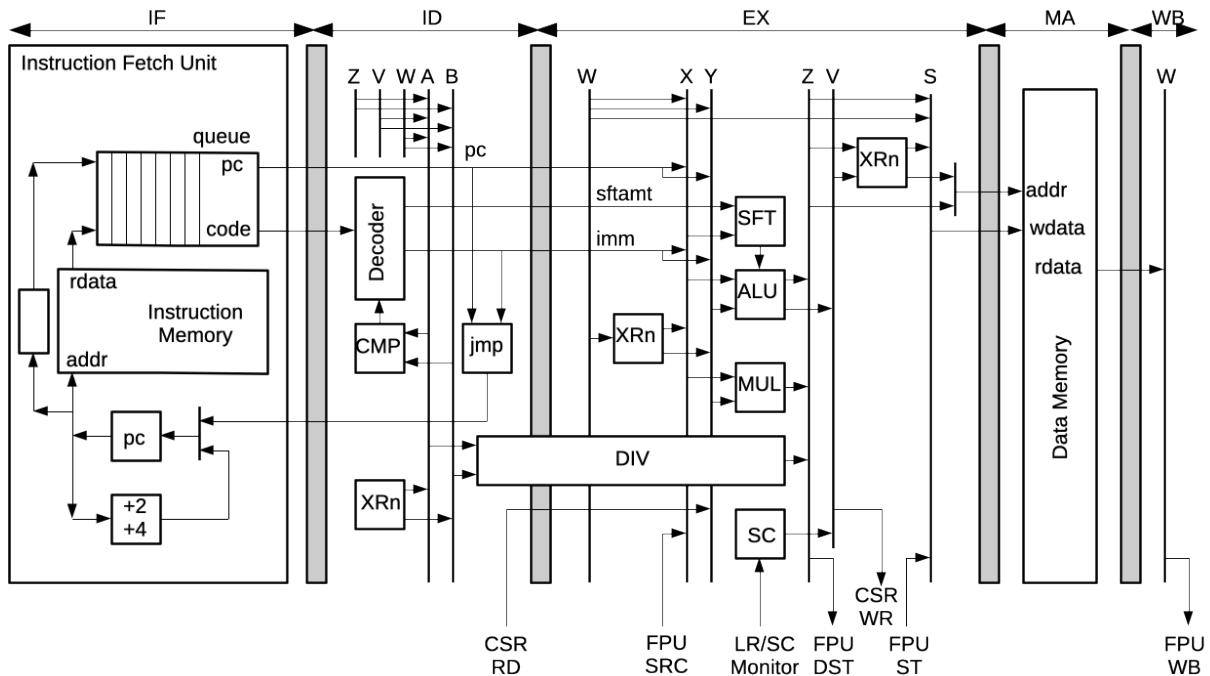


Figure 1.10: CPU Pipeline Logic Structure

1.24 FPU Pipeline Structure

For the mmRISC-1 core, the RV32F/RV32FC (FPU) instructions have an independent pipeline to the CPU (integer). If there are no register conflict between the CPU and the FPU, each pipeline can go on in parallel. The FPU supports IEEE754 normal, sub-normal, infinite, zero, quiet-nan, signaling-nan and positive/negative for each. The supported rounding modes are RNE (Round to Nearest, tied to even), RTZ (Round to Zero), RDN(Round Down towards minus infinite), RUP (Round Up towards plus infinite), RMM (Round to Nearest, ties to Max Magnitude).

As shown in Figure 1.11, the FPU pipelines consist of a combination of “i” stage, “a” stage, ‘m’ stage and “f” stage. The “i” stage means “initial” that converts 32bit IEEE754 format data in FRx registers to an internal number format in which there are 1bit sign, 12bit exponent and 66bit fraction. May be, this FPU will easily be able to support 64bit double type in the near future. The “a” stage executes addition and subtraction between the internal formats. The “m” stage executes multiplication between the internal formats. The “f” stage means “finalization” that converts the internal number format to 32bit IEEE754 format and stores the result to an FRx register.

FADD.S and FSUB.S instructions consist of “i”, “a” and “f” stages. FMUL.S instruction consists of “i”, “m” and “f” stages. Among FADD.S, FSUB.S and FMUL.S, these instructions can be executed contiguously without stalls if there are no register conflicts each other.

FMADD.S, FMSUB.S, FNMAADD.S and FNMSUB.S instructions consist of “i”, “m”, “a” and “f” stages. Among FMADD.S, FMSUB.S, FNMAADD.S and FNMSUB.S, these instructions can be executed contiguously without stalls if there are no register conflicts. But, between FADD.S etc. and FMADD.S etc, they use common stages, so these contiguous combination of instructions insert appropriate stalls in the pipeline flow to avoid resource conflict. Also, if there are register conflicts between instructions, appropriate stalls are inserted in the pipeline flow.

FDIV.S and FSQRT.S use Goldschmidt’s Algorithm, so each pipeline consists of “i”, a combination of multiple “a” and “m”, and a final “f” stage. Each convergence loop counts can be configured by the CSR FCONV. For the default configuration, the execution cycle of FDIV.S is 11, and FSQRT.S is 19.

Pipelines of instructions to load and store FRn (FLW, FSW) are just same as CPU ones.

Pipelines of instructions to convert between integer and float (FCVT.W.S, FCVT.WU.S, FCVTS.W and FCVT.S.WU) consist of “i” stage and “c” stage which is a dedicated stage for conversion.

Pipelines of instruction for operations between FRn and FRn (FSGNJ.S, FSGNJS.S, FSGNJSX.S, FMIN.S and FMAX.S) and operations between FRn and XRn (FMV.X.W, FMV.W.X, FCLASS.S, FEQ.S, FLT.S and FLE.S) complete in only “i” stage which has almost same meaning as “E” stage of CPU.

Detail FPU pipeline logic structure is shown in Figure 1.12.

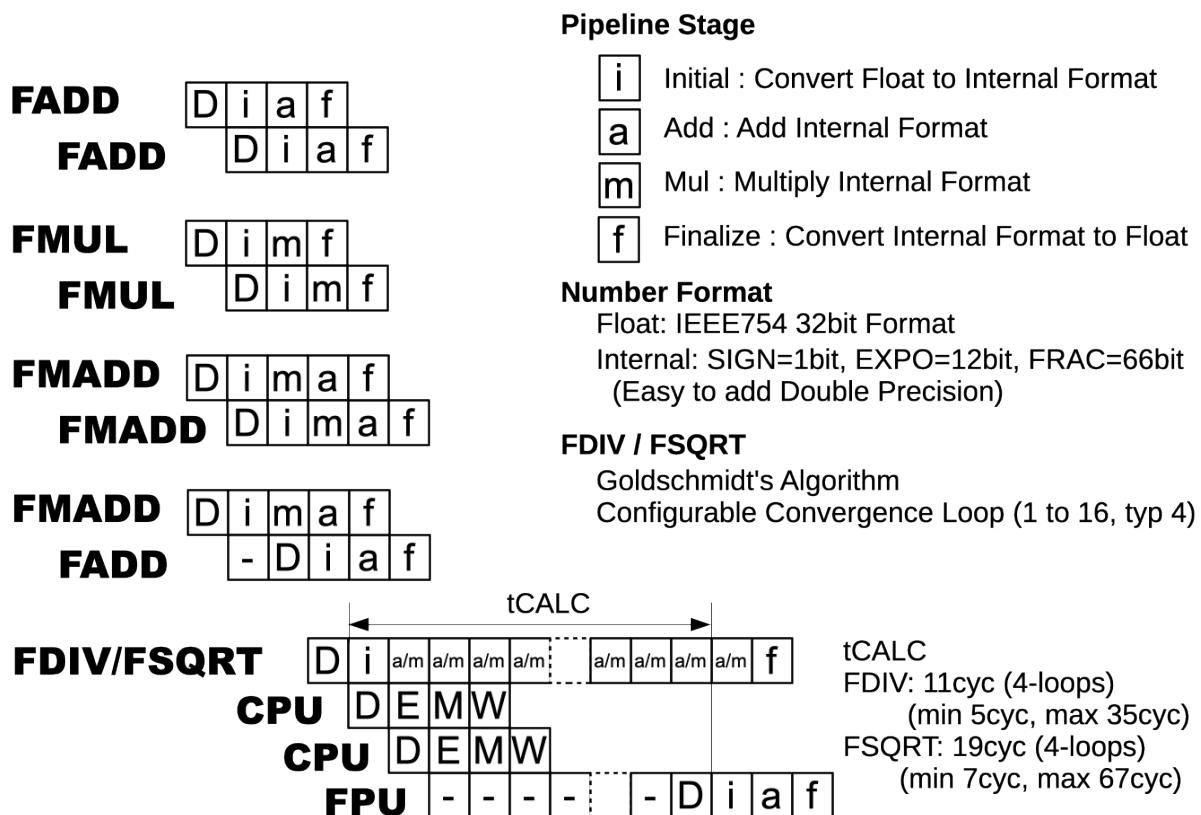


Figure 1.11: FPU Pipeline

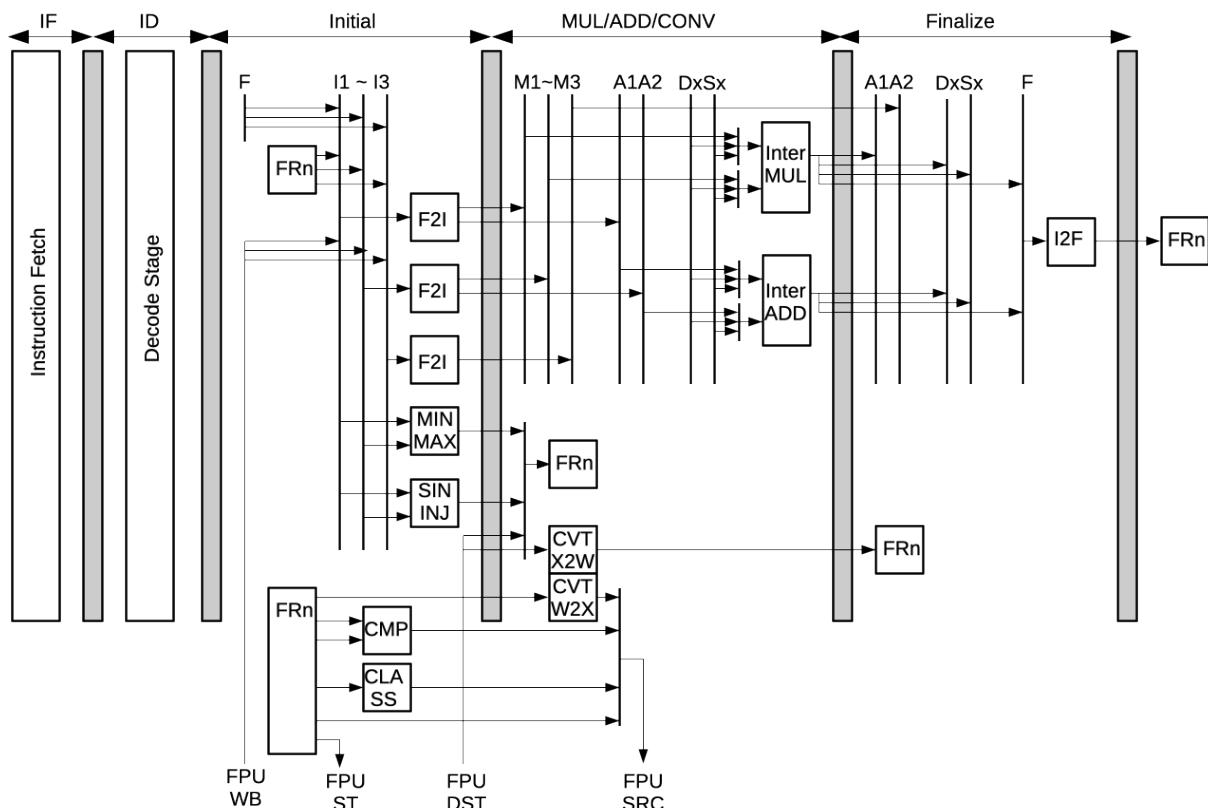


Figure 1.12: FPU Pipeline Logic Structure

1.25 32bit ISA Specifications

RV32I Base Instruction Set			
Class	Mnemonic	Meaning	Operation
Arithmetic	LUI rd, imm	Load Upper Immediate	$rd \leftarrow \{imm, 12'b0\}$
Arithmetic	AUIPC rd, imm	Add Upper Immediate to PC	$rd \leftarrow \{imm, 12'b0\} + PC$
Arithmetic	ADD rd, rs1, rs2	Add	$rd \leftarrow rs1 + rs2$
Arithmetic	SUB rd, rs1, rs2	Subtract	$rd \leftarrow rs1 - rs2$
Arithmetic	ADDI rd, rs1, imm	Add Immediate	$rd \leftarrow rs1 + imm(SE)$
Shift	SLL rd, rs1, rs2	Shift Left Logical	$rd \leftarrow rs1 << rs2[4:0]$ Logical
Shift	SRL rd, rs1, rs2	Shift Right Logical	$rd \leftarrow rs1 >> rs2[4:0]$ Logical
Shift	SRA rd, rs1, rs2	Shift Right Arithmetic	$rd \leftarrow rs1 >> rs2[4:0]$ Arithmetic
Shift	SLLI rd, rs1, shamt	Shift Left Immediate Logical	$rd \leftarrow rs1 << shamt$ Logical
Shift	SRLI rd, rs1, shamt	Shift Right Immediate Logical	$rd \leftarrow rs1 >> shamt$ Logical
Shift	SRAI rd, rs1, shamt	Shift Right Immediate Arithmetic	$rd \leftarrow rs1 >> shamt$ Arithmetic
Logical	XOR rd, rs1, rs2	XOR	$rd \leftarrow rs1 ^ rs2$
Logical	OR rd, rs1, rs2	OR	$rd \leftarrow rs1 rs2$
Logical	AND rd, rs1, rs2	AND	$rd \leftarrow rs1 \& rs2$
Logical	XORI rd, rs1, imm	XOR Immediate	$rd \leftarrow rs1 ^ imm(SE)$
Logical	ORI rd, rs1, imm	OR Immediate	$rd \leftarrow rs1 imm(SE)$
Logical	ANDI rd, rs1, imm	AND Immediate	$rd \leftarrow rs1 \& imm(SE)$
Compare	SLT rd, rs1, rs2	Set Less Than (S)	$rd \leftarrow (rs1 < rs2) (S)$
Compare	SLTU rd, rs1, rs2	Set Less Than (U)	$rd \leftarrow (rs1 < rs2) (U)$
Compare	SLTI rd, rs1, imm	Set Less Than Immediate (S)	$rd \leftarrow (rs1 < imm(SE)) (S)$
Compare	SLTIU rd, rs1, imm	Set Less Than Immediate (U)	$rd \leftarrow (rs1 < imm(SE)) (U)$
Branch	BEQ rs1, rs2, imm	Branch Equal	$PC \leftarrow PC + imm(SE)$ if $rs1 == rs2$
Branch	BNE rs1, rs2, imm	Branch Not Equal	$PC \leftarrow PC + imm(SE)$ if $rs1 != rs2$
Branch	BLT rs1, rs2, imm	Branch Less Than (S)	$PC \leftarrow PC + imm(SE)$ if $rs1 < rs2 (S)$
Branch	BGE rs1, rs2, imm	Branch Greater or Equal (S)	$PC \leftarrow PC + imm(SE)$ if $rs1 >= rs2 (S)$
Branch	BLTU rs1, rs2, imm	Branch Less Than (U)	$PC \leftarrow PC + imm(SE)$ if $rs1 < rs2 (U)$
Branch	BGEU rs1, rs2, imm	Branch Greater or Equal (U)	$PC \leftarrow PC + imm(SE)$ if $rs1 >= rs2 (U)$
Jump & Link	JAL rd, imm	Jump and Link	$rd \leftarrow PC+4, PC \leftarrow PC + imm(SE)$
Jump & Link	JALR rd, imm	Jump and Link Register	$rd \leftarrow PC+4, PC \leftarrow rs1 + imm(SE)$
Load	LB rd, rs1, imm	Load Byte	$rd \leftarrow MB[rs1 + imm(SE)]$ with SE
Load	LH rd, rs1, imm	Load Halfword	$rd \leftarrow MH[rs1 + imm(SE)]$ with SE
Load	LW rd, rs1, imm	Load Word	$rd \leftarrow MW[rs1 + imm(SE)]$
Load	LBU rd, rs1, imm	Load Byte Unsigned	$rd \leftarrow MB[rs1 + imm(SE)]$ with ZE
Load	LHU rd, rs1, imm	Load Halfword Unsigned	$rd \leftarrow MH[rs1 + imm(SE)]$ with ZE
Store	SB rs1, rs2, imm	Store Byte	$MB[rs1 + imm(SE)] \leftarrow rs2$
Store	SH rs1, rs2, imm	Store Halfword	$MH[rs1 + imm(SE)] \leftarrow rs2$
Store	SW rs1, rs2, imm	Store Word	$MW[rs1 + imm(SE)] \leftarrow rs2$
System	FENCE	Synchronize Thread	No Operation
System	ECALL	Environment Call	MEPC-PC, MSTATUS.mpie-mie ,mie-0, MCAUSE-0x0b, MTVAL-0, PC-MTVEC
System	EBREAK	Environment Break	if DCSR.ebreakm == 1, DCSR.cause-1, DPC-PC, Enter to Debug Mode if DCSR.ebreakm == 0 MEPC-PC, MSTATUS.mpie-mie ,mie-0, MCAUSE-0x03, MTVAL-PC, PC-MTVEC

Table 1.85: RV32I Base Instruction Set Specification

RV32 Zifencei			
Class	Mnemonic	Meaning	Operation
System	FENCE.I	Synchronize Instruction & Data	Clear Pre-Fetch Buffer and Start Fetch

Table 1.86: RV32 Zifencei Specification

RV32 Zicsr			
Class	Mnemonic	Meaning	Operation
CSR Access	CSRRW rd csr, rs1	CSR Atomic Read/Write	$rd \leftarrow csr (ZE), csr \leftarrow rs1$
CSR Access	CSRRS rd csr, rs1	CSR Atomic Read & Set Bit	$rd \leftarrow csr (ZE), csr \leftarrow csr rs1$
CSR Access	CSRRC rd csr, rs1	CSR Atomic Read & Clear Bit	$rd \leftarrow csr (ZE), csr \leftarrow csr \& \sim rs1$
CSR Access	CSRRWI rd, csr, imm	CSR Atomic Read/Write Immediate	$rd \leftarrow csr (ZE), csr \leftarrow imm(ZE)$
CSR Access	CSRRSI rd, csr, imm	CSR Atomic Read & Set Bit Imm	$rd \leftarrow csr (ZE), csr \leftarrow csr imm(ZE)$
CSR Access	CSRRCI rd, csr, imm	CSR Atomic Read & Clear Bit Imm	$rd \leftarrow csr (ZE), csr \leftarrow csr \& \sim imm(ZE)$

Table 1.87: RV32 Zicsr Specification

RV32M Multiply and Divide			
Class	Mnemonic	Meaning	Operation
Multiply	MUL rd, rs1, rs2	Multiply	rd = Lower 32bit of (rs1(S) * rs2(S))
Multiply	MULH rd, rs1, rs2	Multiply Upper Half	rd = Upper 32bit of (rs1(S) * rs2(S))
Multiply	MULHSU rd, rs1, rs2	Multiply Upper Half Signed by Unsigned	rd = Upper 32bit of (rs1(S) * rs2(U))
Multiply	MULHU rd, rs1, rs2	Multiply Upper Half Unsigned by Unsigned	rd = Upper 32bit of (rs1(U) * rs2(U))
Divide	DIV rd, rs1, rs2	Divide	rd = rs1(S) / rs2(S)
Divide	DIVU rd, rs1, rs2	Divide Unsigned	rd = rs1(U) / rs2(U)
Remainder	REM rd, rs1, rs2	Remainder	rd = rs1(S) % rs2(S)
Remainder	REMU rd, rs1, rs2	Remainder Unsigned	rd = rs1(U) % rs2(U)

Table 1.88: RV32M Multiply and Divide Specification

RV32A Atomic Memory Access			
Class	Mnemonic	Meaning	Operation
Load Reserved	LR.W rd, rs1	Load Reserved	rd ← MW[rs1] and Reserve
Store Conditional	SC.W rd, rs1, rs2	Store Conditional	if Reserved Address is still Valid, MW[rs1] ← rs2, rd ← 0x00000000 else rd ← 0xffffffff
Atomic Access	AMOSWAP.W rd, rs1, rs2	Atomic Swap	rd ← MW[rs1], MW[rs1] ← rs2
Atomic Access	AMOADD.W rd, rs1, rs2	Atomic Add	rd ← MW[rs1], MW[rs1] ← rs2 + rd
Atomic Access	AMOXOR.W rd, rs1, rs2	Atomic XOR	rd ← MW[rs1], MW[rs1] ← rs2 ^ rd
Atomic Access	AMOAND.W rd, rs1, rs2	Atomic AND	rd ← MW[rs1], MW[rs1] ← rs2 & rd
Atomic Access	AMOOR.W rd, rs1, rs2	Atomic OR	rd ← MW[rs1], MW[rs1] ← rs2 rd
Atomic Access	AMOMIN.W rd, rs1, rs2	Atomic MIN	rd ← MW[rs1], MW[rs1] ← MIN(rs2, rd) (S)
Atomic Access	AMOMAX.W rd, rs1, rs2	Atomic MAX	rd ← MW[rs1], MW[rs1] ← MAX(rs2, rd) (S)
Atomic Access	AMOMINU.W rd, rs1, rs2	Atomic MIN Unsigned	rd ← MW[rs1], MW[rs1] ← MIN(rs2, rd) (U)
Atomic Access	AMOMAXU.W rd, rs1, rs2	Atomic MAX Unsigned	rd ← MW[rs1], MW[rs1] ← MAX(rs2, rd) (U)

Table 1.89: RV32A Atomic Memory Access Specification

RV32F Single Precision Floating Point Operation			
Class	Mnemonic	Meaning	Operation
F Arithmetic	FADD.S fd, fs1, fs2	Floating Add	fd ← fs1 + fs2
F Arithmetic	FSUB.S fd, fs1, fs2	Floating Subtract	fd ← fs1 - fs2
F Arithmetic	FMUL.S fd, fs1, fs2	Floating Multiply	fd ← fs1 * fs2
F Arithmetic	FDIV.S fd, fs1, fs2	Floating Divide	fd ← fs1 / fs2
F Arithmetic	FSQRT.S fd, fs1	Floating Square Root	fd ← sqrt(fs1)
F Arithmetic	FMADD.S fd, fs1, fs2, fs3	Floating Multiply & Add	fd ← +((fs1 * fs2) + fs3)
F Arithmetic	FMSUB.S fd, fs1, fs2, fs3	Floating Multiply & Subtract	fd ← +((fs1 * fs2) - fs3)
F Arithmetic	FNMADD.S fd, fs1, fs2, fs3	Floating Negative Mult & Add	fd ← -((fs1 * fs2) + fs3)
F Arithmetic	FNMSUB.S fd, fs1, fs2, fs3	Floating Negative Mult & Sub	fd ← -((fs1 * fs2) - fs3)
F Minimum	FMIN.S fd, fs1, fs2	Floating Minimum	fd ← MIN(fs1, fs2)
F Maximum	FMAX.S fd, fs1, fs2	Floating Maximum	fd ← MAX(fs1, fs2)
F Sign	FSGNJ.S fd, fs1, fs2	Floating Sign Injection	fd ← { fs2[31], fs1[30:0] }
F Sign	FSGNJS.N fd, fs1, fs2	Floating Sign Injection Opposite	fd ← { ~fs2[31], fs1[30:0] }
F Sign	FSGNJS.S fd, fs1, fs2	Floating Sign Injection XOR	fd ← { fs1[31] ^ fs2[31], fs1[30:0] }
F Move	FMV.X.W rd, fs1	Floating Move from FRn to XRn	rd ← fs1
F Move	FMV.W.X fd, rs1	Floating Mode from XRn to FRn	fd ← rs1
F Convert	FCVT.W.S rd, fs1	Convert Floating to Integer (S)	rd ← (int32_t)fs1
F Convert	FCVT.W.U.S rd, fs1	Convert Floating to Integer (U)	rd ← (uint32_t)fs1
F Convert	FCVT.S.W fd, rs1	Convert Integer to Floating (S)	fd ← (float)rs1 (U)
F Convert	FCVT.S.WU fd, rs1	Convert Integer to Floating (U)	fd ← (float)rs1 (S)
F Compare	FEQ.S rd, fs1, fs2	Floating Compare Equal	rd ← (fs1 == fs2)? 1 : 0
F Compare	FLT.S rd, fs1, fs2	Floating Compare Less Than	rd ← (fs1 < fs2)? 1 : 0
F Compare	FLE.S rd, fs1, fs2	Floating Compare Less or Equal	rd ← (fs1 <= fs2)? 1 : 0
F Classify	FCLASS.S rd, fs1	Floating Classify	rd ← Classify(fs1)
F Load	FLW fd, rs1, imm	Floating Load	fd ← MW[rs1 + imm(SE)]
F Store	FSW rs1, fs2, imm	Floating Store	MW[rs1 + imm(SE)] ← fs2

Table 1.90: RV32F Single Precision Floating Point Operation Specification

RV32 Privileged Instruction			
Class	Mnemonic	Meaning	Operation
Privileged	MRET	Trap-Return	MSTATUS.mie←mpie, PC←MEPC
Privileged	WFI	Wait for Interrupt	Wait for Interrupt if unmasked, invoke exception if masked, execute next instruction

Table 1.91: RV32 Privileged Instruction Specification

The WFI (Wait for Interrupt) Instruction waits for interrupt requests. If the received interrupt is enabled, an exception corresponding to the interrupt starts. If the received interrupt is not enabled (masked by mie = 0), the CPU resumes and starts from the next instruction after WFI.

1.26 16bit ISA Specifications

RV32C Compressed Instruction			
Class	Mnemonic	Meaning	Operation
C Arithmetic	C.LI rd, imm	Load Immediate	$rd \leftarrow \text{imm(SE)}$
C Arithmetic	C.LUI rd, imm	Load Upper Immediate	$rd \leftarrow \{\text{imm(SE)}, 12'b0\}$
C Arithmetic	C.ADD rd, rs2	Add	$rd \leftarrow Rd + rs2$
C Arithmetic	C.SUB rd', rs2'	Subtract	$rd' \leftarrow rd' - rs2'$
C Arithmetic	C.ADDI rd, imm	Add Immediate	$rd \leftarrow rd + \text{imm(SE)}$
C Arithmetic	C.ADDI4SPN rd', uzuimm	Add SP and Imm * 4	$rd' \leftarrow x2 + \text{imm(ZE)} * 4$
C Arithmetic	C.ADDI16SP sp, sp, imm	Add SP and Imm * 16	$sp(x2) \leftarrow xp(x2) + \text{imm(SE)}$
C Shift	C.SLLI rd, shamt	Shift Left Logical Immediate	$rd' \leftarrow rd' \ll \text{shamt Logical}$
C Shift	C.SRLI rd', shamt	Shift Right Logical Immediate	$rd' \leftarrow rd' \gg \text{shamt Logical}$
C Shift	C.SRAI rd', shamt	Shift Right Arithmetic Immediate	$rd' \leftarrow rd' \gg \text{shamt Arithmetic}$
C Logical	C.XOR rd', rs2'	XOR	$rd' \leftarrow rd' \wedge rs2'$
C Logical	C.OR rd', rs2'	OR	$rd' \leftarrow rd' \mid rs2'$
C Logical	C.AND rd', rs2'	AND	$rd' \leftarrow rd' \& \text{imm(SE)}$
C Logical	C.ANDI rd', imm	AND Immediate	$rd' \leftarrow rd' \& \text{imm(SE)}$
C Move	C.MV rd, rs2	Move	$rd \leftarrow rs2$
C Branch	C.BEQZ rs1', imm	Branch Equal Zero	$PC \leftarrow PC + \text{imm(SE)} \text{ if } rs1' == 0$
C Branch	C.BNEZ rs1', imm	Branch Equal Not Zero	$PC \leftarrow PC + \text{imm(SE)} \text{ if } rs1' != 0$
C Jump	C.J imm	Jump	$PC \leftarrow PC + \text{imm(SE)}$
C Jump	C.JR rs1	Jump Register	$PC \leftarrow rs1$
C Jump & Link	C.JAL imm	Jump and Link	$x1 \leftarrow PC+2, PC \leftarrow PC + \text{imm(SE)}$
C Jump & Link	C.JALR rs1	Jump Register and Link	$x1 \leftarrow PC+2, PC \leftarrow rs1$
C Load	C.LW rd', rs1', imm	Load Word	$rd' \leftarrow MW[rs1' + \text{imm(ZE)} * 4]$
C Load	C.LWSP rd, imm	Load Word Stack Pointer	$rd \leftarrow MW[x2 + \text{imm(ZE)} * 4]$
C Store	C.SW rs1', rs2', imm	Store Word	$MW[rs1' + \text{imm(ZE)} * 4] \leftarrow rs2'$
C Store	C.SWSP rs2, imm	Store Word Stack Pointer	$MW[x2 + \text{imm(ZE)} * 4] \leftarrow rs2$
C System	C.EBREAK	Environment Break	if DCSR.ebreakm == 1, DCSR.cause-1, DPC-PC, Enter to Debug Mode if DCSR.ebreakm == 0 MEPC-PC, MSTATUS.mpie-mie ,mie-0, MCAUSE-0x03, MTVAL-PC, PC-MTVEC
C NOP	C.NOP	No Operation	No Operation
C Illegal	ILLEGAL	Illegal Instruction	MEPC-PC, MSTATUS.mpie-mie ,mie-0, MCAUSE-0x02, MTVAL-0, PC-MTVEC

Table 1.92: RV32C Compressed Instruction Specification

RV32FC Floating Point Compressed Instruction			
Class	Mnemonic	Meaning	Operation
FC Load	C.FLW fd', rs1', imm	Load Floating	$fd' \leftarrow MW[rs1' + \text{imm(ZE)} * 4]$
FC Load	C.FLWSP fd, imm	Load Floating Stack Pointer	$fd \leftarrow MW[x2 + \text{imm(ZE)} * 4]$
FC Store	C.FSW rs1', fs2', imm	Store Floating	$MW[rs1' + \text{imm(ZE)} * 4] \leftarrow fs2'$
FC Store	C.FSWSP fs2, imm	Store Floating Stack Pointer	$MW[x2 + \text{imm(ZE)} * 4] \leftarrow fs2$

Table 1.93: RV32FC Floating Point Compressed Instruction Specification

1.27 32bit ISA Code Assignments

RV32I Base Instruction Set															
Class	Mnemonic	31	27	26	25	24	20	19	15	14	12	11	7	6	0
Arithmetic	LUI rd, imm							imm[31:12]				rd		0110111	
Arithmetic	AUIPC rd, imm							imm[31:12]				rd		0010011	
Arithmetic	ADD rd, rs1, rs2				0000000		rs2		rs1	000		rd		0110011	
Arithmetic	SUB rd, rs1, rs2				0100000		rs2		rs1	000		rd		0110011	
Arithmetic	ADDI rd, rs1, imm					imm[11:0]			rs1	000		rd		0010011	
Shift	SLL rd, rs1, rs2				0000000		rs2		rs1	001		rd		0110011	
Shift	SRL rd, rs1, rs2				0000000		rs2		rs1	101		rd		0110011	
Shift	SRA rd, rs1, rs2				0100000		rs2		rs1	101		rd		0110011	
Shift	SLLI rd, rs1, shamt				0000000		shamt		rs1	001		rd		0010011	
Shift	SRLI rd, rs1, shamt				0000000		shamt		rs1	101		rd		0010011	
Shift	SRAI rd, rs1, shamt				0100000		shamt		rs1	101		rd		0010011	
Logical	XOR rd, rs1, rs2				0000000		rs2		rs1	100		rd		0110011	
Logical	OR rd, rs1, rs2				0000000		rs2		rs1	110		rd		0110011	
Logical	AND rd, rs1, rs2				0000000		rs2		rs1	111		rd		0110011	
Logical	XORI rd, rs1, imm					imm[11:0]			rs1	100		rd		0010011	
Logical	ORI rd, rs1, imm					imm[11:0]			rs1	110		rd		0010011	
Logical	ANDI rd, rs1, imm					imm[11:0]			rs1	111		rd		0010011	
Compare	SLT rd, rs1, rs2				0000000		rs2		rs1	010		rd		0110011	
Compare	SLTU rd, rs1, rs2				0000000		rs2		rs1	011		rd		0110011	
Compare	SLTI rd, rs1, imm					imm[11:0]			rs1	010		rd		0010011	
Compare	SLTIU rd, rs1, imm					imm[11:0]			rs1	011		rd		0010011	
Branch	BEQ rs1, rs2, imm				imm[12 10:5]		rs2		rs1	000	imm[4:1 11]		1100011		
Branch	BNE rs1, rs2, imm				imm[12 10:5]		rs2		rs1	001	imm[4:1 11]		1100011		
Branch	BLT rs1, rs2, imm				imm[12 10:5]		rs2		rs1	100	imm[4:1 11]		1100011		
Branch	BGE rs1, rs2, imm				imm[12 10:5]		rs2		rs1	101	imm[4:1 11]		1100011		
Branch	BLTU rs1, rs2, imm				imm[12 10:5]		rs2		rs1	110	imm[4:1 11]		1100011		
Branch	BGEU rs1, rs2, imm				imm[12 10:5]		rs2		rs1	111	imm[4:1 11]		1100011		
Jump & Link	JAL rd, imm					imm[20 10:1 11 19:12]					rd		1101111		
Jump & Link	JALR rd, imm					imm[11:0]			rs1	000	rd		1100111		
Load	LB rd, rs1, imm					imm[11:0]			rs1	000	rd		0000011		
Load	LH rd, rs1, imm					imm[11:0]			rs1	001	rd		0000011		
Load	LW rd, rs1, imm					imm[11:0]			rs1	010	rd		0000011		
Load	LBU rd, rs1, imm					imm[11:0]			rs1	100	rd		0000011		
Load	LHU rd, rs1, imm					imm[11:0]			rs1	101	rd		0000011		
Store	SB rs1, rs2, imm				imm[11:5]		rs2		rs1	000	imm[4:0]		0100011		
Store	SH rs1, rs2, imm				imm[11:5]		rs2		rs1	001	imm[4:0]		0100011		
Store	SW rs1, rs2, imm				imm[11:5]		rs2		rs1	010	imm[4:0]		0100011		
Synchronization	FENCE				fm[3:0] pred[3:0] succ[3:0]				rs1	000	rd		0001111		
System	ECALL				0000000000000				00000	000	00000		1110011		
System	EBREAK				0000000000001				00000	000	00000		1110011		

Table 1.94: RV32I Base Instruction Set Code

RV32 Zifencei															
Class	Mnemonic	31	27	26	25	24	20	19	15	14	12	11	7	6	0
System	FENCE.I					imm[11:0]			rs1	001	rd			0001111	

Table 1.95: RV32 Zifencei Code

RV32 Zicsr															
Class	Mnemonic	31	27	26	25	24	20	19	15	14	12	11	7	6	0
CSR Access	CSRRW rd csr, rs1					csr			rs1	001	rd			1110011	
CSR Access	CSRRS rd csr, rs1					csr			rs1	010	rd			1110011	
CSR Access	CSRRC rd csr, rs1					csr			rs1	011	rd			1110011	
CSR Access	CSRRIWI rd, csr, imm					csr			uimm	101	rd			1110011	
CSR Access	CSRRSI rd, csr, imm					csr			uimm	110	rd			1110011	
CSR Access	CSRRCI rd, csr, imm					csr			uimm	111	rd			1110011	

Table 1.96: RV32 Zicsr Code

RV32M Multiply and Divide															
Class	Mnemonic	31	27	26	25	24	20	19	15	14	12	11	7	6	0
Multiply	MUL rd, rs1, rs2			0000001		rs2		rs1	000			rd		0110011	
Multiply	MULH rd, rs1, rs2			0000001		rs2		rs1	001			rd		0110011	
Multiply	MULHSU rd, rs1, rs2			0000001		rs2		rs1	010			rd		0110011	
Multiply	MULHU rd, rs1, rs2			0000001		rs2		rs1	011			rd		0110011	
Divide	DIV rd, rs1, rs2			0000001		rs2		rs1	100			rd		0110011	
Divide	DIVU rd, rs1, rs2			0000001		rs2		rs1	101			rd		0110011	
Remainder	REM rd, rs1, rs2			0000001		rs2		rs1	110			rd		0110011	
Remainder	REMU rd, rs1, rs2			0000001		rs2		rs1	111			rd		0110011	

Table 1.97: RV32M Multiply and Divide Code

RV32A Atomic Memory Access															
Class	Mnemonic	31	27	26	25	24	20	19	15	14	12	11	7	6	0
Load Reserved	LR.W rd, rs1		00010	aq	rl	00000		rs1	010			rd		0101111	
Store Conditional	SC.W rd, rs1, rs2		00011	aq	rl		rs2		rs1	010			rd		0101111
Atomic Access	AMOSWAP.W rd, rs1, rs2		00001	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOADD.W rd, rs1, rs2		00000	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOXOR.W rd, rs1, rs2		00100	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOAND.W rd, rs1, rs2		01100	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOOR.W rd, rs1, rs2		01000	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOMIN.W rd, rs1, rs2		10000	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOMAX.W rd, rs1, rs2		10100	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOMINU.W rd, rs1, rs2		11000	aq	rl	rs2		rs1	010			rd		0101111	
Atomic Access	AMOMAXU.W rd, rs1, rs2		11100	aq	rl	rs2		rs1	010			rd		0101111	

Table 1.98: RV32A Atomic Memory Access Code

RV32F Single Precision Floating Point Operation															
Class	Mnemonic	31	27	26	25	24	20	19	15	14	12	11	7	6	0
F Arithmetic	FADD.S fd, fs1, fs2			0000000		fs2		fs1	rm			fd		1010011	
F Arithmetic	FSUB.S fd, fs1, fs2			0000100		fs2		fs1	rm			fd		1010011	
F Arithmetic	FMUL.S fd, fs1, fs2			0001000		fs2		fs1	rm			fd		1010011	
F Arithmetic	FDIV.S fd, fs1, fs2			0001100		fs2		fs1	rm			fd		1010011	
F Arithmetic	FSQRT.S fd, fs1			0101100		00000		fs1	rm			fd		1010011	
F Arithmetic	FMADD.S fd, fs1, fs2, f	fs3	0	0		fs2		fs1	rm			fd		1000011	
F Arithmetic	FMSUB.S fd, fs1, fs2, f	fs3	0	0		fs2		fs1	rm			fd		1000011	
F Arithmetic	FNMADD.S fd, fs1, fs2, f	fs3	0	0		fs2		fs1	rm			fd		1001111	
F Arithmetic	FNMSUB.S fd, fs1, fs2, f	fs3	0	0		fs2		fs1	rm			fd		1001011	
F Minimum	FMIN.S fd, fs1, fs2		0010100			fs2		fs1	000			fd		1010011	
F Maximum	FMAX.S fd, fs1, fs2		0010100			fs2		fs1	001			fd		1010011	
F Sign	FSGNJ.S fd, fs1, fs2		0010000			fs2		fs1	000			fd		1010011	
F Sign	FSGNJS.S fd, fs1, fs2		0010000			fs2		fs1	001			fd		1010011	
F Move	FMV.X.W rd, fs1		1110000			00000		fs1	000			rd		1010011	
F Move	FMV.W.X fd, rs1		1111000			00000		rs1	000			fd		1010011	
F Convert	FCVT.W.S rd, fs1		1100000			00000		fs1	rm			rd		1010011	
F Convert	FCVT.WU.S rd, fs1		1100000			00001		fs1	rm			rd		1010011	
F Convert	FCVT.S.W fd, rs1		1101000			00000		rs1	rm			fd		1010011	
F Convert	FCVT.S.WU fd, rs1		1101000			00001		rs1	rm			fd		1010011	
F Compare	FEQ.S rd, fs1, fs2		1010000			fs2		fs1	010			rd		1010011	
F Compare	FLT.S rd, fs1, fs2		1010000			fs2		fs1	001			rd		1010011	
F Compare	FLE.S rd, fs1, fs2		1010000			fs2		fs1	000			rd		1010011	
F Classify	FCLASS.S rd, fs1		1110000			00000		fs1	001			rd		1010011	
F Load	FLW fd, rs1, imm			imm[11:0]				rs1	010			fd		0000111	
F Store	FSW rs1, fs2, imm			imm[11:5]				fs2	rs1	010		imm[4:0]		0100111	

Table 1.99: RV32F Single Precision Floating Point Operation Code

RV32 Privileged Instruction															
Class	Mnemonic	31	27	26	25	24	20	19	15	14	12	11	7	6	0
Privileged	MRET			0011000		00010		00000	000			00000		1110011	
Privileged	WFI			0001000		00101		00000	000			00000		1110011	

Table 1.100: RV32 Privileged Instruction Code

1.28 16bit ISA Code Assignments

RV32C Compressed Instruction																		
Class	Mnemonic	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Note
C Arithmetic	C.LI rd, imm	010			imm[5]			rd!=0			imm[4:0]	01	HINT:rd=0					
C Arithmetic	C.LUI rd, imm	011			nzimm[17]			rd!={0,2}			nzimm[16:12]	01	RES:nzimm=0; HINT:rd=0					
C Arithmetic	C.ADD rd, rs2	100			1			rd!=0			rs2!=0	10	HINT:rd=0					
C Arithmetic	C.SUB rd', rs2'	100			0			rd'			00	rs2'	01					
C Arithmetic	C.ADDI rd, imm	000			nzimm[5]			rs1/rd!=0			nzimm[4:0]	01	HINT:nzimm=0					
C Arithmetic	C.ADDI4SPN rd', uzuimm	000						nzuumm[5:4 9:6 2 3]			rd'	00	RES:nzuumm=0					
C Arithmetic	C.ADDI16SP sp, sp, imm	011			nzimm[9]			2			nzimm[4 6 8:7 5]	01	HINT:nzimm=0					
C Shift	C.SLLI rd, shamt	000			shamt[5]			rd!=0			shamt[4:0]	10	HINT:rd=0; NSE:shamt[5]=1					
C Shift	C.SRLI rd', shamt	100			shamt[5]	00		rd'			shamt[4:0]	01	NSE:shamt[5]=1					
C Shift	C.SRAI rd', shamt	100			shamt[5]	01		rd'			shamt[4:0]	01	NSE:shamt[5]=1					
C Logical	C.XOR rd', rs2'	100			0			rd'			01	rs2'	01					
C Logical	C.OR rd', rs2'	100			0			rd'			10	rs2'	01					
C Logical	C.AND rd', rs2'	100			0			rd'			11	rs2'	01					
C Logical	C.ANDI rd', imm	100			imm[5]			rd'			imm[4:0]	01						
C Move	C.MV rd, rs2	100			0			rd!=0			rs2!=0	10	HINT:rd=0					
C Branch	C.BEQZ rs1', imm	110			imm[8 4:3]			rs1'			imm[7:6 2:1 5]	01						
C Branch	C.BNEZ rs1', imm	111			imm[8 4:3]			rs1'			imm[7:6 2:1 5]	01						
C Jump	C.J imm	101						imm[11 4 9:8 10 6 7 3:1 5]				01						
C Jump	C.JR rs1	100			0			rs1!=0			0	10	RES:rs1=0					
C Jump & Link	C.JAL imm	001						imm[11 4 9:8 10 6 7 3:1 5]				01						
C Jump & Link	C.JALR rs1	100			1			rs1!=0			0	10						
C Load	C.LW rd', rs1', imm	010			uimm[5:3]			rs1'			uimm[2 6]	rd'	00					
C Load	C.LWSP rd, imm	010			uimm[5]			rd!=0			uimm[4:2 7:6]	10	RES:rd=0					
C Store	C.SW rs1', rs2', imm	110			uimm[5:3]			rs1'			uimm[2 6]	rs2'	00					
C Store	C.SWSP rs2, imm	110						uimm[5:2 7:6]			rs2	10						
C System	C.EBREAK	100			1			0			0	10						
C NOP	C.NOP	000			nzimm[5]			00000			nzimm[4:0]	01	HINT:nzimm!=0					
C Illegal	ILLEGAL	000						00000000			000	00						

Table 1.101: RV32C Compressed Instruction Code

RV32FC Floating Point Compressed Instruction																		
Class	Mnemonic	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Note
FC Load	C.FLW fd', rs1', imm	011			uimm[5:3]			rs1'			uimm[2 6]	fd'	00					
FC Load	C.FLWSP fd, imm	011			uimm[5]			fd			uimm[4:2 7:6]	10						
FC Store	C.FSW rs1', fs2', imm	111			uimm[5:3]			rs1'			uimm[2 6]	fs2'	00					
FC Store	C.FSWSP fs2, imm	111						uimm[5:2 7:6]			fs2	10						

Table 1.102: RV32FC Floating Point Compressed Instruction Code

1.29 JTAG Data Register for User

In the JTAG TAP controller of mmRISC-1, a 32bit USER register is added to the standard specification as shown in Table 1.103. The 5bit address of the USER register is 0x0f which is specified by the IR register. As shown in Figure 1.13, if the USER register is selected, the TAP controller captures 32bit width input signal JTAG_DR_USER_IN into the JTAG shift chain at CAPTURE_DR state, and updates the USER register by copying from the JTAG shift chain at UPDATE_DR state. The contents of the USER register are connected to 32bit width output signal JTAG_DR_USER_OUT.

Address	Name	Description
0x00	BYPASS	JTAG recommends this encoding
0x01	IDCODE	JTAG recommends this encoding
0x0F	USER	For User (mmRISC-1 original)
0x10	DTMCS	For Debugging
0x11	DMI	For Debugging
0x12	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x13	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x14	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x15	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x16	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x17	Reserved (BYPASS)	Reserved for future RISC-V debugging
0x1F	BYPASS	JTAG requires this encoding

Table 1.103: JTAG Data Register for User

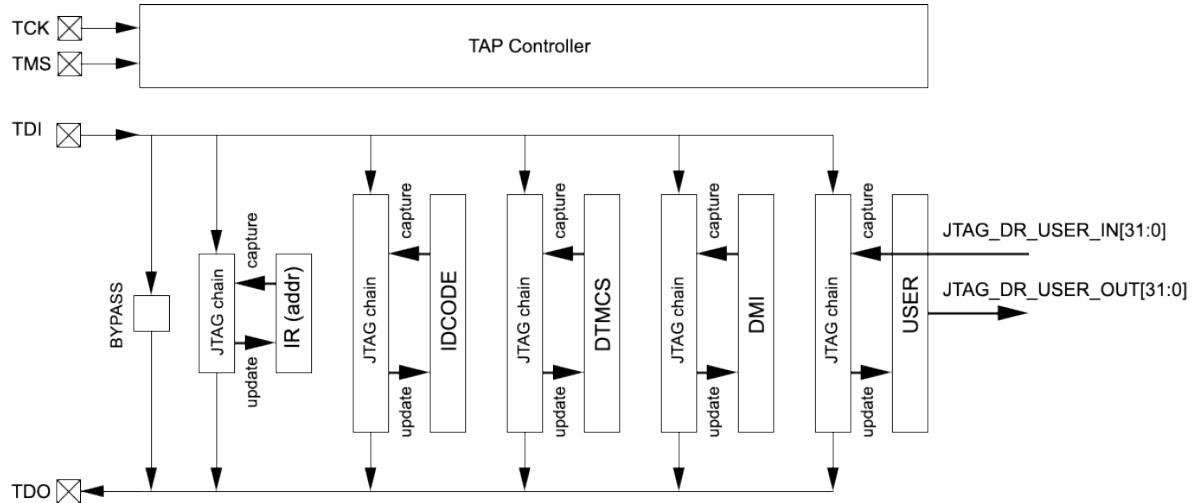


Figure 1.13: Block Diagram of JTAG Chain in mmRISC-1

1.30 Halt on Reset

The standard debug specification of RISC-V has Halt-on-Reset feature, and mmRISC-1 supports the standard spec, of course. The feature is controlled by setresethaltreq and clrresethaltreq in DMCONTROL register in DM (Debug Module). Once the Halt-on-Reset is enabled, the CPU always halts before first instruction execution when CPU reset (by RES_CPU), unless the feature is disabled by JTAG access or power-on-reset (RES_ORG).

Please note that the standard Halt-on-Reset feature is disabled when a power-on-reset (RES_ORG) occurs. This means the CPU starts its running on a power-on-reset, and keeps its running until next CPU reset after enabling Halt-on-Reset via JTAG access. Even if you want to prevent CPU execution just after power-on-reset, for example, in order to erase and write embedded non-volatile memory or in order to download program code to internal RAM, the standard specification can not do that. So, the mmRISC-1 has an alternative Halt-on-Reset feature in addition.

Figure 1.14 shows the feature in mmRISC-1. When there is a negation of RES_ORG, if an input signal FORCE_HALTORESET_REQ is low, the CPU starts from 1st instruction at reset vector, which is normal operation.

On the other hand, when there is a negation of RES_ORG, if the input signal FORCE_HALTORESET_REQ is high, the CPU halts until receiving a resume command from debug I/F. In the case, FORCE_HALTORESET_ACK will be asserted to indicate the request is acknowledged. After the acknowledgement, external circuit which requires CPU halt should negate FORCE_HALTORESET_REQ immediately.

You can use this feature for both JTAG debug I/F and cJTAG debug I/F.

As for 4-wire JTAG I/F, FORCE_HALTORESET_REQ should be properly created according to external request. Figure 1.15 might be a good example. When there is a negation of RES_ORG, if RESET_HALT_N(active-low) is driven at low level, the CPU goes to halt state, and resumes from reset vector after resume command is sent via debug port.

As for 2-wire cJTAG I/F, a conversion block CJTAG_2_JTAG.v supports the feature, that is, when there is a negation of RES_ORG, if TMSC is driven to low level, CPU halts until receiving resume command from debugger. Details are described in another section.

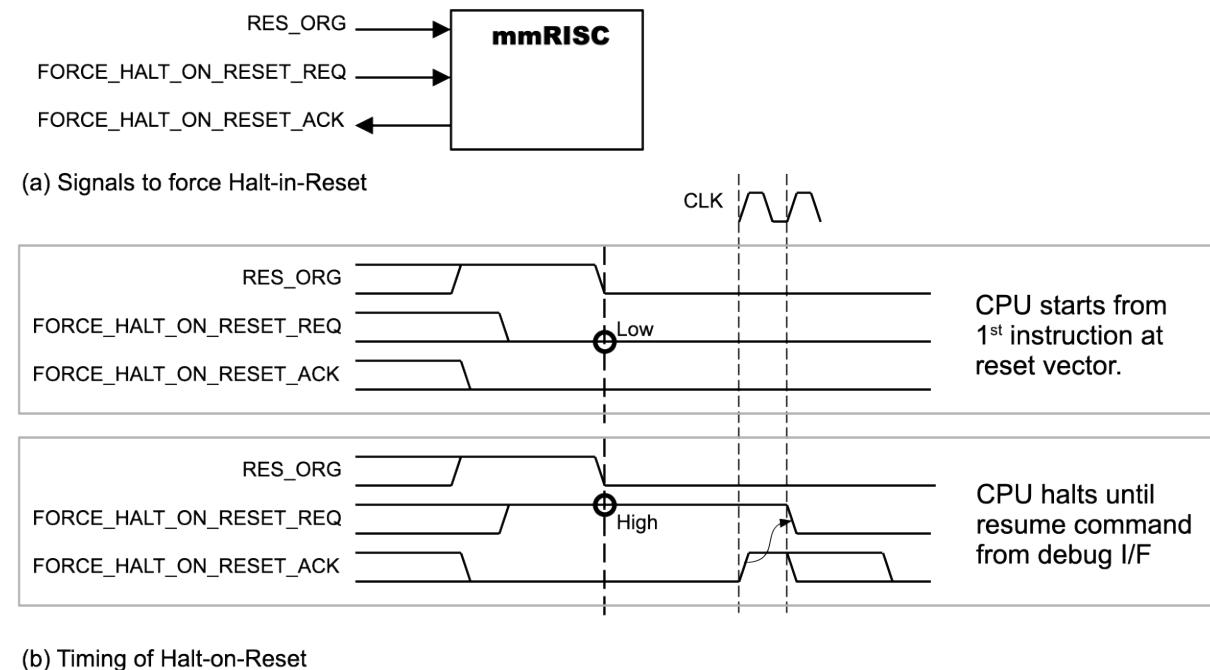


Figure 1.14: Halt on Reset

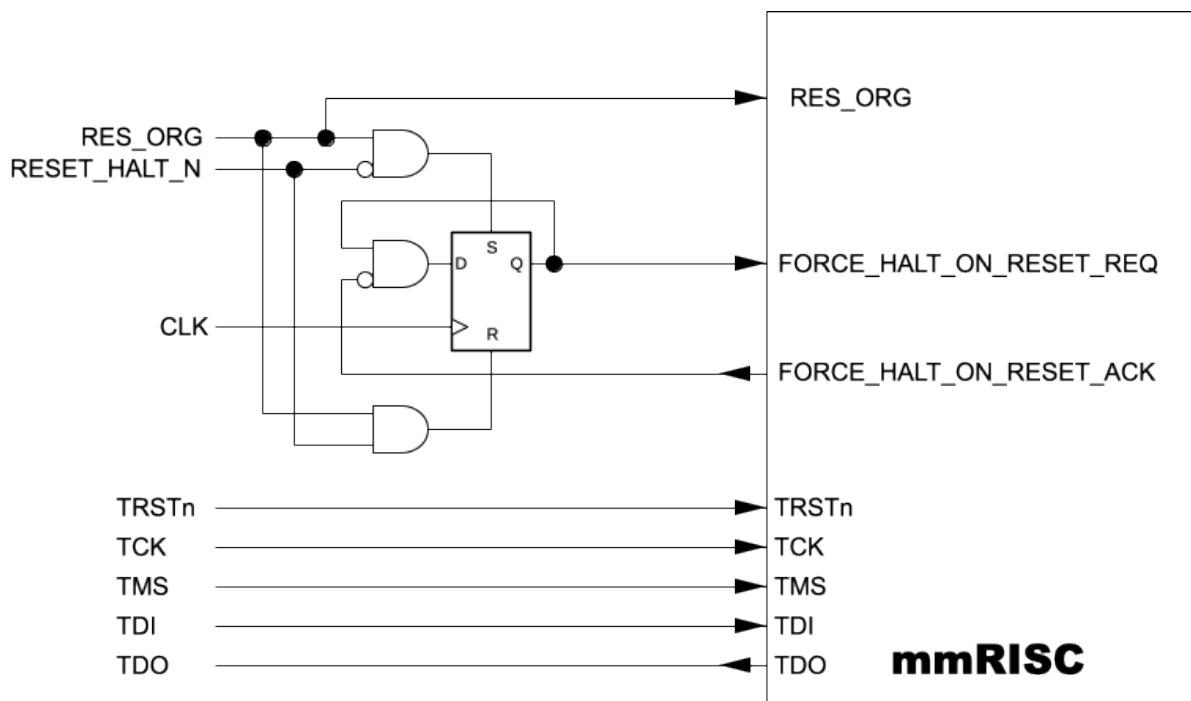


Figure 1.15: Example of Control Logic for Halt on Reset

1.31 cJTAG (Compact JTAG)

While the debug interface of mmRISC-1 core is essentially 4-wire JTAG, the mmRISC-1 project supports 2-wire cJTAG (Compact JTAG) as its debug interface instead of 4-wire JTAG. This project provides a module CJTAG_2_JTAG to convert 2-wire cJTAG to 4-wire JTAG which is located at outside of mmRISC-1 core boundary, that is at chip top layer. If you want to use cJTAG, please enable `define ENABLE_CJTAG switch in common/defines_chip.v.

The cJTAG interface consists of only TCKC and TMSC as shown in Table 1.104.

The relationship between the frequency of the system clock and TCKC does not matter.

For example, both cases with fCLK=20MHz/fTCK=10MHz and fCLK=100KHz/fTCK=10MHz are OK.

Name	Meaning	Description
TCKC	Test Clock for cJTAG	Clock which is driven by external Adapter.
TMSC	Test Mode Select for cJTAG	Bi-directional Serial Data

Table 1.104: Signal of cJTAG Interface

1.31.1 cJTAG Signal Sequence

The cJTAG signal sequence is divided into 3 phases shown below.

Escape Sequence

After the reset, the CJTAG_2_JTAG is in Offline state. During TCKC is held high, if TMSC has rising edges (L→H) 3 times or more, the CJTAG_2_JTAG detects escape sequence and enters Online state as shown in Figure 1.16(a).

Activation Sequence

After the Online Sequence, the CJTAG_2_JTAG waits for the Activation Sequence as shown in Figure 1.16(b). During this sequence, TMSC is driven by external adapter, and the CJTAG_2_JTAG samples TMSC at the rising edge of TCKC. If it detects the Activation Sequence for OSCAN1, which consists of (1) OAC (Online Activation Code) = 4'b0011 (MSB first), (2) EC (Extension Code) = 4'b0001, and (3) CP (Check Packet) = 4'b0000, it enters OSCAN1 mode. If it detects any other sequence, it returns to Offline state.

OSCAN1 Sequence

In OSCAN1 mode, the CJTAG_2_JTAG starts to convert from cJTAG to JTAG as its essential function. The JTAG source signals TMS, TDI and TDO are multiplexed onto the TMSC signal, and three TCKC rising edges are required to perform one bit of JTAG scan as shown in Figure 1.17. At the first rising edge of TCKC, TMSC is driven by an external adapter and conveys the inverse of TDI. At the second rising edge of TCKC, TMSC is driven by an external adapter and conveys TMS. And at the third rising edge of TCKC, TMSC is driven by the CJTAG_2_JTAG and conveys TDO.

At the JTAG port of the CJTAG_2_JTAG, TCK toggles only at the TDO phase of cJTAG. Before the TCK toggle, TDI and TMS are made according to the TMSC level of cJTAG at the first and second edges of TCKC, and TDO is loaded on TMSC at the TDO phase of cJTAG. Note that the TDO is made at the previous falling edge

of TCK. At last, at the rising edge of TCK, the JTAG side one bit scan operation is performed.

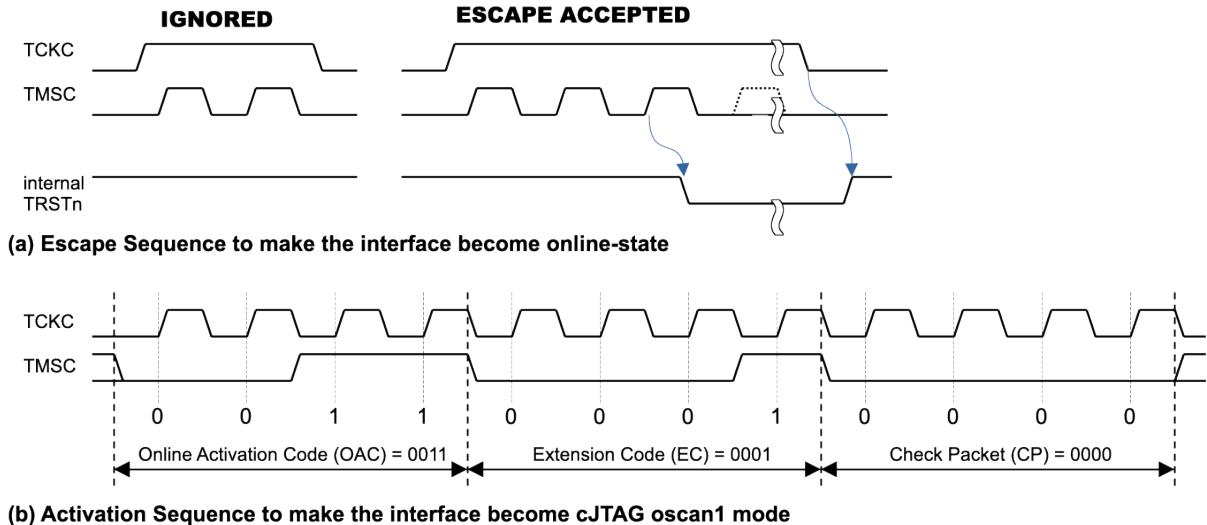


Figure 1.16: cJTAG Escape Sequence and Activation Sequence

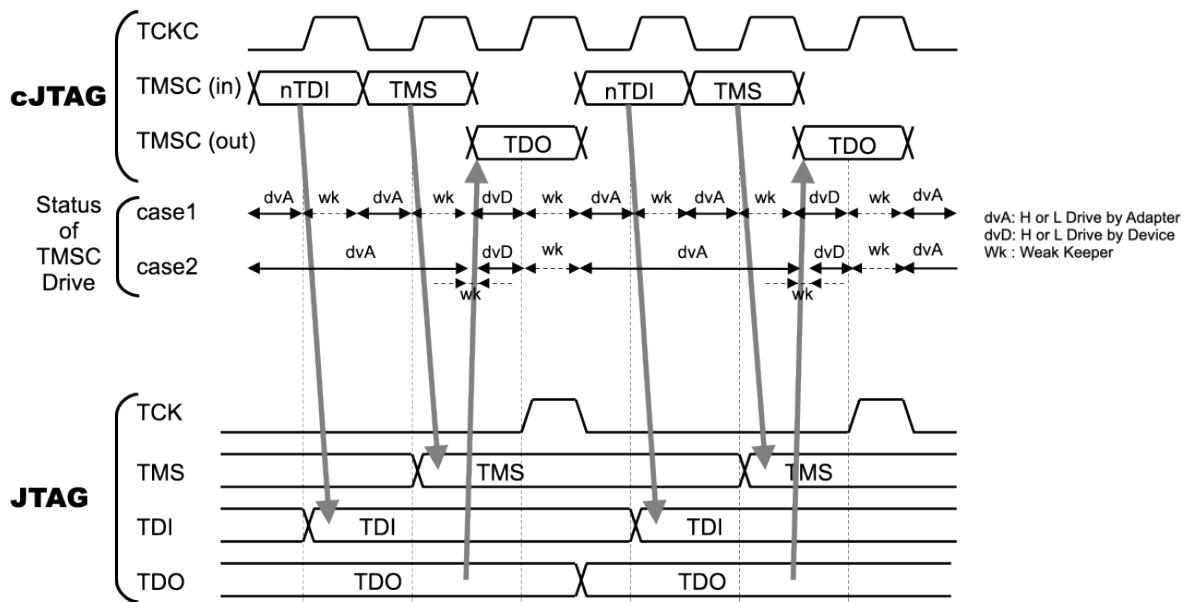


Figure 1.17: cJTAG OSCAN1 Sequence

1.31.2 cJTAG TCKC/TMSC Pin Controls

To avoid a drive conflict in OSCAN1 Sequence, TMSC is driven by its source only while TCKC is low, and relies on a Bus Keeper logic to maintain a valid logic level while TCKC is high. Moreover, when the external adapter wants to start the escape sequence suddenly during the OSCAN1 sequence, especially during the CJTAG_2_JTAG is driving TMSC as TDO information, the CJTAG_2_JTAG should release TMSC by watching if the TCKC level is high.

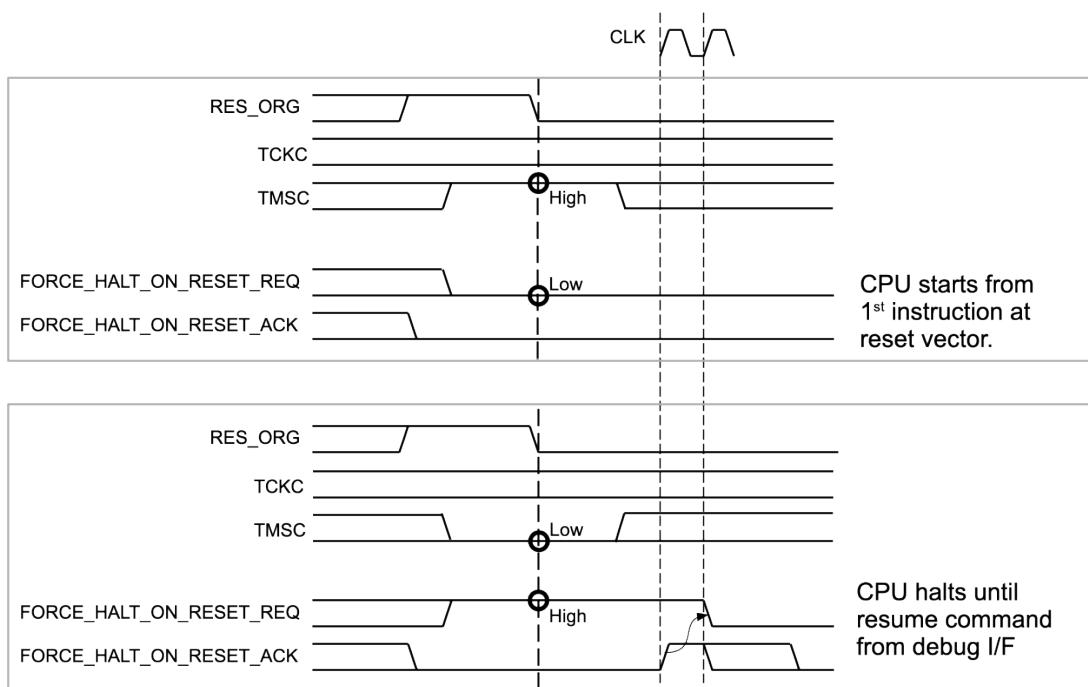
As a result, the CJTAG_2_JTAG controls cJTAG pins as shown in Table 1.105 and the chip top block and I/O buffer block should follow this specification.

Signal	Reset	Escape Sequence	Activation Sequence	OSCAN1 Sequence
TCKC	Pull Up	Pull Up	Pull Up	Pull Up
TMSC	Pull Up	Pull Up	Pull Up	Bus Keeper

Table 1.105: cJTAG Pin Control

1.31.3 cJTAG Halt on Reset

The CJTAG_2_JTAG supports the Halt-on-Reset function as well as the mmRISC-1 core does. As shown in Figure 1.18, when the negation of RES_ORG occurs, if TMSC is driven at a high level, the CJTAG_2_JTAG does not assert FORCE_HALT_ON_RESET_REQ, and the CPU starts from the 1st instruction at reset vector, which is the normal operation. If TMSC is driven to a low level, the CJTAG_2_JTAG asserts FORCE_HALT_ON_RESET_REQ, and negates FORCE_HALT_ON_RESET_ACK after the assertion of FORCE_HALT_ON_RESET_ACK from the mmRISC core. Then the CPU halts before executing the 1st instruction at the reset vector until a resume command comes from debugger. During the time when the reset input is asserted, the TMSC should be pulled-up at the chip level as shown in Table 1.105, so if TMSC is left open, the CPU starts execution normally.

**Figure 1.18: cJTAG Halt on Reset**

1.31.4 cJTAG Adapter (FTDI)

In general, the interface adapter from USB to 4-wire JTAG for OpenOCD controlled by an FTDI chip is very popular. But if it is difficult for you to get an adapter from USB to 2-wire cJTAG which operates under the OpenOCD environment, you can make it by attaching a simple circuit (`cjtag_adapter.v`) to the adapter from USB to 4-wire JTAG as shown in Figure 1.19. An example of a Verilog HDL code of the simple circuit (`cjtag_adapter.v`) is shown in Listing 1.4.

This simple circuit supports the Halt-on-Reset feature. If RESET_HALT_N (negated

tive) is asserted to a low level at the negation of the reset input, TMSC goes to a low level to halt the CPU after reset. If you do not use the Halt-on-Reset feature of mmRISC-1, fix RESET_HALT_N to High.

Please note that this simple adapter does not follow the cJTAG standard because TMSC does not become HiZ during TCKC is high. But this adapter works fine even when connecting to cJTAG standard devices. During the OSCAN1 sequence of the cJTAG protocol, even if the adapter suddenly begins the escape sequence and the activation sequence, it drives TCKC to high and drives TMSC to high or low, there are no contention on TMSC signal because the target device does not drive TMSC due to TCKC's high level.

On the other hand, this adapter does not release TMSC when TCKC is high, which is a worry if there might be contention on TMSC line. As for the OpenOCD of RISC-V, during the TDO phase in the OSCAN1 sequence when the target device drives TMSC, the falling edge of TCK and the negating edge of TMS, both from JTAG adapter, are at the same time, which means TMSC becomes HiZ just at the falling edge of TCK. But the CJTAG_2_JTAG in the target device drives TMSC (as TDO) several clocks after the TCKC falling edge, and after the rising edge of TCKC (during TCKC is high), the target device releases TMSC, so this system can avoid signal contentions on TMSC line.

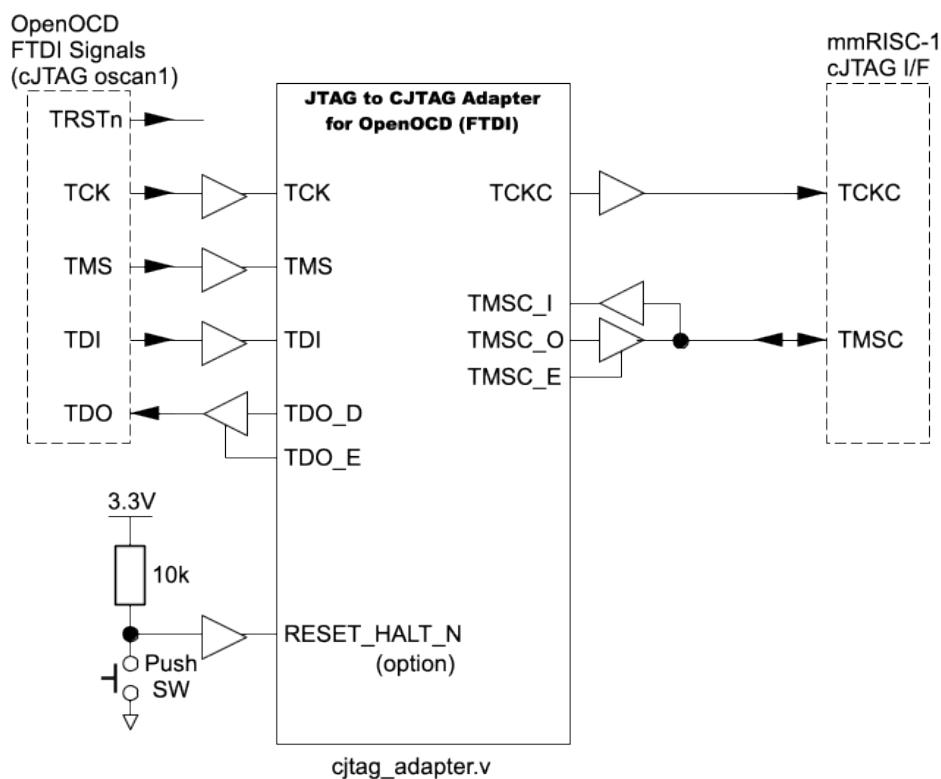


Figure 1.19: External Adapter to convert JTAG to cJTAG

```
// TCKC
// if Push SW is ON, Force High for target not to drive TMSC
// else Through TCK
assign TCKC = (~RESET_HALT_N)? 1'b1
           : TCK;
// TMSC
// if Push SW is ON, Force Low to halt CPU
// else TMS determines output enable,
//       and TDI determines output level
assign TMSC_O = (~RESET_HALT_N)? 1'b0
```

```

        : TDI;
assign TMSC_E = (~RESET_HALT_N)? 1'b1
        : (TMS          )? 1'b1
        :                   1'b0;
// TDO
assign TDO_D  = TMSC_I;
assign TDO_E  = 1'b1;

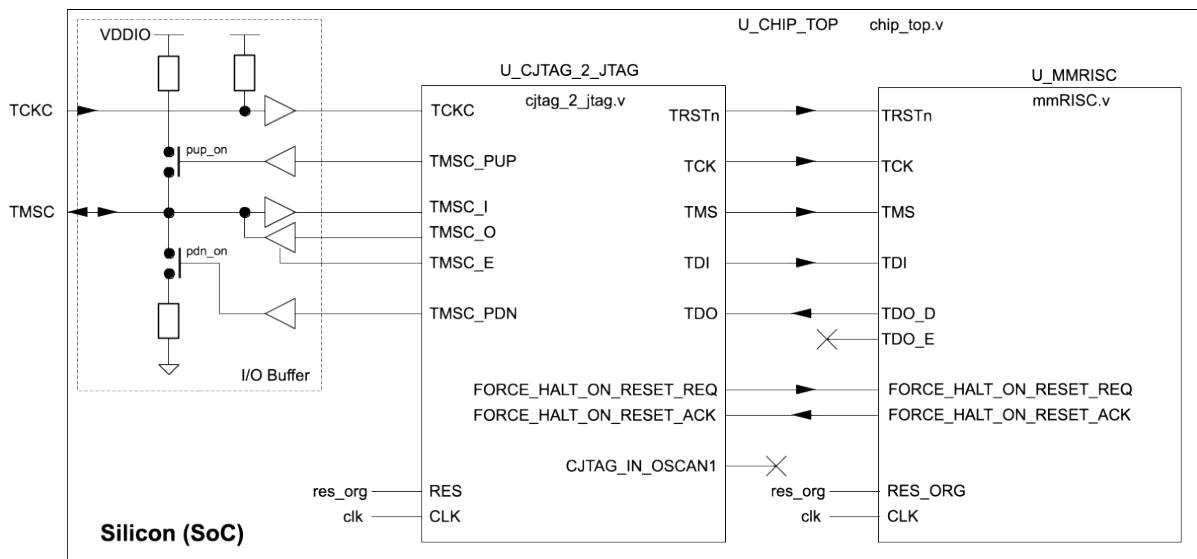
```

Listing 1.4: RTL code of *cjtag_adapter.v*

If you want to have a USB to cJTAG adapter which follows the standard specification of cJTAG, the adapter should watch the Escape sequence, the Activation sequence, and the OSCAN1 sequence. During the Escape sequence and the activation sequence, the driving state of TMSC is controlled by only TMS. Once entering OSCAN1 mode, the driving state of TMSC is controlled by not only TMS but also TCKC (HiZ during TCKC=1). This case can be realized by using some MCU device for the adapter.

1.31.5 cJTAG Implementation for Silicon (SoC)

An example of a cJTAG Implementation for Silicon (SoC) is shown in Figure 1.20. At the same layer of mmRISC, a converter from cJTAG to JTAG (CJTAG_2_JTAG) is instantiated. The I/O buffer block including TCKC and TMSC should handle both the pull-up and the bus keeper function as described in Section 1.31.2 which are supported by the CJTAG_2_JTAG block. The method to control the I/O Port for cJTAG Pins is shown in Listing 1.5.

**Figure 1.20:** cJTAG Implementation for Silicon (SoC)

```

// During Reset , PULLUP ON, PULLDN OFF
// Before OSCAN1, PULLUP ON, PULLDN OFF
// During OSCAN1, Bus Keeper
//           (PULLUP when TMSC_I==1, PULLDN when TMSC_I==0)
// TMSC is driven only when in OSCAN1 mode.
//           (oscan1_state[2] shows it is in TDO phase.)
//
assign TMSC_PUP = ( RES           )? 1'b1
        : (~CJTAG_IN_OSCAN1)? 1'b1
        : TMSC_I;
assign TMSC_PDN = ( RES           )? 1'b0
        : (~CJTAG_IN_OSCAN1)? 1'b0
        : ~TMSC_I;
assign TMSC_E  = (RES )? 1'b0
        : (TCKC)? 1'b0

```

```
: (CJTAG_IN_OSCAN1 & oscan1_state[2])? 1'b1
: 1'b0;
```

Listing 1.5: I/O Port Control for cJTAG Pins

1.31.6 JTAG/cJTAG Implementation for FPGA and Test Bench

An implementation for FPGA and Test Bench supporting both JTAG and cJTAG is shown in Figure 1.21. In the CHIP_TOP layer, which is the SoC top layer, is surrounded by an additional upper layer CHIP_TOP_WRAP. The cJTAG Adapter described in Section 1.31.4 is located in the layer CHIP_TOP_WRAP as CJTAG_ADAPTER. The cJTAG external pin controls described in Section 1.31.2 are also implemented in the layer CHIP_TOP_WRAP. You can switch the debug interface between 4-wire JTAG or 2-wire cJTAG by setting the signal level of "enable_cjtag" in chip_top_wrap.v. This signal is connected to GPIO2[6] corresponding to SW6 on DE10-Lite board to test both cases.

If you have a dedicated USB-cJTAG adapter, you can directly connect it to TCKC and TMSC of the SoC without loopback connections from U_CJTAG_ADAPTER in Figure 1.21.

If you want to realize a cJTAG Adapter which follows the cJTAG standard specification as described in the last paragraph of Section 1.31.4, the CJTAG_ADAPTER block should watch the cJTAG protocol sequence like the CJTAG_2_JTAG block does.

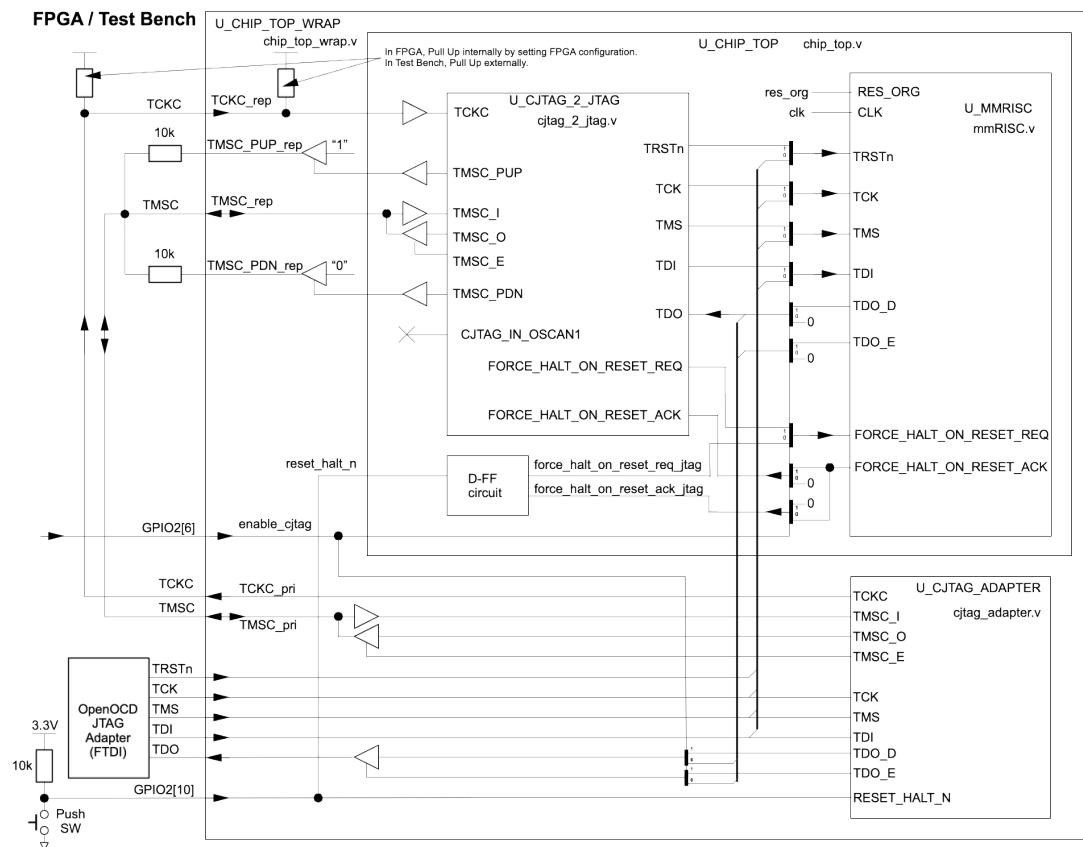


Figure 1.21: cJTAG Implementation for FPGA and Test Bench

1.32 Security (Authentication)

The debug specification of RISC-V supports Security (Authentication). If security is enabled, only following accesses from JTAG/cJTAG are permitted.

1. authenticated in dmstatus is readable.
2. authbusy in dmstatus is readable.
3. version in dmstatus is readable.
4. dmactive in dmcontrol is readable and writable.
5. authdata is readable and writable.

As shown in Figure 1.22, there are three input signals related to security.

If DEBUG_SECURE is Low, the security is disabled, so DEBUG_SECURE_CODE_0[31:0] and DEBUG_SECURE_CODE_1[31:0] are ignored.

If DEBUG_SECURE is High, the security is enabled. The debug authentication is accepted if authdata matches to DEBUG_SECURE_CODE_0[31:0] or DEBUG_SECURE_CODE_1[31:0].

The reason why two secure codes are prepared is as follows. Usually the secure code is stored, for example, in non-volatile memory (FLASH, FUSE etc.) integrated in the chip by end-user. If the stored code is undefined in such case that the non-volatile memory has not been initialized due to the wafer being very fresh. In the case, if the secure is enabled and secure code is only one, it is impossible to match the authcode to the unknown code. So the mmRISC-1 receives two secure codes, and the one code is set by end-user, and the other code is fixed by chip manufacturer as secret master key. Even if the secure code defined by end-user is unknown, manufacturer can open the key.

When you want to set authdata from OpenOCD, write "riscv authdata_write AUTHCODE" after "init" command in OpenOCD configuration file like as Listing 1.6 which is based on USB-JTAG interface with FT2232D chip.

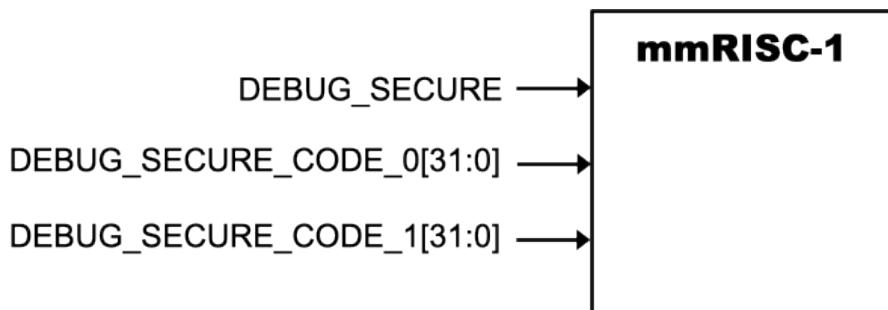


Figure 1.22: Security Configuration

```

...
init
riscv authdata_write 0xbeefcafe
...
  
```

Listing 1.6: Authentication Command in OpenOCD Configuration File

1.33 Low Power Mode

A typical SoC has a low power controller to put the CPU core into low power mode. It seems that your SoC with mmRISC-1 has a similar configuration as shown in Figure 1.23.

The control timing is described below. As shown in Figure 1.24, please assert STBY_REQ when the low power controller receives a STBY request to put the CPU into low power consumption mode. All HARTs in mmRISC-1 temporarily stop executing instructions, and each HART asserts the stby_ack signal once all instruction executions in the pipeline become empty. After all HART's stby_acks are asserted, mmRISC-1 asserts the STBY_ACK signal to inform the low power controller that mmRISC-1 has entered to STBY state. After that, the low power controller should stop the system clock to the CPU. Then, the mmRISC-1 becomes physically low power consumption state. The reason why the CPU flushes the pipeline before entering a low power state is to prevent the clock from stopping while the instruction or data memory access is incomplete.

To exit from the low power consumption state, please restart the clock to the CPU before negating the STBY_REQ signal. The mmRISC-1 negates the STBY_ACK signal and each hart resumes instruction execution.

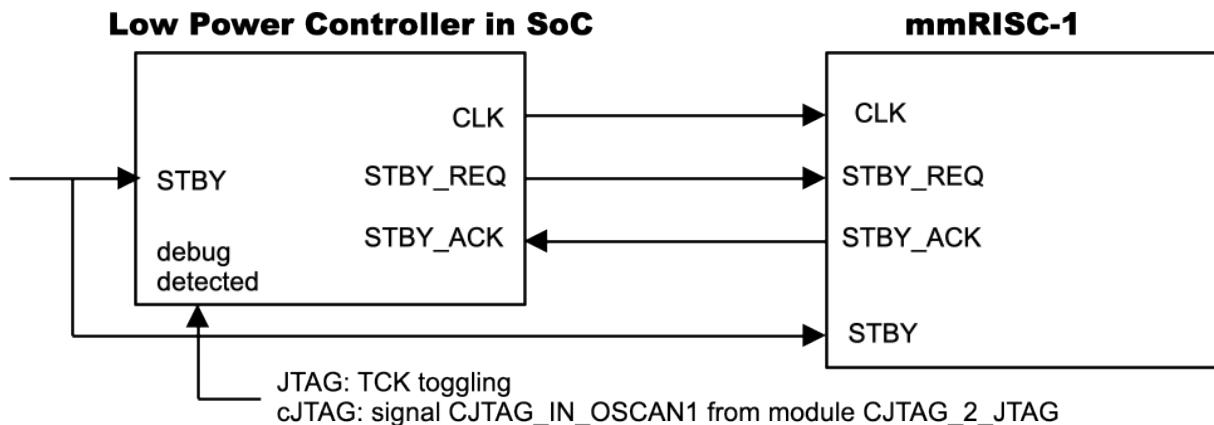


Figure 1.23: Block Diagram of Low Power Support

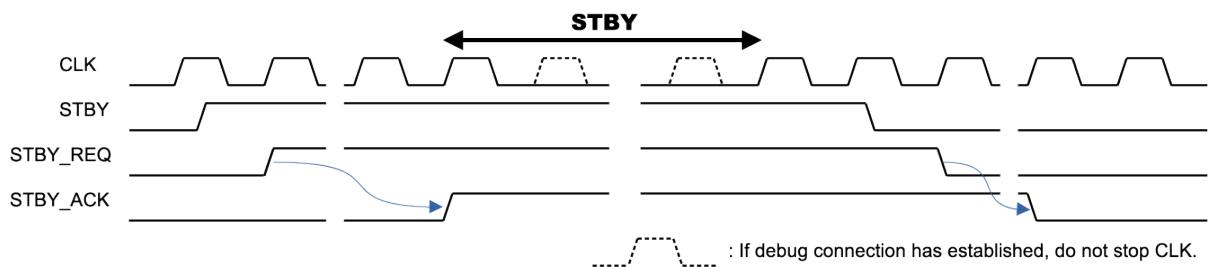


Figure 1.24: Timing of Low Power Support

Note 1 : An output signal STBY_ACK shows that mmRISC-1 core enters low power mode. During the state, if the system clock is stopping, when a debugger such as OpenOCD accesses the mmRISC-1 via JTAG or cJTAG interface, the access will fail because JTAG DTM (Debug Transport Module) requires toggling CLK (System Clock). So, once a debug transaction on JTAG/cJTAG occurs, the low power controller should not stop CLK even

when a low power transaction described above. The way to detect the debug transaction is (1) For, 4-wire JTAG, Detecting TCK Toggling, or (2) For 2-wire cJTAG, Detecting CJTAG_IN_OSCAN1 is high level.

Note 2 : Even when in STBY state, the debugger can access mmRISC-1 if CLK does not stop. But if the Hart-Available status implemented in DMSTATUS register of DM (Debug Module) shows "Unavailable" during STBY, the debugger might report access errors. You can select whether the Hart-Available status in DMSTATUS is reflected by STBY state or not by setting UNAVAILABLE_WHEN_STBY switch in defines_core.v. The default setting is that the switch is OFF and Hart-Available status is always 1.

Chapter 2

mmRISC-1 Tiny SoC for FPGA

This chapter describes an example implementation of mmRISC-1 SoC.

2.1 Overview of SoC

The “mmRISC_1 Tiny SoC for FPGA” is a sample design of the mmRISC-1 application. The overview is shown in Table 2.1. Target FPGA is the Intel MAX 10, and target board is Terasic DE10-Lite Board.

Item	Description
CPU Core	mmRISC-1 (1 hart) with RV32IMAC or RV32IMAFC
Debug Interface	RISC-V standard JTAG Debug (Rev.0.13.2) Interface : 4-wire JTAG or 2-wire cJTAG
Internal System Bus	Multi Layer Bus Matrix, AHB-Lite
Instruction Memory	128KB RAM, 1 cyc Read
Data Memory	48KB RAM, 1 cyc Read, 1cyc Write
External SDRAM	Simple Interface to 64MB (32MB x 16bit) SDRAM
MTIME (Timer)	Memory mapped MTIME which generates periodic IRQ_MTIME and software interrupt IRQ_MSOF
INT_GEN (Interrupt Generator)	Interrupt Generator which generates 64 IRQ[63:0] and external interrupt IRQ_EXT
UART	Simple UART with Baud Rate Generator, 1ch
I2C	Simple Master I2C, 2ch
SPI	Simple Master SPI, 1ch with multiple Chip Select Signals
GPIO	General Purpose Input Output Port, 32bits x 3 ports
Reset	Power on Rest and External Input
Clock	External Clock Input + PLL
Target FPGA	Intel MAX 10 10M50DAF484C7G
Target FPGA Board	Terasic DE10-Lite Board
JTAG Interface	FTDI FT2232D (Olimex ARM-USB-OCD(H) Compatible)
Supported Special Function	(1) Both 4-wire JTAG and 2-wire cJTAG (2) Authentication for Debugger (3) Halt on Rest (4) Low Power Mode (STBY)
RTL	Verilog-2001 / System Verilog

Table 2.1: Overview of mmRISC-1 Tiny SoC for FPGA

2.2 Block Diagram of SoC

Block diagram of mmRISC-1 Tiny SoC for FPGA is shown in Figure 2.1 and Figure 2.2. The former is the one with 4-wire JTAG debugger interface, and the latter is the one with 2-wire cJTAG debugger interface. You can switch the debug interface between 4-wire JTAG or 2-wire cJTAG by setting the signal level of "enable_cjtag" in chip_top_wrap.v. This signal is connected to GPIO2[6] to test both cases, as described in Section 1.31.6.

The mmRISC-1 cpu core has 1 hart with RV32IMAC or RV32IMAFC ISA which is combined with JTAG Debugger.

Internal system bus is Multilayer Bus Matrix with AHB-Lite interface connecting CPU core, memories and all memory mapped peripheral devices. Instruction memory is a RAM with 1cyc read cycle time. Data memory is another RAM with 1cyc read cycle time and 1cyc write cycle time.

The SoC has simple SDRAM interface for 64MB (32MB x 16bit) SDRAM to expand work memory area.

UART is a simple asynchronous Tx/Rx device with configurable baud rate generator. I2C is a Inter Integrated Circuit for 2-wire serial communication interface. The SoC has 2-channel master I2C. SPI is a Serial Peripheral Interface for synchronous communication interface. The SoC has 1-channel master SPI. GPIO is general purpose Input Output with 32bits x 3ports. INT_GEN and MTIME are peripherals described in mmRISC-1 chapter.

Reset is generated from external input or power-on-reset logic. System clock frequency depends on configuration of CPU ISA according to result of timing analysis. As for the target FPGA, RV32IMAC without FPU configuration can operate in 20MHz wheres RV32IMAFC with FPU in 16.67MHz. The input clock frequency of the target board is 50MHz, so PLL makes appropriate clock frequency.

JTAG Debugger interface for OpenOCD is a handmade board using FTDI FT2232D chip which is almost compatible with commercial Olimex ARM-USB-OCD(H). Of course you can use the commercial product as JTAG interface.

As for the FPGA system with 4-wire JTAG debug interface as shown in Figure 2.1, the JTAG signals from USB to JTAG adapter with FTDI chip are connected to mmRISC-1 core directly.

On the other hand, as for the FPGA system with 2-wire cJTAG debug interface as shown in Figure 2.2, the JTAG signals from USB to JTAG adapter with FTDI chip are inputted to the FPGA once, and "cJTAG Adapter" block converts them to 2-wire cJTAG signals which are connected to cJTAG_2_JTAG block through external loop back path, and at last the cJTAG_2_JTAG block converts cJTAG signals to JTAG signal which are connected to mmRISC-1 core.

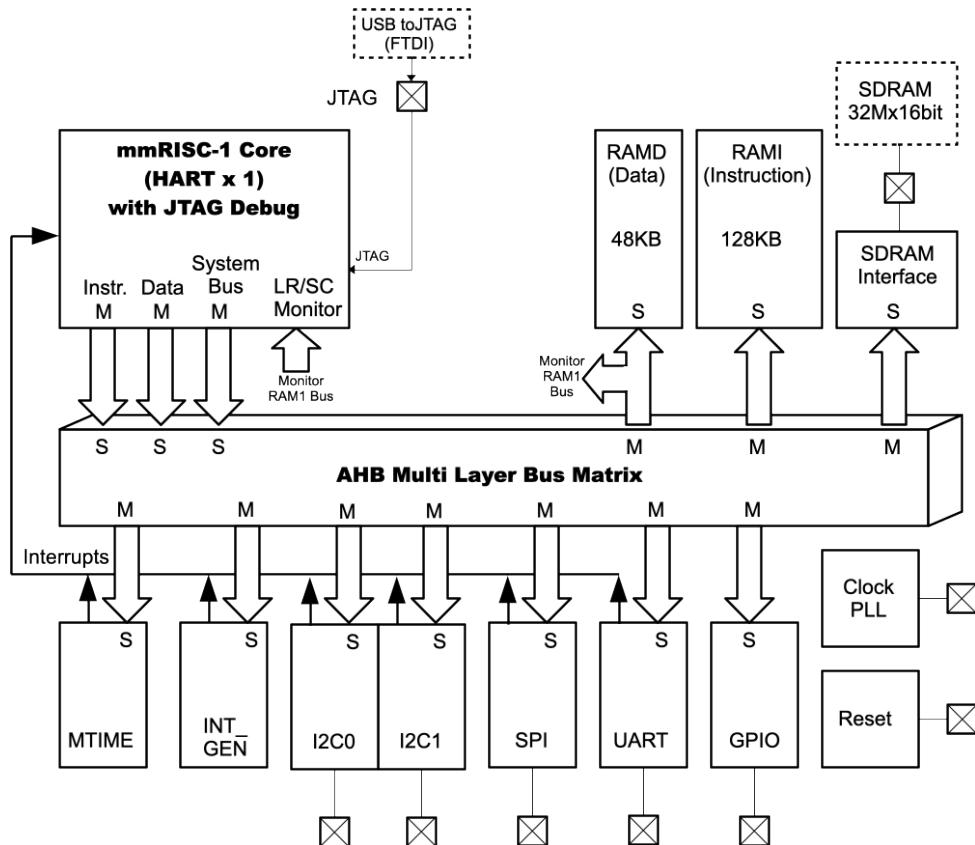


Figure 2.1: Block Diagram of mmRISC-1 Tiny SoC for FPGA with 4-wire JTAG interface (if "enable_cjtag" is fixed to low level)

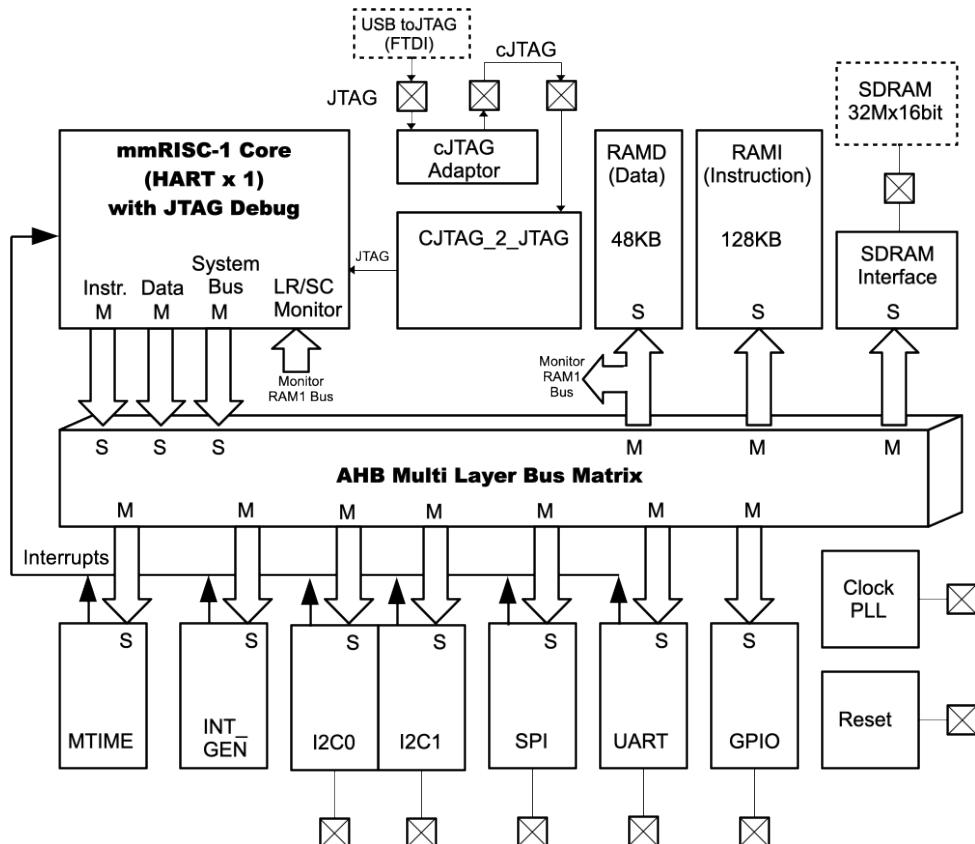


Figure 2.2: Block Diagram of mmRISC-1 Tiny SoC for FPGA with 2-wire cJTAG interface (if "enable_cjtag" is fixed to high level)

2.3 RTL Files and Structure of SoC

RTL files of mmRISC-1 tiny SoC are shown in Table 2.2, and the corresponding RTL structure is shown in Figure 2.3.

Directory	File Name	Description	Note
verilog/common	defines_chip.v	Definitions and Configurations for Chip	
verilog/chip	chip_top_wrap.v	Wrap of Top Layer of Chip	
verilog/chip	chip_top.v	Top Layer of Chip	
verilog/cjtag	cjtag_2_jtag.v	Converter from cJTAG to JTAG	
verilog/cjtag	cjtag_adapter.v	cJTAG Adapter (JTAG → cJTAG)	
verilog/ahb_matrix	ahb_top.v	AHB Bus Matrix Top Layer	
verilog/ahb_matrix	ahb_master.v	AHB Bus Matrix Master Port	
verilog/ahb_matrix	ahb_interconnect.v	AHB Bus Matrix Interconnect	
verilog/ahb_matrix	ahb_arb.v	AHB Bus Matrix Arbiter	
verilog/ahb_matrix	ahb_slave.v	AHB Bus Matrix Slave Port	
verilog/ram	ram.v	RAM for Instruction and Data Memory	
verilog/ram	ram_fpga.v	RAM for Instruction for FPGA with Initial Data	
fpga	RAM128KB_DP.v	Block RAM IP for ram_fpga.v	
fpga	RAM128KB_DP.mif	Block RAM Initial Data	
verilog/ahb_sdram/logic	ahb_lite_sdram.v	Simple SDRAM Interface (for 64MB : 32M x 16bits)	Zhelnio
verilog/ahb_sdram/model	sdr.v	External SDRAM Model for Test Bench	Micron
verilog/ahb_sdram/model	sdr_parameters.vh	External SDRAM Model Parameters for sdr.v	
verilog/int_gen	int_gen.v	Interrupt Generator	
verilog/uart	uart.v	UART Top Layer	
verilog/uart/sasc/trunk/rtl/verilog	sasc_top.v	Simple Asynchronous Serial Controller Top Layer	OpenCores SASC IP
verilog/uart/sasc/trunk/rtl/verilog	sasc_fifo4.v	Simple Asynchronous Serial Controller FIFO	
verilog/uart/sasc/trunk/rtl/verilog	sasc_brg.v	Simple Asynchronous Serial Controller Baud Rate Generator	
verilog/uart/sasc/trunk/rtl/verilog	timescale.v	Simple Asynchronous Serial Controller Time Scale	
verilog/i2c	i2c.v	I2C Top Layer	
verilog/i2c/i2c/trunk/rtl/verilog	i2c_master_top.v	I2C Master Controller Top Level	OpenCores I2C IP
verilog/i2c/i2c/trunk/rtl/verilog	i2c_master_byte_ctrl.v	I2C Master Controller Byte Control Block	
verilog/i2c/i2c/trunk/rtl/verilog	i2c_master_bit_ctrl.v	I2C Master Controller Bit Control Block	
verilog/i2c	i2c_slave_model.v	I2C Slave Model for Test Bench	
verilog/spi	spi.v	SPI Top Layer	
verilog/spi/simple_spi/trunk/rtl/verilog	simple_spi_top.v	Simple SPI Top Level	OpenCores Simple SPI
verilog/spi/simple_spi/trunk/rtl/verilog	fifo4.v	Simple SPI FIFO Block	
verilog/port	port.v	GPIO	
fpga	PLL.v	PLL IP for FPGA (50MHz - > 20MHz and 16.667MHz)	
mmRISC-1 RTLs	Table 1.3	mmRISC-1 Block	

Table 2.2: RTL Files of SoC

Basically, the debug interface of mmRISC-1 is 4-wire JTAG, but you can use 2-wire cJTAG (Compact JTAG) instead. The chip top RTL is surrounded by chip_top_wrap.v as shown in Figure 1.21. You can switch the debug interface between 4-wire JTAG or 2-wire cJTAG by setting the signal level of "enable_cjtag" in chip_top_wrap.v. This signal is connected to GPIO2[6] to test both cases, as described in Section 1.31.6.

When selecting cJTAG interface, the cjtag_adapters.v is activated in the layer of chip_top_wrap.v, and cjtag_2_jtag.v is utilized in the layer of chip_top.v.

The SoC RTL structure contains cJTAG related logics even when you do not use cJTAG interface. When you fix the signal "enable_cjtag" to low level and synthesize whole SoC RTL, the unnecessary cJTAG related logics will be eliminated by the logic synthesizing tool.

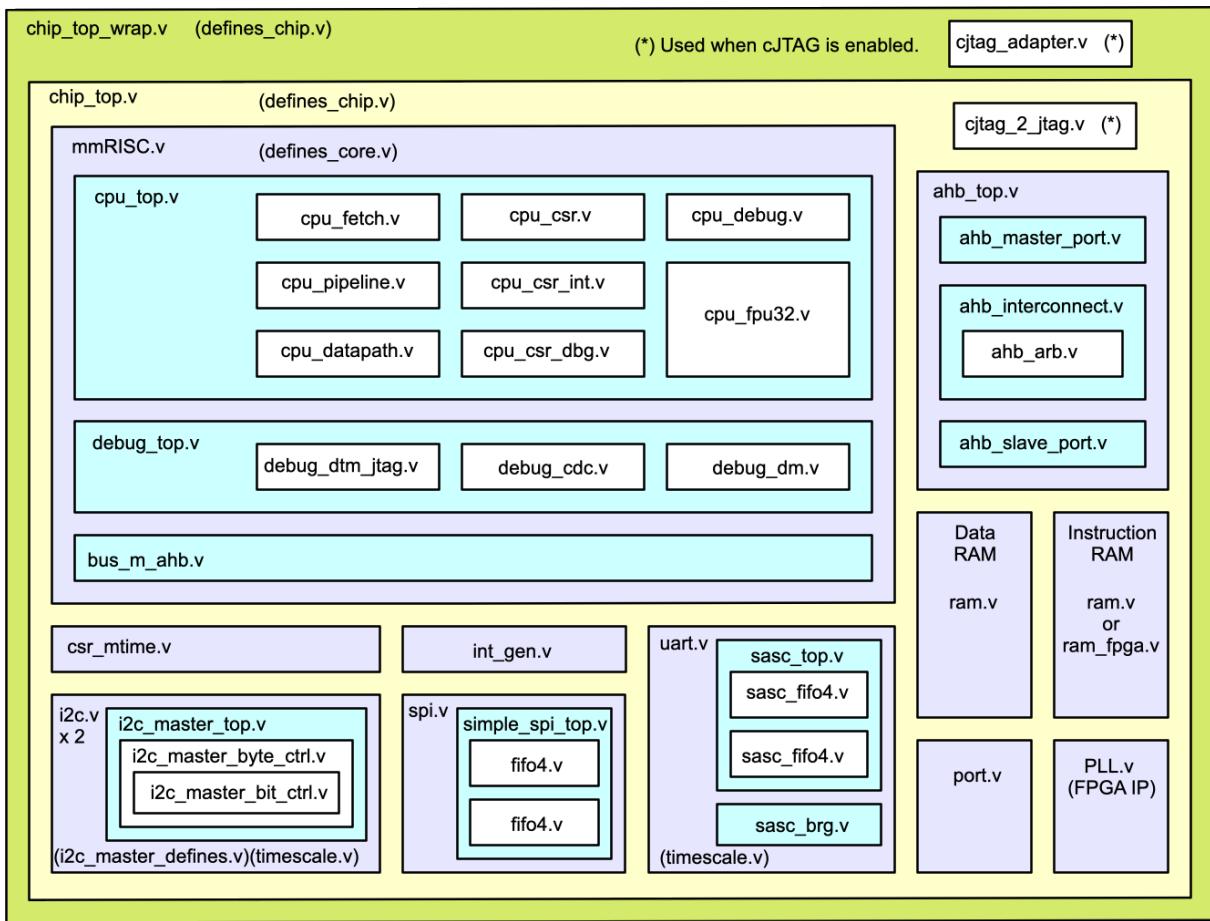


Figure 2.3: Structure of RTL Files of SoC

2.4 Input / Output Signals of SoC

Input / Output Signals of Tiny SoC Top Layer (`chip_top_wrap.v`) are shown in Table 2.3.

Group	Direction	Width	Name	Description	Note
System	input		RES_N	External Reset Input (negative)	
System	input		CLK50	External Clock Input (50MHz)	
System	input		STBY_REQ	Stand-by Request	
System	output		STBY_ACK_N	Stand-by Acknowledge (negative)	
System	output		RESOUT_N	External Reset Output	
System	inout		SRSTn	System Reset I/O (Open Drain)	Only for Simulation
JTAG	input		TRSTn	JTAG Tap Reset	Connected to USB-JTAG
JTAG	input		TCK	JTAG Clock	adapter (FTDI chip)
JTAG	input		TMS	JTAG Mode Select	even if you want to
JTAG	input		TDI	JTAG Data Input	use cJTAG interface.
JTAG	output		TDO	JTAG Data Output (3-state)	
JTAG	output		RTCK	Return Clock	Only for Simulation
cJTAG	output		TCKC_pri	cJTAG TCKC Loop Primary	When you use cJTAG,
cJTAG	input		TCKC_rep	cJTAG TCKC Loop Replica	each T***_pri should be
cJTAG	inout		TMSC_pri	cJTAG TMSC Loop Primary	connected to T***_rep
cJTAG	inout		TMSC_rep	cJTAG TMSC Loop Replica	at outside of the SoC.
cJTAG	output		TMSC_PUP_rep	cJTAG TMSC should be Pulled Up when 1	
cJTAG	output		TMSC_PDN_rep	cJTAG TMSC should be Pulled Down when 0	
GPIO	inout	[31:0]	GPIO0	GPIO0 32bit	
GPIO	inout	[31:0]	GPIO1	GPIO1 32bit	
GPIO	inout	[31:0]	GPIO2	GPIO2 32bit	
UART	input		RXD	UART Receive Data	
UART	output		TXD	UART Transmit Data	
I2C0	inout		I2C0_SCL	I2C0 SCL	
I2C0	inout		I2C0_SDA	I2C0 SDA	
I2C0	output		I2C0_ENA	I2C0 External Device Enable	Fixed to 1
I2C0	output		I2C0_ADR	I2C0 External Device Address Select	Fixed to 0
I2C0	input		I2C0_INT1	I2C0 External Device Interrupt Input 1	
I2C0	input		I2C0_INT2	I2C0 External Device Interrupt Input 2	
I2C1	inout		I2C1_SCL	I2C1 SCL	
I2C1	inout		I2C1_SDA	I2C1 SDA	
SPI	output	[3:0]	SPI_CSN	SPI Chip Select	
SPI	output		SPI_SCK	SPI Clock	
SPI	output		SPI_MOSI	SPI Master Output / Slave Input	
SPI	input		SPI_MISO	SPI Master Input / Slave Output	
SDRAM	output		SDRAM_CLK	SDRAM Clock	
SDRAM	output		SDRAM_CKE	SDRAM Clock Enable	
SDRAM	output		SDRAM_CS _n	SDRAM Chip Select	
SDRAM	output	[1:0]	SDRAM_DQM	SDRAM Byte Data Mask	
SDRAM	output		SDRAM_RAS _n	SDRAM Row Address Strobe	
SDRAM	output		SDRAM_CAS _n	SDRAM Column Address Strobe	
SDRAM	output		SDRAM_WEn	SDRAM Write Enable	
SDRAM	output	[1:0]	SDRAM_BA	SDRAM Bank Address	
SDRAM	output	[12:0]	SDRAM_ADDR	SDRAM Address	
SDRAM	inout	[15:0]	SDRAM_DQ	SDRAM Data	

Table 2.3: Input / Output Signals of Tiny SoC Top Layer (`chip_top_wrap.v`)

2.5 Clock & Reset for SoC

The clock structure of the SoC is shown in Figure 2.4. For simulation, an internal clock clk01 is directly connected to the external clock input CLK50. For FPGA, the clk01 is generated by a PLL, and its frequency is 20MHz if the RV32F/RV32FC ISA are not implemented, or 16MHz if they are implemented. On the other hand, a slow clock clk2 with a frequency of 100KHz is also generated. The system clock clk is selected from either clk01 or clk2 by a glitch-less selector as shown in Figure 2.5. The selection signal is the GPIO2[7] input from an external pin. The reason why such a slow clock clk2 can be selected as the system clock is to verify that the any magnitude of the ratio between the system clock (clk) frequency and the JTAG/cJTAG clock (TCK/TCKC) frequency is acceptable in the mmRISC-1 core.

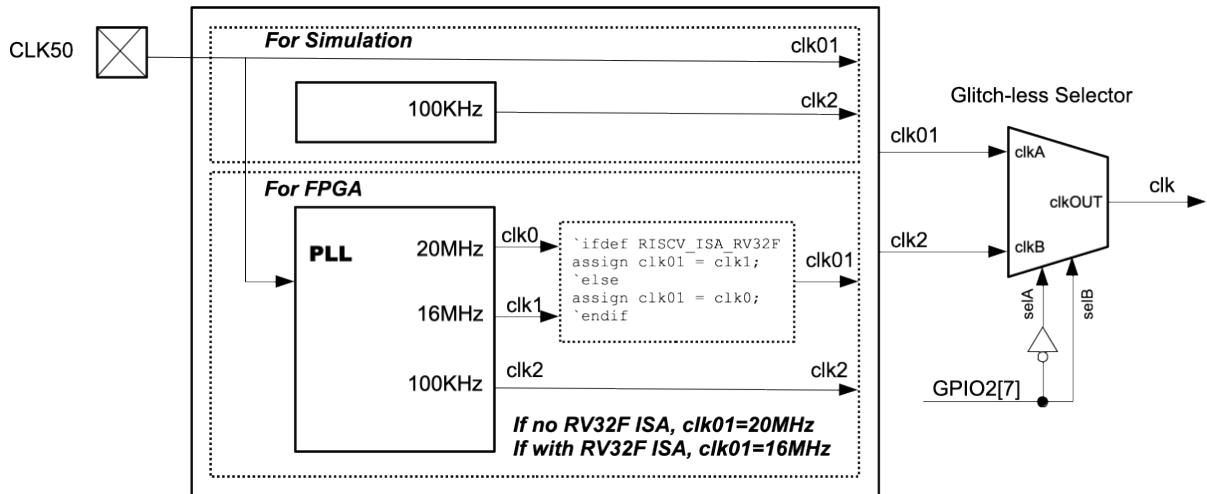


Figure 2.4: Clock Structure in Tiny SoC

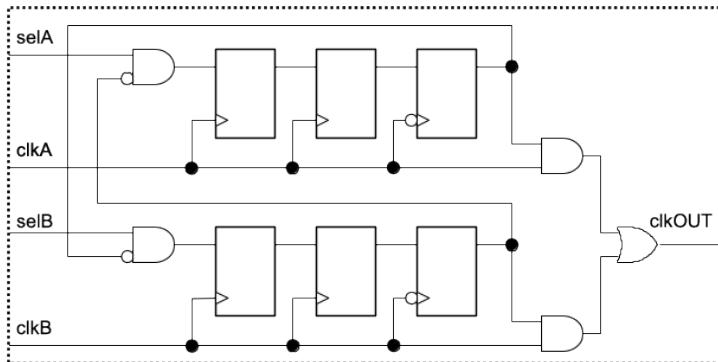


Figure 2.5: Glitch-less Clock Selector

Reset structure in the SoC is shown in Figure 2.6. Power on reset circuit is created using counter logic which is initialized at startup by an initial statement for simulation or by setting power-up-level parameters for FPGA. The power on reset signal and external reset input RES_N are used as the source of RES_ORG of mmRISC block. For FPGA, SRSTn_IN of mmRISC block is simply from external reset input RES_N and SRSTn_OUT is not used. For simulation, SRSTn_IN is from SRSTn and the SRSTn is driven by SRSTn_OUT of mmRISC block. RES_SYS output from mmRISC block is used as the reset of peripheral blocks. Reset structure in mmRISC block is shown in Figure 1.6.

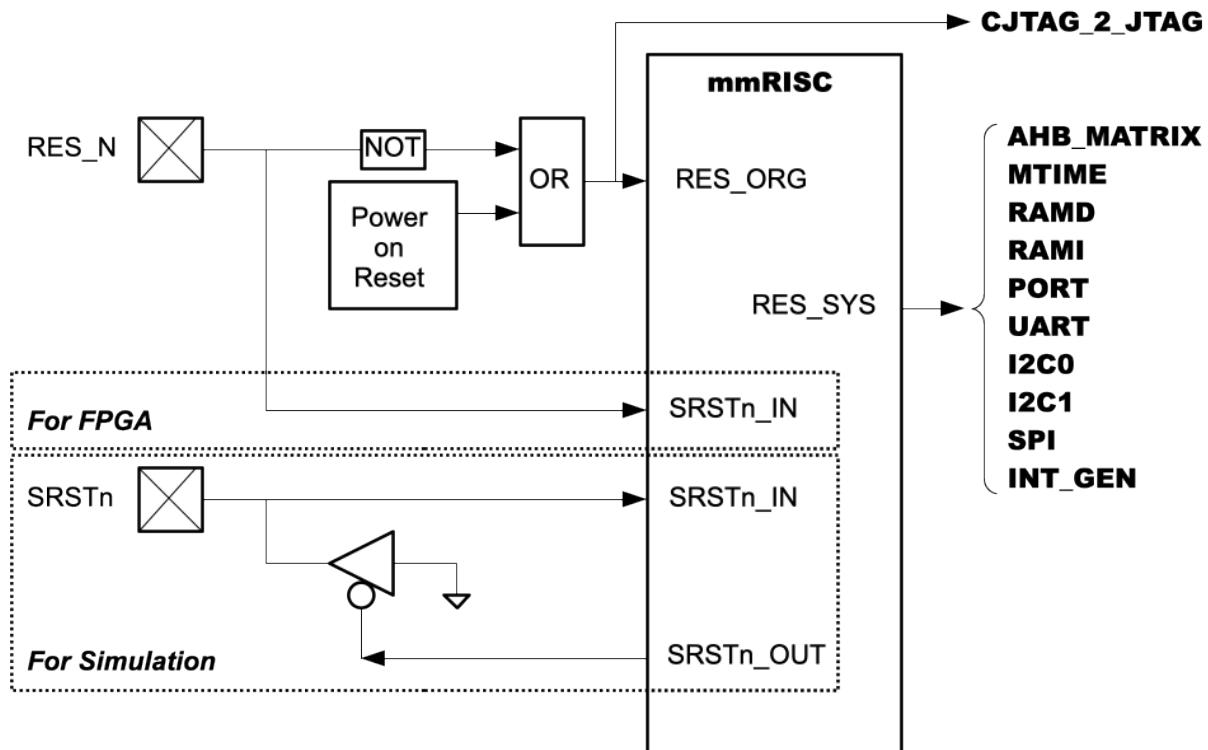


Figure 2.6: Reset Structure in Tiny SoC

2.6 Memory Map

Memory map of the SoC is shown in Table 2.4. All master ports of mmRISC block can access all peripheral resources assigned at the same address. Each area consists of an actual physical area and its shadowed areas.

Address Range	Instruction Bus	Data Bus	System Bus (debugger)	Note
0x00000000-0x48fffff	reserved	reserved	reserved	
0x49000000-0x4900001f	MTIME	MTIME	MTIME	
0x49000020-0x7fffffff	reserved	reserved	reserved	
0x80000000-0x87fffffff	SDRAM 64MB	SDRAM 64MB	SDRAM 64MB	
0x88000000-0x8fffffff	RAMD (data) 48KB	RAMD (data) 48KB	RAMD (data) 48KB	FPGA Block RAM
0x90000000-0x9fffffff	RAMI (instr) 128KB	RAMI (instr) 128KB	RAMI (instr) 128KB	
0xa0000000-0xafffffff	PORT	PORT	PORT	
0xb0000000-0xbfffffff	UART	UART	UART	
0xc0000000-0xcfffffff	INT_GEN	INT_GEN	INT_GEN	
0xd0000000-0xd00000ff	I2C0	I2C0	I2C0	
0xd0000100-0xd00001ff	I2C1	I2C1	I2C1	
0xd0000200-0xdfffffff	reserved	reserved	reserved	
0xe0000100-0xefffffff	SPI	SPI	SPI	
0xf0000200-0xffffffff	reserved	reserved	reserved	

Table 2.4: Memory Map of Tiny SoC

2.7 Interrupt Assignment

Interrupt assignment in the Tiny SoC is shown in Table 2.5. The IRQ[5:0] are made by a logical OR between interrupt requests from peripherals (UART, I2C and SPI) and interrupt requests from INT_GEN.

Interrupt Name	Source Module	Exception Code (MCAUSE)	Vector Address (Vectorized Mode)
IRQ_MSOF	MTIME	0x03	base + 0x0000000c
IRQ_MTIME	MTIME	0x07	base + 0x0000001c
IRQ_EXT	INT_GEN	0x0b	base + 0x0000002c
IRQ0	INT_GEN UART	0x10	base + 0x00000040
IRQ1	INT_GEN I2C0	0x11	base + 0x00000044
IRQ2	INT_GEN I2C1	0x12	base + 0x00000048
IRQ3	INT_GEN SPI	0x13	base + 0x0000004c
IRQ4	INT_GEN I2C_INT1 (external)	0x14	base + 0x00000050
IRQ5	INT_GEN I2C_INT2 (external)	0x15	base + 0x00000054
IRQ6	INT_GEN	0x16	base + 0x00000058
...
IRQ63	INT_GEN	0x4f	base + 0x0000013c

Table 2.5: Interrupt Assignment in Tiny SoC

2.8 Multi-Layer AHB Bus Matrix

Overview

Provided Multi-Layer AHB Bus Matrix is originally designed for mmRISC-1 Tiny SoC. Both master port counts and slave port counts can be specified by any number using module parameters (MASTERS, SLAVES) defined at instantiation of this module in the upper layer, respectively. The Bus Matrix supports the priority of master access on each slave port from either fixed or round-robin.

Input / Output Signals

Input / Output signals of Multi-Layer AHB Bus Matrix are shown in Table 2.6. Each Master port signal and Slave port signal is defined as an array with element counts which are the same as the port counts (MASTERS and SLAVES).

Group	Direction	Width	Name	Description	Note
Config	Parameter		MASTERS	Counts of Master Port	
Config	Parameter		SLAVES	Counts of Slave Port	
System	input		HCLK	System Clock	
System	input		M_HRESETn	System Reset	
Master	input		M_HSEL[0:MASTERS-1]	AHB Lite Master Select	
Master	input	[1:0]	M_HTRANS[0:MASTERS-1]	AHB Lite Master Transfer Type	
Master	input		M_HWRITE[0:MASTERS-1]	AHB Lite Master Write	
Master	input		M_HMASTLOCK[0:MASTERS-1]	AHB Lite Master Locked Transfer	
Master	input	[2:0]	M_HSIZEx[0:MASTERS-1]	AHB Lite Master Access Size	
Master	input	[2:0]	M_HBURST[0:MASTERS-1]	AHB Lite Master Burst Access	
Master	input	[3:0]	M_HPROTx[0:MASTERS-1]	AHB Lite Master Protection	
Master	input	[31:0]	M_HADDR[0:MASTERS-1]	AHB Lite Master Address	
Master	input	[31:0]	M_HWDATA[0:MASTERS-1]	AHB Lite Master Write Data	
Master	input		M_HREADY[0:MASTERS-1]	AHB Lite Master Ready Input	
Master	output		M_HREADYOUT[0:MASTERS-1]	AHB Lite Master Ready Output	
Master	output	[31:0]	M_HRDATA[0:MASTERS-1]	AHB Lite Master Read Data	
Master	output		M_HRESP[0:MASTERS-1]	AHB Lite Master Response	
Master	input	(*)	M_PRIORITY[0:MASTERS-1]	Master Priority	
Slave	output		S_HSEL[0:SLAVES-1]	AHB Lite Slave Select	
Slave	output	[1:0]	S_HTRANS[0:SLAVES-1]	AHB Lite Slave Transfer Type	
Slave	output		S_HWRITE[0:SLAVES-1]	AHB Lite Slave Write	
Slave	output		S_HMASTLOCK[0:SLAVES-1]	AHB Lite Slave Locked Transfer	
Slave	output	[2:0]	S_HSIZEx[0:SLAVES-1]	AHB Lite Slave Access Size	
Slave	output	[2:0]	S_HBURST[0:SLAVES-1]	AHB Lite Slave Burst Access	
Slave	output	[3:0]	S_HPROTx[0:SLAVES-1]	AHB Lite Slave Protection	
Slave	output	[31:0]	S_HADDR[0:SLAVES-1]	AHB Lite Slave Address	
Slave	output	[31:0]	S_HWDATA[0:SLAVES-1]	AHB Lite Slave Write Data	
Slave	output		S_HREADY[0:SLAVES-1]	AHB Lite Slave Ready Input	
Slave	input		S_HREADYOUT[0:SLAVES-1]	AHB Lite Slave Ready Output	
Slave	input	[31:0]	S_HRDATA[0:SLAVES-1]	AHB Lite Slave Read Data	
Slave	input		S_HRESP[0:SLAVES-1]	AHB Lite Slave Response	
Slave	input	[31:0]	S_HADDR_BASE[0:SLAVES-1]	Slave Address Base	
Slave	input	[31:0]	S_HADDR_MASK[0:SLAVES-1]	Slave Address Mask	
(*) [MASTERS_BIT-1:0](MASTERS_BIT=\$clog2(MASTERS))					

Table 2.6: Input / Output Signals of Multi-Layer AHB Bus Matrix)

Master access priority on each Slave port

Priority of master access on each slave port is specified by input signal M_PRIORITY[0:MASTERS-1]. For example, if the master port count MASTERS = 8, and M_PRIORITY[0:7]

(each element has a 3bit width) are specified as shown in Listing 2.1, master Port 0 is the highest priority group (level 0) with a single port. The next highest priority is a group (level 1) in which masters ports 1 to 3 rotate their priority in round-robin manner. The next highest priority is a group (level 2) in which master ports 4 to 5 rotate their priority in round-robin manner. The next priority group (level 3) is single master port 6. The lowest priority group (level 4) is single master port 7. Note that round-robin groups need to be consolidated by the adjacent bus master number. A settings shown in Listing 2.2 is not guaranteed.

```
M_PRIORITY[0]=0 : Level0 Single Group Highest
M_PRIORITY[1]=1 : Level1 Round-Robin Group among [1][2][3]
M_PRIORITY[2]=1 : Level1 Round-Robin Group among [1][2][3]
M_PRIORITY[3]=1 : Level1 Round-Robin Group among [1][2][3]
M_PRIORITY[4]=2 : Level2 Round-Robin Group among [4][5]
M_PRIORITY[5]=2 : Level2 Round-Robin Group among [4][5]
M_PRIORITY[6]=3 : Level3 Single Group
M_PRIORITY[7]=4 : Level4 Single Group Lowest
```

Listing 2.1: An Example of Priority Configuration

```
(NG) M_PRIORITY[0]=0 : Level0 Single Group Highest
(NG) M_PRIORITY[1]=1 : Level1 Round-Robin Group among [1][3][5]
(NG) M_PRIORITY[2]=2 : Level2 Round-Robin Group among [2][4]
(NG) M_PRIORITY[3]=1 : Level1 Round-Robin Group among [1][3][5]
(NG) M_PRIORITY[4]=2 : Level2 Round-Robin Group among [2][4]
(NG) M_PRIORITY[5]=1 : Level1 Round-Robin Group among [1][3][5]
(NG) M_PRIORITY[6]=3 : Level3 Single Group
(NG) M_PRIORITY[7]=4 : Level4 Single Group Lowest
```

Listing 2.2: An Example of Unacceptable Priority Configuration

Address Range Definition on each Slave port

Address range of each slave port is specified by input signals of S_HADDR_BASE and S_HADDR_MASK. For example, for slave port 0, if the input signals are configured as shown in Listing 2.3, address range of Slave port 0 becomes 0x01000000 – 0x010fffff. At all bit positions where the bit of S_HADDR_MASK is set to 1, the address issued by the master and each bit of S_HADDR_BASE are compared, and if they match, the slave port is accessed. However, make sure that there are no duplicate addresses among slave ports.

```
S_HADDR_BASE[0] = 32'h01000000
S_HADDR_MASK[0] = 32'hffff0000
```

Listing 2.3: An Example of Address Range Definition

About HMASTLOCK

On a slave port, if an access from the same master continues immediately after the access that the master sets HMASTLOCK to a slave, another bus master with any priority is not inserted. In other words, on a slave port, once a master access with HMASTLOCK=1 happens, other master which accesses to the slave port is stalled until the same master access with HMASTLOCK=0 finishes. This feature supports atomic memory access.

Functional Limitation

While a master is making burst access to a slave port, if another higher-priority bus master interrupts the burst access, the bus master on the slave port is switched even during burst. Even in that case, output values of access commands related to burst

information such as HTRANS (IDLE, BUSY, NOSEQ, SEQ), HBURST and HSIZE on the slave port are completely same as the ones that master outputs . Therefore burst information combined with HTRANS, HBURST and HSIZE on slave port is meaningless, and only HTRANS[1] and HSIZE indicate meaningful information as single access.

2.9 Peripheral : RAM (Instruction / Data)

Overview

The RAM is a single port read-write memory with 32bit data width. It is used as both instruction memory and data memory in the tiny SoC. The RAM size can be specified by module parameters (RAM_SIZE in bytes) defined at instantiation of this module in the upper layer. Access interface is AHB-Lite and supports 1cyc access for both reading and writing and 1cyc in-turn access between reading and writing. To realize complete 1cyc access even for AHB-Lite interface (write data is 1cyc after address command), write buffer and dual port RAM are used to implement it. For logic simulation of the SoC, the contents of RAM can be initialized by \$readmemh from file “rom.memh” .

How to initialize Instruction RAM contents when FPGA powers up

To initialize Instruction RAM contents when FPGA powers up, please use ram_fpga.v for instruction RAM instead of ram.v. In the ram_fpga.v, it instantiates RAM128KB_DP.v which can be initialized by RAM128KB_DP.mif. To generate RAM128KB_DP.mif from your Intel-hex file, please execute commands shown in Listing 2.4 and then configure your FPGA.

Input / Output Signals

Input / Output signals of RAM are shown in Table 2.7.

```
$ cd mmRISC-1 (move to mmRISC_1 directory)
$ ./tools/hex2mif [your.hex] > ./fpga/128KB_DP.mif
```

Listing 2.4: Commands to generate RAM initialization Data for FPGA

Group	Direction	Width	Name	Description	Note
Config	Parameter		RAM_SIZE	RAM Size in bytes	
System	input		RES	Reset	
System	input		CLK	System Clock	
AHB	input		S_HSEL	AHB Lite Slave Select	ignored
AHB	input	[1:0]	S_HTRANS	AHB Lite Slave Transfer Type	
AHB	input		S_HWRITE	AHB Lite Slave Write	
AHB	input		S_HMASTLOCK	AHB Lite Slave Locked Transfer	ignored
AHB	input	[2:0]	S_HSIZE	AHB Lite Slave Access Size	
AHB	input	[2:0]	S_HBURST	AHB Lite Slave Burst Access	ignored
AHB	input	[3:0]	S_HPROT	AHB Lite Slave Protection	ignored
AHB	input	[31:0]	S_HADDR	AHB Lite Slave Address	
AHB	input	[31:0]	S_HWDATA	AHB Lite Slave Write Data	
AHB	input		S_HREADY	AHB Lite Slave Ready Input	
AHB	output		S_HREADYOUT	AHB Lite Slave Ready Output	
AHB	output	[31:0]	S_HRDATA	AHB Lite Slave Read Data	
AHB	output		S_HRESP	AHB Lite Slave Response	always output 0

Table 2.7: Input / Output Signals of RAM)

2.10 Peripheral : SDRAM Controller)

Overview

SDRAM Controller interfaces with internal AHB Lite bus and an external 64MB (32MB x 16bits) single data rate SDRAM. This interface logic is published on https://github.com/zhelnio/ahb_lite_sdram and is very simple so that it supports only single full ROW-COLUMN access and burst access with row/column hit judgement is not supported. Also, this logic is configured for ISSI IS42S16320D and no configuration registers are implemented. The SDRAM Clock is simply generated as the inverted System Clock in chip_top.v because the system clock frequency is slow (20MHz or 16.667MHz). To simulate SDRAM access on Test Bench, SDRAM model from Micron is used. The file name are shown in Table 2.2.

Input / Output Signals

Input / Output signals of SDRAM Controller are shown in Table 2.8.

Group	Direction	Width	Name	Description	Note
System	input		HRESETn	Reset	
System	input		HCLK	System Clock	
AHB	input		HSEL	AHB Lite Slave Select	ignored
AHB	input	[1:0]	HTRANS	AHB Lite Slave Transfer Type	
AHB	input		HWRITE	AHB Lite Slave Write	
AHB	input		HMASTLOCK	AHB Lite Slave Locked Transfer	ignored
AHB	input	[2:0]	HSIZE	AHB Lite Slave Access Size	
AHB	input	[2:0]	HBURST	AHB Lite Slave Burst Access	ignored
AHB	input	[3:0]	HPROT	AHB Lite Slave Protection	ignored
AHB	input	[31:0]	HADDR	AHB Lite Slave Address	
AHB	input	[31:0]	HWDATA	AHB Lite Slave Write Data	
AHB	input		HREADY	AHB Lite Slave Ready Input	
AHB	output		HREADYOUT	AHB Lite Slave Ready Output	
AHB	output	[31:0]	HRDATA	AHB Lite Slave Read Data	
AHB	output		HRESP	AHB Lite Slave Response	always output 0
AHB	input		SI_Endian	Endian Control	ignored
SDRAM	output		CKE	SDRAM Clock Enable	
SDRAM	output		CSn	SDRAM Chip Select	
SDRAM	output		RASn	SDRAM Row Address Strobe	
SDRAM	output		CASn	SDRAM Column Address Strobe	
SDRAM	output		WEn	SDRAM Write Enable	
SDRAM	output	[12:0]	ADDR	SDRAM Address	
SDRAM	output	[1:0]	BA	SDRAM Bank Address	
SDRAM	inout	[15:0]	DQ	SDRAM Data	
SDRAM	output	[1:0]	DQM	SDRAM Byte Data Mask	

Table 2.8: Input / Output Signals of SDRAM Controller)

2.11 Peripheral : Peripheral : Interrupt Generator (INT_GEN)

Overview

INT_GEN (Interrupt Generator) generates interrupt requests of IRQ_EXT and IRQ[63:0] by software setting.

Input / Output Signals

Input / Output signals of INT_GEN are shown in Table 2.9.

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clock	
AHB	input		S_HSEL	AHB Lite Slave Select	ignored
AHB	input	[1:0]	S_HTRANS	AHB Lite Slave Transfer Type	
AHB	input		S_HWRITE	AHB Lite Slave Write	
AHB	input		S_HMASTLOCK	AHB Lite Slave Locked Transfer	ignored
AHB	input	[2:0]	S_HSIZE	AHB Lite Slave Access Size	
AHB	input	[2:0]	S_HBURST	AHB Lite Slave Burst Access	ignored
AHB	input	[3:0]	S_HPROT	AHB Lite Slave Protection	ignored
AHB	input	[31:0]	S_HADDR	AHB Lite Slave Address	
AHB	input	[31:0]	S_HWDATA	AHB Lite Slave Write Data	
AHB	input		S_HREADY	AHB Lite Slave Ready Input	
AHB	output		S_HREADYOUT	AHB Lite Slave Ready Output	
AHB	output	[31:0]	S_HRDATA	AHB Lite Slave Read Data	
AHB	output		S_HRESP	AHB Lite Slave Response	always output 0
INT	output		IRQ_EXT	External Interrupt Request	
INT	output	[63:0]	IRQ	Interrupt Request	

Table 2.9: Input / Output Signals of INT_GEN)

Control Registers

Controls registers of INT_GEN are shown in Table 2.10, Table 2.11 and Table 2.12, respectively. The interrupt request levels of IRQ_EXT and IRQ[63:0] are controlled by corresponding bits in each register.

Memory Mapped Peripheral Base Address = 0xc0000000					
Offset Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
31:1	0	31'h0	R	Always 0.	
0	irq_ext	1'b0	R/W	Generate External Interrupt 0: Negate External Interrupt 1: Assert External Interrupt	

Table 2.10: INT_GEN Control Register IRQ_EXT)

Memory Mapped Peripheral Base Address = 0xc0000000					
Offset Address	Name	Description			
0x0004	IRQ0	IRQ00 – IRQ31 Request			
Bit	Field	Initial	Access	Description	Note
31	0	1'b0	R/W	Generate IRQ31 0: Negate IRQ31 1: Assert IRQ31	
30	0	1'b0	R/W	Generate IRQ30 0: Negate IRQ30 1: Assert IRQ30	
...	
0	0	1'b0	R/W	Generate IRQ00 0: Negate IRQ00 1: Assert IRQ00	

Table 2.11: INT_GEN Control Register IRQ0)

Memory Mapped Peripheral Base Address = 0xc0000000					
Offset Address	Name	Description			
0x0008	IRQ1	IRQ32 – IRQ63 Request			
Bit	Field	Initial	Access	Description	Note
31	0	1'b0	R/W	Generate IRQ63 0: Negate IRQ63 1: Assert IRQ63	
30	0	1'b0	R/W	Generate IRQ62 0: Negate IRQ62 1: Assert IRQ62	
...	
0	0	1'b0	R/W	Generate IRQ32 0: Negate IRQ32 1: Assert IRQ32	

Table 2.12: INT_GEN Control Register IRQ1)

2.12 Peripheral : UART

Overview

UART is a simple asynchronous Tx/Rx device with a configurable baud rate generator and a 4-depths FIFO buffer for each Tx and Rx. The core logic of the UART is based on SASC (Simple Asynchronous Serial Controller) published on OpenCores (<https://opencores.org/projects/sasc>) by Usselmann Rudolf. Please see the IP's document for details. The UART can generate an interrupt request when the Tx FIFO buffer has room to write (not full) or the Rx buffer has at least one data to read (not empty).

Input / Output Signals

Input / Output signals of UART are shown in Table 2.13.

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clock	
AHB	input		S_HSEL	AHB Lite Slave Select	ignored
AHB	input	[1:0]	S_HTRANS	AHB Lite Slave Transfer Type	
AHB	input		S_HWRITE	AHB Lite Slave Write	
AHB	input		S_HMASTLOCK	AHB Lite Slave Locked Transfer	ignored
AHB	input	[2:0]	S_HSIZEx	AHB Lite Slave Access Size	
AHB	input	[2:0]	S_HBURST	AHB Lite Slave Burst Access	ignored
AHB	input	[3:0]	S_HPROT	AHB Lite Slave Protection	ignored
AHB	input	[31:0]	S_HADDR	AHB Lite Slave Address	
AHB	input	[31:0]	S_HWDATA	AHB Lite Slave Write Data	
AHB	input		S_HREADY	AHB Lite Slave Ready Input	
AHB	output		S_HREADYOUT	AHB Lite Slave Ready Output	
AHB	output	[31:0]	S_HRDATA	AHB Lite Slave Read Data	
AHB	output		S_HRESP	AHB Lite Slave Response	always output 0
UART	input		RXD	Receive Data	
UART	output		TXD	Transmit Data	
UART	input		CTS	Clear to Send	tie to 0 if unused
UART	output		RTS	Request to Send	
INT	output		IRQ_UART	Interrupt Request	

Table 2.13: Input / Output Signals of UART

Control Registers

Controls registers of UART are shown in Table 2.14 to Table 2.17. Please refer the technical document of the SASC on OpenCores (<https://opencores.org/projects/sasc>).

Memory Offset	Mapped Peripheral Base Address = 0xb0000000					
Address	Name	Description				
0x00	UART_TXD UART_RXD	UART TxD Data Register (When Writing) UART RxD Data Register (When Reading)				
7:0	txd rxd	8'h0	R/W	When Send a Data, write a 8bit value to this register. When Receive a Data, read a 8bit value from this register.		

Table 2.14: UART_TXD / UART_RXD

Memory Mapped Peripheral Base Address = 0xb0000000					
Offset Address	Name	Description			
0x01	UART_CSR	UART Control and Status Register			
Bit	Field	Initial	Access	Description	Note
7	ietx	1'b0	R/W	Interrupt Enable for Tx 0: Disable 1: Enable	Interrupt = ietx & txrdy ierx & rxrdy
6	ierx	1'b0	R/W	Interrupt Enable for Rx 0: Disable 1: Enable	
5:2	0	4'b0000	R	Always 0.	
1	txrdy	1'b1	R	Tx Data Ready, means Tx FIFO is not full.	
0	rxrdy	1'b0	R	Rx Data Ready, means Rx FIFO is not empty.	

Table 2.15: UART_CSR

Memory Mapped Peripheral Base Address = 0xb0000000					
Offset Address	Name	Description			
0x02	UART_BG0	UART Baud Rate Generator 0			
Bit	Field	Initial	Access	Description	Note
7:0	bg0	8'h0	R/W	Baud Rate Generator 0 Baud Rate = (fCLK/4) / ((bg0+2)*(bg1))	

Table 2.16: UART_BG0

Memory Mapped Peripheral Base Address = 0xb0000000					
Offset Address	Name	Description			
0x03	UART_BG1	UART Baud Rate Generator 1			
Bit	Field	Initial	Access	Description	Note
7:0	bg1	8'h0	R/W	Baud Rate Generator 1 Baud Rate = (fCLK/4) / ((bg0+2)*(bg1))	

Table 2.17: UART_BG1

2.13 Peripheral : I2C

Overview

This module is a simple master I2C interface based on OpenCores I2C Master controller (<http://www.opencores.org/projects/i2c/>) by Richard Herveille. Please refer to the IP's document for more details. The I2C can generate interrupt requests when (A) a byte transfer has been completed or (B) arbitration is lost.

Input / Output Signals

Input / Output signals of UART are shown in Table 2.18. The I/O buffers for SCL and SDA should be prepared in upper layer as shown in Figure 2.7.

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clock	
AHB	input		S_HSEL	AHB Lite Slave Select	ignored
AHB	input	[1:0]	S_HTRANS	AHB Lite Slave Transfer Type	
AHB	input		S_HWRITE	AHB Lite Slave Write	
AHB	input		S_HMASTERLOCK	AHB Lite Slave Locked Transfer	ignored
AHB	input	[2:0]	S_HSIZE	AHB Lite Slave Access Size	
AHB	input	[2:0]	S_HBURST	AHB Lite Slave Burst Access	ignored
AHB	input	[3:0]	S_HPROT	AHB Lite Slave Protection	ignored
AHB	input	[31:0]	S_HADDR	AHB Lite Slave Address	
AHB	input	[31:0]	S_HWDATA	AHB Lite Slave Write Data	
AHB	input		S_HREADY	AHB Lite Slave Ready Input	
AHB	output		S_HREADYOUT	AHB Lite Slave Ready Output	
AHB	output	[31:0]	S_HRDATA	AHB Lite Slave Read Data	
AHB	output		S_HRESP	AHB Lite Slave Response	always output 0
I2C	input		I2C_SCL_I	I2C SCL Input	
I2C	output		I2C_SCL_O	I2C SCL Output	
I2C	output		I2C_SCL_OEN	I2C SCL Output Enable	
I2C	input		I2C_SDA_I	I2C SDA Input	
I2C	output		I2C_SDA_O	I2C SDA Output	
I2C	output		I2C_SDA_OEN	I2C SDA Output Enable	
INT	output		IRQ_I2C	Interrupt Request	

Table 2.18: Input / Output Signals of I2C

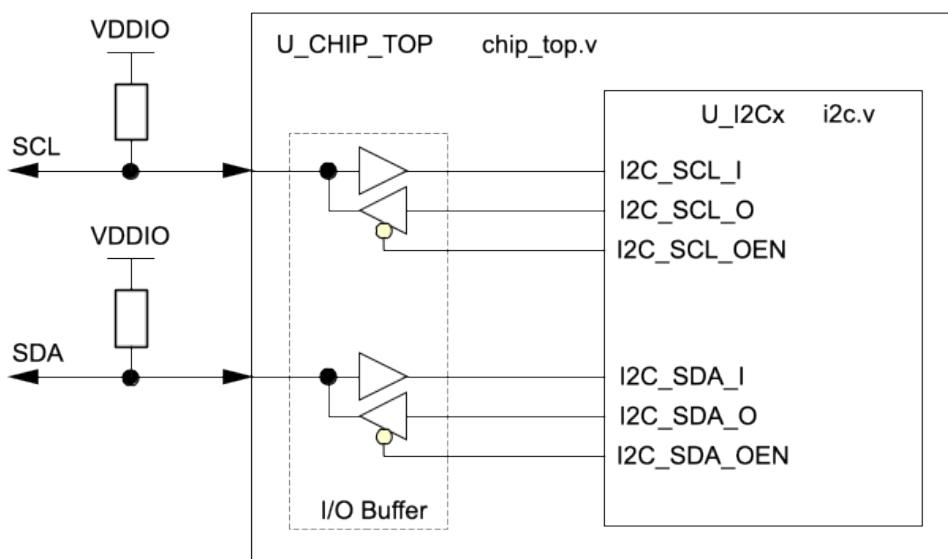


Figure 2.7: I/O Buffers for I2C

Control Registers

Controls registers of I2C are shown in Table 2.19 to Table 2.25. Please refer the technical document of the OpenCores I2C Master controller (<http://www.opencores.org/projects/i2c/>)

Memory Mapped Peripheral Base Address = 0xd0000000					
Offset Address	Name	Description			
0x0000	I2C_PRERL	I2C Clock Prescaler Low			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:0	PRERL	8'hff	R/W	Clock Prescaler Lower 8bits (PRERH, PRERL) = (System Clock Freq) / (5 * I2C Clock Freq) - 1	

Table 2.19: I2C_PRERL

Memory Mapped Peripheral Base Address = 0xd0000000					
Offset Address	Name	Description			
0x0004	I2C_PRERH	I2C Clock Prescaler High			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:0	PRERH	8'hff	R/W	Clock Prescaler Higher 8bits (PRERH, PRERL) = (System Clock Freq) / (5 * I2C Clock Freq) - 1	

Table 2.20: I2C_PRERH

Memory Mapped Peripheral Base Address = 0xd0000000					
Offset Address	Name	Description			
0x0008	I2C_CTL	I2C Control Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7	EN	1'b0	R/W	I2C Core Enable 0: Disable 1: Enable	
6	IEN	1'b0	R/W	I2C Interrupt Enable 0: Disavle 1: Enabke	
5:0	0	5'h0	R	Always 0.	

Table 2.21: I2C_CTL

Memory Mapped Peripheral Base Address = 0xd0000000					
Offset Address	Name	Description			
0x000c	I2C_TXR	I2C Transmit Data			
Bit	Field	Initial	Access	Description	Note
31:8	0	-	W	Always 0.	
7:1	TXD[7:1]	-	W	Transmit Data	
0	TXD[0]/RW	-	W	Transmit Data or RW bit (0: W, 1: R)	

Table 2.22: I2C_TXR

Memory Mapped Peripheral Base Address = 0xd0000000					
Offset Address	Name	Description			
0x000c	I2C_RXR	I2C Receive Data			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:0	RXD	8'h0	R	Receive Data	

Table 2.23: I2C_RXR

Memory Mapped Peripheral Base Address = 0xd0000000					
Offset Address	Name	Description			
0x0010	I2C_CR	I2C Command Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	-	W	Always 0.	
7	STA	-	W	Generate Start Condition (repeated, too)	
6	STO	-	W	Generate Stop Condition	
5	RD	-	W	Read from Slave	
4	WR	-	W	Write to Slave	Cleared Automatically
3	ACK	-	W	Send ACK or NACK to Receiver 0: Send ACK 1: Send NACK	
2:1	0	-	W	Always 0.	
0	IACK	-	W	Clear Pending Interrupt	Cleared Automatically

Table 2.24: I2C_CR

Memory Mapped Peripheral Base Address = 0xd0000000					
Offset Address	Name	Description			
0x0010	I2C_SR	I2C Status Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7	RXACK	1'b0	R	RXACK Status 0: Received Acknowledge from Slave 1: Not Received	
6	BUSY	1'b0	R	I2C Transaction Busy Status 0: After STOP 1: After START	
5	AL	1'b0	R	Arbitration Lost This bit is set when the core lost arbitration. Arbitration is lost when: • a STOP signal is detected, but non requested • The master drives SDA high, but SDA is low.	
4:2	0	3'b000	R	Always 0.	
1	TIP	1'b0	R	Transfer in progress 0: Transfer Complete 1: Transferring Data	
0	IF	1'b0	R	Interrupt Flag This bit is set when an interrupt is pending, which will cause a processor interrupt request if the IEN bit is set. The Interrupt Flag is set when: • one byte transfer has been completed • arbitration is lost	

Table 2.25: I2C_SR

2.14 Peripheral : SPI

Overview

This module is a simple master SPI interface based on the OpenCores SPI Interface (https://opencores.org/projects/simple_spi) by Richard Herveille. Please refer to the IP's document for more details. The SPI can generate interrupt requests upon completion of transfer blocks specified by the ICNT bits in the SPI_SPTR register. Please control the SPI Chip Select Pin level by directly manipulating the SPI_SPCS register as if it were a GPIO output port.

Input / Output Signals

Input / Output signals of SPI are shown in Table 2.26. The SPI_MOSI signal always drives 0 or 1 (not 3-state) because the SPI has only master mode.

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clock	
AHB	input		S_HSEL	AHB Lite Slave Select	ignored
AHB	input	[1:0]	S_HTRANS	AHB Lite Slave Transfer Type	
AHB	input		S_HWRITE	AHB Lite Slave Write	
AHB	input		S_HMASTLOCK	AHB Lite Slave Locked Transfer	ignored
AHB	input	[2:0]	S_HSIZEx	AHB Lite Slave Access Size	
AHB	input	[2:0]	S_HBURST	AHB Lite Slave Burst Access	ignored
AHB	input	[3:0]	S_HPROT	AHB Lite Slave Protection	ignored
AHB	input	[31:0]	S_HADDR	AHB Lite Slave Address	
AHB	input	[31:0]	S_HWDATA	AHB Lite Slave Write Data	
AHB	input		S_HREADY	AHB Lite Slave Ready Input	
AHB	output		S_HREADYOUT	AHB Lite Slave Ready Output	
AHB	output	[31:0]	S_HRDATA	AHB Lite Slave Read Data	
AHB	output		S_HRESP	AHB Lite Slave Response	always output 0
SPI	output	[3:0]	SPI_CSx	SPI Chip Selects	
SPI	output		SPI_SCK	SPI Clock	
SPI	output		SPI_MOSI	SPI Master Output / Slave Input	
SPI	input		SPI_MISO	SPI Master Input / Slave Output	
INT	output		IRQ_SPI	Interrupt Request	

Table 2.26: Input / Output Signals of SPI

Control Registers

Controls registers of SPI are shown in Table 2.27 to Table 2.31. Please refer the technical document of the OpenCores SPI Interface (https://opencores.org/projects/simple_spi).

Memory Mapped Peripheral Base Address = 0xe0000000					
Offset Address	Name	Description			
0x0000	SPI_SPCR	SPI Control Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7	SPIE	1'b0	R/W	SPI Interrupt Enable 0: Disable 1: Enable	
6	SPE	1'b0	R/W	SPI Core Enable 0: Disable 1: Enable	
5	0	1'b0	R	Always 0.	
4	MSTR	1'b1	R	Master Mode Select 0: Slave Mode 1: Master Mode (This bit is always fixed to 1.)	
3	CPOL	1'b0	R/W	Clock Polarity 0: SCK=Low during Transaction IDLE 1: SCK=High during Transaction IDLE	
2	CPHA	1'b0	R/W	Clock Phase 0: MOSI changes at 1 st edge of SCK, 3 rd edge of SCK, ... 1: MOSI changes at 2 nd edge of SCK, 4 th edge of SCK, ...	
1:0	SPR	2'b00	R/W	SPI Clock (SCK) Rate Select, it is used with EPSR bit in SPI_SPER EPSR=00, SPR=00 : Clock Rate = CLK/2 EPSR=00, SPR=01 : Clock Rate = CLK/4 EPSR=00, SPR=10 : Clock Rate = CLK/16 EPSR=00, SPR=11 : Clock Rate = CLK/32 EPSR=01, SPR=00 : Clock Rate = CLK/8 EPSR=01, SPR=01 : Clock Rate = CLK/64 EPSR=01, SPR=10 : Clock Rate = CLK/128 EPSR=01, SPR=11 : Clock Rate = CLK/256 EPSR=10, SPR=00 : Clock Rate = CLK/512 EPSR=10, SPR=01 : Clock Rate = CLK/1024 EPSR=10, SPR=10 : Clock Rate = CLK/2048 EPSR=10, SPR=11 : Clock Rate = CLK/4096 Others are reserved.	

Table 2.27: I2C_SPCR

Memory Mapped Peripheral Base Address = 0xe0000000					
Offset Address	Name	Description			
0x0004	SPI_SPSR	SPI Status Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7	SPIF	1'b0	R/W	SPI Interrupt Flag The Serial Peripheral Interrupt Flag is set upon completion of a transfer block. If SPIF is asserted ('1') and SPIE is set, an interrupt is generated.	Write 1 to Clear
6	WCOL	1'b0	R/W	Write Collision Flag The Write Collision flag is set when the Serial Peripheral Data register is written to, while the Write FIFO is full.	Write 1 to Clear
5:4	0	2'b00	R	Always 0.	
3	WFFULL	1'b0	R	Write FIFO Full	
2	WFEMPTY	1'b1	R	Write FIFO Empty	
1	RFFULL	1'b0	R	Read FIFO Full	
0	RFEMPTY	1'b1	R	Read FIFO Empty	

Table 2.28: I2C_SPSR

Memory Mapped Peripheral Base Address = 0xe0000000					
Offset Address	Name	Description			
0x0008	SPI_SPDR	SPI Data Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:0	DATA	8'hxx	R/W	When Write, the Data is written in Write FIFO When Read, the Data is read from Read FIFO	

Table 2.29: I2C_SPDR

Memory Mapped Peripheral Base Address = 0xe0000000					
Offset Address	Name	Description			
0x000c	SPI_SPER	SPI Extension Register			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:6	ICNT	2'b00	R/W	Interrupt Count 00 : SPIF is set after every completed transfer 01 : SPIF is set after every two completed transfers 10 : SPIF is set after every three completed transfers 11 : SPIF is set after every four completed transfers	
5:2	0	4'b0000	R	Always 0.	
1:0	EPSR	2'b00	R/W	Extended SPI Clock Rate Select Used with SPR bits in SPI_SPCR	

Table 2.30: I2C_SPER

Memory Mapped Peripheral Base Address = 0xe0000000					
Offset Address	Name	Description			
0x0010	SPI_SPCS	SPI Chip Select Pin			
Bit	Field	Initial	Access	Description	Note
31:8	0	24'h0	R	Always 0.	
7:0	SPCS	8'hff	R/W	SPI Chip Select Pin Level Lower 4bits of SPCS are used. SPCS[0] controls CPI_CS[0]. SPCS[1] controls CPI_CS[1]. SPCS[2] controls CPI_CS[2]. SPCS[3] controls CPI_CS[3].	

Table 2.31: I2C_SPCS

2.15 Peripheral : GPIO

Overview

GPIO (module name PORT) has 3 x 32bit input / output ports. Each pin direction can be configured respectively.

Input / Output Signals

Input / Output signals of GPIO (PORT) are shown in Table 2.32. The I/O buffers for each GPIO signal are implemented in the GPIO module.

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clock	
AHB	input		S_HSEL	AHB Lite Slave Select	ignored
AHB	input	[1:0]	S_HTRANS	AHB Lite Slave Transfer Type	
AHB	input		S_HWRITE	AHB Lite Slave Write	
AHB	input		S_HMASTLOCK	AHB Lite Slave Locked Transfer	ignored
AHB	input	[2:0]	S_HSIZE	AHB Lite Slave Access Size	
AHB	input	[2:0]	S_HBURST	AHB Lite Slave Burst Access	ignored
AHB	input	[3:0]	S_HPROT	AHB Lite Slave Protection	ignored
AHB	input	[31:0]	S_HADDR	AHB Lite Slave Address	
AHB	input	[31:0]	S_HWDATA	AHB Lite Slave Write Data	
AHB	input		S_HREADY	AHB Lite Slave Ready Input	
AHB	output		S_HREADYOUT	AHB Lite Slave Ready Output	
AHB	output	[31:0]	S_HRDATA	AHB Lite Slave Read Data	
AHB	output		S_HRESP	AHB Lite Slave Response	always output 0
GPIO	inout	[31:0]	GPIO0	32bit GPIO0	
GPIO	inout	[31:0]	GPIO1	32bit GPIO1	
GPIO	inout	[31:0]	GPIO2	32bit GPIO2	

Table 2.32: Input / Output Signals of GPIO(PORT)

Control Registers

Controls registers of GPIO (PORT) are shown in Table 2.33 to Table 2.38. The PDDn configures the direction of each pin; 0 for input, 1 for output. When a port bit is configured as an input, the read value of the data register PDRn corresponds to the input signal level. When a port bit is configured as an output, the read value of the data register PDRn is the register value, and the write value is stored in the data register PDRn and it is output to the corresponding external pin.

Memory Mapped Peripheral Base Address = 0xa0000000					
Offset Address	Name	Description			
Bit	Field	Initial	Access	Description	Note
31:0	pdr0	-	R/W	Port 0 Data Each bit corresponds to each pin. Input Port : Read Pin Level, Write PDR (Port Data Register) Output Port : Read PDR, Write PDR	

Table 2.33: PDR0

Memory Mapped Peripheral Base Address = 0xa0000000					
Offset Address	Name	Description			
0x0004	PDR1	Port Data Register 1			
Bit	Field	Initial	Access	Description	Note
31:0	pdr1	-	R/W	Port 1 Data Each bit corresponds to each pin. Input Port : Read Pin Level, Write PDR (Port Data Register) Output Port : Read PDR, Write PDR	

Table 2.34: PDR1

Memory Mapped Peripheral Base Address = 0xa0000000					
Offset Address	Name	Description			
0x0008	PDR2	Port Data Register 2			
Bit	Field	Initial	Access	Description	Note
31:0	pdr2	-	R/W	Port 2 Data Each bit corresponds to each pin. Input Port : Read Pin Level, Write PDR (Port Data Register) Output Port : Read PDR, Write PDR	

Table 2.35: PDR2

Memory Mapped Peripheral Base Address = 0xa0000000					
Offset Address	Name	Description			
0x0010	PDD0	Port Data Direction 0			
Bit	Field	Initial	Access	Description	Note
31:0	pdd0	32'h0	R/W	Port 0 Data Direction Each bit corresponds to each pin. 0: Input Port 1: Output Port	

Table 2.36: PDD0

Memory Mapped Peripheral Base Address = 0xa0000000					
Offset Address	Name	Description			
0x0014	PDD1	Port Data Direction 1			
Bit	Field	Initial	Access	Description	Note
31:0	pdd1	32'h0	R/W	Port 1 Data Direction Each bit corresponds to each pin. 0: Input Port 1: Output Port	

Table 2.37: PDD1

Memory Mapped Peripheral Base Address = 0xa0000000					
Offset Address	Name	Description			
0x0018	PDD2	Port Data Direction 2			
Bit	Field	Initial	Access	Description	Note
31:0	pdd2	32'h0	R/W	Port 2 Data Direction Each bit corresponds to each pin. 0: Input Port 1: Output Port	

Table 2.38: PDD2

2.16 Using Miscellaneous Features of mmRISC-1

JTAG Data Register for User

This feature is explained in Section 1.29, but it is not used in the SoC. The JTAG_DR_USER_OUT from mmRISC is directly looped back to the JTAG_DR_USER_IN with inversion in order to verify the output and input of the signals.

Halt on Reset

This feature is explained in Section 1.30. The detailed operations of Halt-on-Reset in the SoC are illustrated in Table 2.39. Regardless of JTAG or cJTAG, if you push the KEY1 on the FPGA board when releasing a push switch KEY0 (RES_N), the CPU enters a halt state and does not execute instructions. Once the debugger issues a resume command, the CPU resumes its instruction execution.

When you use 4-wire JTAG, an inverted signal of GPIO2[9](KEY1) is directly connected to FORCE_HALT_ON_RESET_REQ of the mmRISC-1 core.

When you use 2-wire cJTAG, if the module CJTAG_2_JTAG detects that the TMSC is set to low when reset is released, the module asserts FORCE_HALT_ON_RESET_REQ.

cJTAG

This feature is explained in Section 1.31. If you use conventional 4-wire JTAG for the debug interface, please set the signal level of "enable_cjtag" to low in chip_top_wrap.v. On the other hand, if you use 2-wire cJTAG interface for the debug interface, please set the signal level of "enable_cjtag" to high. The signal "enable_cjtag" is connected to GPIO2[6] corresponding to SW6 on DE10-Lite board in order to test both cases.

In the case of using 2-wire cJTAG, the cJTAG Implementation for Silicon (SoC) is shown in Figure 1.20. At the same layer of mmRISC, a converter from cJTAG to JTAG (CJTAG_2_JTAG) is instantiated. In the I/O buffer block, the TCKC should have pull-up function, and the TMSC should have both pull-up and bus-keeper functions, as described in Section 1.31.2 which are supported by the CJTAG_2_JTAG block. The method to control the I/O Port for cJTAG Pins is shown in Listing 1.5.

An example of a cJTAG Implementation for FPGA and Test Bench is shown in Figure 1.21. In this system, the CHIP_TOP layer, which is the SoC top layer, is surrounded by an additional upper layer CHIP_TOP_WRAP. The cJTAG Adapter described in Section 1.31.4 is located as CJTAG_ADAPTER in the layer CHIP_TOP_WRAP. The cJTAG external pin controls described in Section 1.31.2 are also performed in the layer CHIP_TOP_WRAP.

If you want to realize a cJTAG Adapter which completely follows the cJTAG standard specification as described in the last paragraph of Section 1.31.4, the CJTAG_ADAPTER block should receive reset and clock from the layer CHIP_TOP_WRAP and watch the cJTAG protocol sequence like the CJTAG_2_JTAG block does.

Security (Authentication)

This feature is explained in Section 1.32. As shown in Figure 1.22, there are three input signals related to security. In the SoC, the security feature is controlled by a switch on the FPGA board. The DEBUG_SECURE is connected to GPIO2[9] which is the SW9, so when the switch is ON, the DEBUG_SECURE becomes 1. The DEBUG_SECURE_CODE_0[31:0] is fixed to 32'h12345678, and the DEBUG_SECURE_CODE_1[31:0] is fixed to 32'hbeefcafe, respectively. To debug the

secure SoC, you should write "riscv autodata_write" command in the OpenOCD configuration file as shown in Listing 1.6.

Switch (defines_chip.v)	JTAG Mode	Operation of Halt-on-Reset
`USE _FORCE _HALT_ON _RESET	signal level of "enable_cjtag" in chip_top_wrap.v	
OFF	LOW	JTAG (4-wire) N/A (The CPU always runs after reset.)
OFF	HIGH	cJTAG (2-wire) N/A (The CPU always runs after reset.)
ON	LOW	JTAG (4-wire) When the reset (res_org) is deasserted, if GPIO2[10] (a push switch KEY1 on the DE10-Lite FPGA board) is at a low level, the FORCE_HALT_ ON_RESET_REQ, which is an input of mmRISC-1 core, becomes 1. The FORCE_HALT_ON_RESET _REQ will be cleared when FORCE_HALT_ON_R ESET_ACK is asserted. This means, if you push the KEY1 on the FPGA board when releasing a push switch KEY0 (RES_N), the CPU enters a halt state and does not execute instructions. Once the debugger issues a resume command, the CPU starts its instruction execution.
ON	HIGH	cJTAG (2-wire) When the reset (res_org) is deasserted, if the input level of TMSC is at a low level, the FORCE_HALT _ON_RESET_REQ is asserted by the cjtag_2_jtag.v. The FORCE_HALT_ON_RESET _REQ will be cleared when FORCE_HALT_ON_R ESET_ACK is asserted. The cjtag_adapter.v in the top layer of the SoC, which is chip_top_wrap.v, sets the TMSC to 0 when the RESET_HALT_N, which is connected to GPIO2[10] (a push switch KEY1 on the DE10-Lite FPGA board), is at a low level, the TMSC becomes 0. This means, if you push the KEY1 on the FPGA board when releasing a push switch KEY0 (RES_N), the CPU enters a halt state and does not execute instructions. Once the debugger issues a resume command, the CPU starts its instruction execution.

Table 2.39: Operations of Halt-on-Reset in the SoC

Low Power Mode

This feature is explained in Section 1.33. In the SoC, when an input signal of STBY_REQ which is connected to SW8 on the FPGA board is high, the chip top layer requests the mmRISC-1 core to enter into the stand-by mode, and a stand-by acknowledge signal is connected to an output signal STBY_ACK_N which is connected to HEX57 which is the decimal point of 7-segment LED "HEX5". In the SoC, the system clock is not stopped during stand-by mode, therefore, the debug

operations via JTAG or cJTAG can be accepted.

Slow Clock

To verify that the any magnitude of the ratio between the system clock (clk) frequency and the JTAG/cJTAG clock (TCK/TCKC) frequency is acceptable in the mmRISC-1 core, the system clock can be selected from fast (16MHz or 20MHz) or slow (100KH) by setting level of GPIO2[7] at any time. If the GPIO2[7] is low level, the system clock becomes fast. Details are described in Section [2.5](#).

Chapter 3

mmRISC-1 Simulations and Verifications

This chapter describes several simulation and verification methods of mmRISC-1 SoC.

3.1 Development Environment and Tools

Please prepare the development environment and tools shown in Table 3.1. You can obtain the logic simulation tool Questa by installing Intel Quartus Prime. Please build and install the GNU tool chains and the OpenOCD from their respective GitHub repositories. Please add the locations of all executable binaries to the environment variable \$PATH. Details of USB-JTAG interface hardware are described later in this document.

Item	Description	Note
Platform	Ubuntu 64bit for x86-64	
Simulator	Mentor Questa Intel FPGA Starter Edition	Included in Questa Prime
Compliance Test	https://github.com/riscv-non-isa/riscv-arch-test	
Unit Tests	https://github.com/riscv-software-src/riscv-tests	
FPGA Board	Terasic DE10-Lite Board	
FPGA Tool	Intel Quartus Prime Lite Edition	
C/C++ IDE	Eclipse IDE for Embedded C/C++ Developers	
GNU Tool Chain	https://github.com/riscv-collab/riscv-gnu-toolchain	
OpenOCD	https://github.com/riscv/riscv-openocd	
USB-JTAG	Handmade Board or Olimex ARM-USB-OCD(H)	Section 4.2

Table 3.1: Development Environment and Tools

Notes on ARM based environment

When you use an ARM-based macOS machine such as a MacBook Pro with Apple Silicon M-series, please install Parallels Desktop for Mac as a virtual PC environment first. Then, install both Ubuntu for 64-bit ARM and Windows for 64-bit ARM on the virtual machine. Please use each tool properly as shown in Table 3.2. I recommend you to share the same design files on macOS with Ubuntu, Windows and macOS during your design work.

Operation	Platform	Tool	Note
Software Development	Ubuntu for 64bit ARM	- Eclipse - GNU Tool Chain - OpenOCD	
Logic Simulation	Windows for 64bit ARM	- Mentor Questa Intel FPGA Starter Edition	Included in Questa Prime
FPGA Synthesis and P&R	Windows for 64bit ARM	- Intel Quartus Prime Lite Edition	
FPGA Configuration	macOS for ARM	- UrJTAG	Convert *.sof or *.pof to *.svf file by the Programmer Tool in Intel Quartus and download it by using the "jtag" command in UrJTAG on macOS terminal.

Table 3.2: Development Tools on ARM macOS

3.2 Github Repository Files

You can get the full set of related files of mmRISC-1 from GitHub by using following command.

```
$ git clone https://github.com/munetomo-maruyama/mmRISC-1.git
```

Major files in the repository are shown in Table 3.3.

Group	Directory	File / Directory Name	Description	Note
Document	mmRISC-1/doc	mmRISC-1_TRM_RevXX.pdf latex_files	Technical Reference Manual LaTeX Source Files to make Technical Reference Manual	
RTL	mmRISC-1/verilog/*	*.v	RTL for mmRISC-1 Core and Tiny SoC	
Simulation	mmRISC-1/simulation /modelsim /mmRISC_Simulation	tb_TOP.v sim_TOP.do rom.memh	Test Bench of mmRISC-1 Tiny SoC Command Script of ModelSim Initial Data of Instruction RAM generated by hex2v command	
		do_riscv-arch-test.py flist	Python Script to simulate RISC-V Compliance Test “riscv-arch-test” for I/C/M/Zifence/Privileged File list of RTL codes	
		tb_TOP.v sim_TOP.do ./riscv-arch-test/ ./riscv-arch-test/work/rv32_m/	Test Bench of mmRISC-1 Tiny SoC Command Script of Questa Directory of RISC-V Architectural Testing Framework Directory of Build Binary Files (*.elf) and Expected Signature Output Files (*.signature.output)	
		do_riscv-tests.py flist	Python Script to simulate unit tests of RISC-V Processors “riscv-tests” including Atomic and Floating File list of RTL codes	
		tb_TOP.v sim_TOP.do ./riscv-tests/work/isa/RV32*/	Test Bench of mmRISC-1 Tiny SoC Command Script of Questa Directory of Build Binary Files (*)	
		hex2v.c hex2mif.c	Converter from Intel Hex format to Initial RAM data for verilog \$readmemh() Converter from Hex format to Initial RAM data for FPGA.	
		./src/*c, *h, *S link.ld .cproject, .projct	C Source Files, C Header Files and Startup Assembler Source File for each Application Linker Script Eclipse Project Files	
		./src_app/inc/*.h ./src_app/src/*c, *.S ./src_kernel/inc/*.h ./src_kernel/src/*c, *.S link.ld .cproject, .projct	C Source Files, C Header Files and Startup Assembler Source File for FreeRTOS Blinky Application C Source Files, C Header Files and Startup Assembler Source File for FreeRTOS Kernel Linker Script Eclipse Project Files	
		openocd.cfg openocd_cjtag.cfg openocd_olimex.cfg openocd_rpi.cfg	OpenOCD JTAG Configuration File for FTDI FT2232D OpenOCD cJTAG Configuration File for FTDI FT2232D OpenOCD JTAG Configuration File for Olimex ARM-USB-OCD-H OpenOCD JTAG Configuration File for Raspberry Pi 4 Model B	
		mmRISC.pqf PLL.* RAM128KB_DP.* mmRISC.sdc ./output_files	Quartus Prime Project File PLL IP Files Block RAM (2-port) for Instruction RAM (*.mif is initialization data) Design Constraints File Directory for Generated Files	

Table 3.3: Major Files in GitHub Repository

3.3 RTL Verification Methods

3.3.1 Vector Simulation

A directory named “mmRISC-1/simulation/modelsim/mmRISC_Simulation” contains a sample set of vector simulation environments. The directory name "modelsim" refers to the predecessor application of Questa for logic simulation.

(a) Common Preparation (Do Once)

Before you try a simulation, please build "tools/hex2v.c" and "tools/hex2mif.c" once by using the following command, and add the locations of the generated executable binaries "hex2v" and "hex2mif" to the environment variable \$PATH. If your environment is Windows, use MinGW-64 or other appropriate compiling tools for standard C programs.

```
$ gcc -o hex2v hex2v.c
$ gcc -o hex2mif hex2mif.c
```

(b) Preparation of Script Command for Questa

Please edit a command script file "sim_TOP.do" if necessary. The default script instructs the following operations.

- To convert an Intel Hex binary program code “*.hex” to "rom.memh", which initializes an instruction RAM by \$readmemh() in the RTL code, use the command “hex2v” prepared above. Please edit the binary code file name if you want to change the codes to be stored in the instruction RAM.
- To compile each RTL file for verification by the command “vlog” . Please edit the RTL file list if you change the structure of RTL files.
- To launch the Simulator by the command “vsim” .
- To add Signals to display in the Wave Window by using Questa internal command “add wave” . Please edit the command list if you want to change the waveform display.
- To start Logic Simulation by the Questa internal command “run” .
- The RTL codes of the Instruction RAM and the Data RAM infer FPGA Block RAM IP and do not explicitly include the FPGA IP. If you want to include FPGA Block RAM IP with initialized data for the Instruction RAM, please add the option "+define+FPGA" in the vlog command. This macro switches ram.v to ram_fpga.v for the Instruction RAM (RAM1) in the chip_top layer. The ram_fpga.v refers to the FPGA IP of RAM128KB_DP.v stored in the directory “fpga” . The initialization data is specified by RAM128KB_DP.mif which can be generated by using the following command.

```
$ ./tools/hex2mif ***.hex > RAM128KB_DP.mif
```

Note that the Block RAM is made by a 2-Port type because the AHB write data period is after the active period of the address and strobes.

(c) Preparation of RTL for Test Bench

Please edit the RTL code of the test bench "tb_TOP.v", if necessary. In the test bench, the following features are supported.

- Generation of Reset and Clock - Instantiation of CHIP_TOP_WRAP (chip_top_wrap.v)
- Simulation Time Out (`TB_STOP) - Initialization of nodes in the Power-On-Reset circuit.
- Simulation Stop Condition (Both reaching the end of the stimulus or writing to a Memory at address = 0xfffffffffc with wdata = 0xdeaddead)
- Dumping out instruction Decode Sequences to "dump.txt" (if `DUMP_ID_STAGE is defined)
- Tasks for JTAG/cJTAG Access (JTAG Reset, Shift-In/Out IR/DR, DMI Write and Read); If `define ENABLE_CJTAG is OFF, normal 4-wire JTAG operations are enabled. If it is ON, 2-wire cJTAG operations based on RISC-V OpenOCD are enabled.
- Stimulus for the debugging operations (if `JTAG_OPERATION is defined) : Read IDCODE, JTAG Reset, Debugger Command for Stop/Step/Run).
- If you want to insert several wait cycles in Instruction RAM access and Data RAM access, define ``RAM_WAIT'' in "defines_chip.v". The wait cycles are determined by bit [3:2] of the access address in "ram.v" logic.
- If you want to implement multiple (four) harts in mmRISC-1 and let them intervene on bus access among them, define ``MULTI_HART'' in "define_chip.v". Each hart has different reset vector, but all harts fetch instruction codes from the same instruction RAM due to shadowed addressing.

(d) Run the Logic Simulation

Set the current directory to "mmRISC-1/simulation/modelsim/mmRISC_Simulation", and launch Questa by using the "vsim" command. In the Transcript window, enter the command "do sim_TOP.do" to execute the command script.

(e) Check Waveforms and Dump File (if any)

Hopefully, you will get successful results ! A Questa Screen Shot is shown in Figure 3.1.

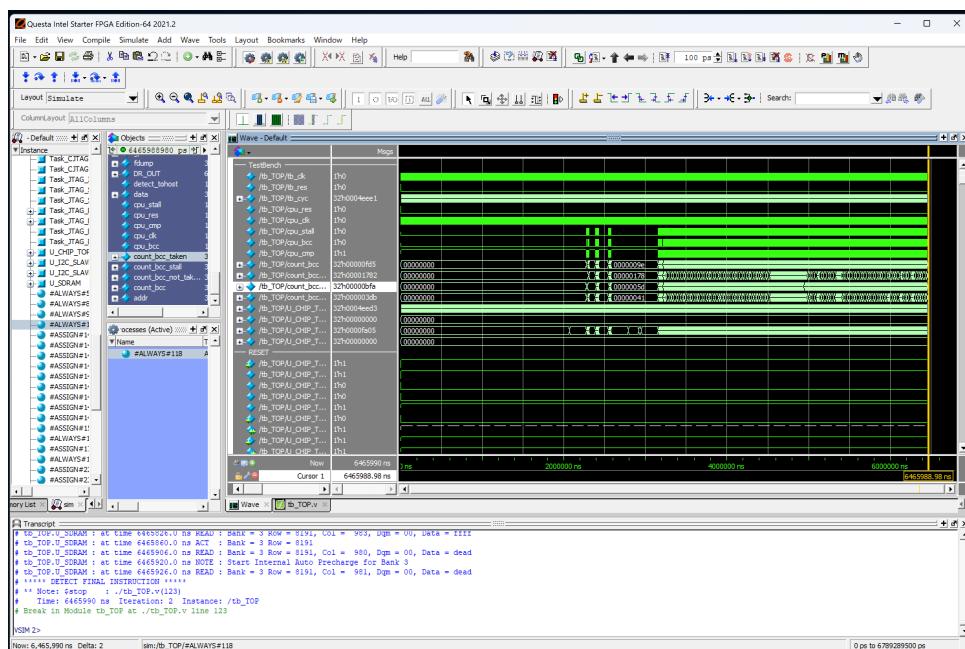


Figure 3.1: Questa Screen Shot

3.3.2 RISC-V Compliance Test "riscv-arch-test" for ISA of I/C/M/Zicsr/Zifence/Privileged

The directory "mmRISC-1/simulation/modelsim/riscv_arch_test" contains a sample set of compliance test resources for I/C/M/Zicsr/Zifence/Privileged instructions which are called "riscv-arch-test".

(a) Preparation of "riscv-arch-test"

The mmRISC-1 repository includes completed test binaries and signatures. However, if you want to prepare them by yourself, you need to install the RISC-V ISA Simulator "Spike" in advance. Then, you need to download "riscv-arch-test" from Github as shown in Table 3.1. After that, you need to follow its documentation and run the makefile to generate elf-binaries with disassembly lists. Next, you need to simulate them to generate expected signatures. You should place all the elf-binaries, disassembly lists and signatures under the directory "riscv-arch-test/work/rv32i_m/*" and classify them by each ISA group. Finally, you should move to the directory "riscv-arch-test" under the directory "mmRISC-1/simulation/modelsim/riscv_arch_test".

(b) Running Simulations

A python script "do_riscv_arch_test.py" is prepared to execute the test automatically. It simulates the mmRISC-1 Tiny SoC with four different settings: (1) No RAM Wait Cycles + 1 Hart, (2) No RAM Wait Cycles + 4 Harts, (3) With RAM Wait Cycles + 1 Hart, and (4) With RAM Wait Cycles + 4 Harts. It takes each instruction sequence binary (elf) from "riscv-arch-test/riscv-arch-test/work/rv32i_m/*/*.elf", and simulates it. Each program code stores signature data in the address range from <begin_signature> to <end_signature>, and stores 0x00000001 in the address <tohost> at the end of the program. These <*> parameters are available in the corresponding disassembly list ".objdump" for each binary code. After the simulation is finished, the RTL test bench writes the stored data from <begin_signature> to <end_signature> into the dump list file "dump.signature". The python script then compares the dump with the prepared expected signatures. If they are identical, the python script proceeds to simulate the next program code. Otherwise, the python script terminates. You can see reports like those in Listing 3.1.

```
===== [Test 1] ===== [BUS_INTERVENTION_01] =====
./riscv-arch-test/work/rv32i_m/privilege/misalign-blt-01.elf
Dump Begin 32'h90002080
Dump End   32'h90002190
To Host    32'h90001000
===== [Test 2] ===== [BUS_INTERVENTION_01] =====
./riscv-arch-test/work/rv32i_m/privilege/misalign-lh-01.elf
Dump Begin 32'h90002080
Dump End   32'h90002190
To Host    32'h90001000
===== [Test 3] ===== [BUS_INTERVENTION_01] =====
./riscv-arch-test/work/rv32i_m/privilege/misalign-beq-01.elf
Dump Begin 32'h90002080
Dump End   32'h90002190
To Host    32'h90001000
===== [Test 4] ===== [BUS_INTERVENTION_01] =====
./riscv-arch-test/work/rv32i_m/privilege/ecall.elf
Dump Begin 32'h90002070
Dump End   32'h90002090
To Host    32'h90001000
...
```

Listing 3.1: Output of "riscv-arch-test"

3.3.3 RISC-V Unit Test "riscv-tests" including Atomic and Floating Point ISA

The directory "mmRISC-1/simulation/modelsim/riscv_tests" contains a sample set of unit tests, called "riscv-tests", that includes Atomic and Floating Point ISA.

(a) Preparation of "riscv_tests"

The mmRISC-1 repository includes completed test binaries. However, if you want to prepare them by yourself, you need to download "riscv-tests" from Github as shown in Table 3.1. After that, you need to follow its documentation and run the makefile to generate elf-binaries and disassemble lists. You should place all the elf-binaries and disassemble lists under the directory "riscv-tests/work/isa/*" and classify them by each ISA group. Finally, you should move to the directory "riscv-test" under the directory "mmRISC-1/simulation/modelsim/riscv_tests".

(b) Running Simulations

A python script "do_riscv_tests.py" is prepared to run the test automatically. It simulates mmRISC-1 Tiny SoC with four different settings: (1) No RAM Wait Cycles + 1 Hart, (2) No RAM Wait Cycles + 4 Harts, (3) With RAM Wait Cycles + 1 Hart, and (4) With RAM Wait Cycles + 4 Harts. It takes each instruction sequence binary (elf) from "riscv-tests/riscv-tests/work/isa/RV32*/*" and simulates it. In the script file, you need to specify the directory according to the ISA you want to verify. For example, specify RV32IMC directory for RV32IMC, RV32IMAC directory for RV32A, and RV32IMAFC directory for RV32F and RV32FC. Each program code stores 0x00000001 in address <tohost> at the success point of the program, or stores different data in address <tohost> if it fails. You can find the <tohost> parameter in the corresponding disassembly list "*.dump" for each binary code. After the simulation finishes, the RTL test bench writes 'result.txt' that shows "PASS" or "FAIL" as a text. The python script then checks the result text, and if it is PASS, it proceeds to simulate the next program code. Otherwise, it terminates. You can see a report like the one in Listing 3.2.

```
===== [Test 1] ===== [BUS_INTERVENTION_01] =====
./riscv-tests/work/isa/RV32IMFC/rv32uf -p-fcvt_w
To Host 32'h90001000
===== [Test 2] ===== [BUS_INTERVENTION_01] =====
./riscv-tests/work/isa/RV32IMFC/rv32uf -p-fclass
To Host 32'h90001000
===== [Test 3] ===== [BUS_INTERVENTION_01] =====
./riscv-tests/work/isa/RV32IMFC/rv32uf -p-fadd
To Host 32'h90001000
===== [Test 4] ===== [BUS_INTERVENTION_01] =====
./riscv-tests/work/isa/RV32IMFC/rv32uf -p-ldst
To Host 32'h90001000
===== [Test 5] ===== [BUS_INTERVENTION_01] =====
./riscv-tests/work/isa/RV32IMFC/rv32uf -p-move
To Host 32'h90001000
...
```

Listing 3.2: Output of "riscv-tests"

3.4 Verification of Floating Point Operations

To verify the calculation results of the CPU's floating-point related instructions, we need a large number of test cases and their expected values. These consist of combinations of calculation types, various formats such as normal/subnormal/zero/infinite/NaN numbers, and multiple rounding methods. For example, mmRISC-1 used the following methods:

- Preparation of Test Cases by using Berkeley TestFloat
- Verification of all Test Cases by using FPGA system

3.4.1 Preparation of Test Cases by using Berkeley TestFloat

(a) Installation of Berkley Float

You need to download and build the Berkeley SoftFloat and the Berkeley TestFloat before proceeding. You can find them in the following links:

<https://github.com/ucb-bar/berkeley-softfloat-3.git>
<https://github.com/ucb-bar/berkeley-testfloat-3.git>

You also need to make sure that you have an executable binary file "testfloat_gen" in the directory "TestFloat-3e/build/.../". This file is a test-case-generator for floating-point operations.

(b) Generating Test Case Files

The directory "mmRISC-1/workspace" contains mmRISC-1 sample Eclipse projects. You can find a directory "mmRISC_Floating/testcase" in it. In this directory, you can make floating-point test cases by using the given scripts as shown in Table 3.4. Each script runs the test-case-generator executable binary "testfloat_gen", so you need to edit the correct location of the executable binary for your environment. To run each script and generate the corresponding test case text file, you need to enter the following command.

```
$ ./gen_testfloat_X
(X is A, B, D, F2S, F2U, M, N, Q, S2F or U2F)
```

(c) Structure of Test Case Files

The generated test case files for 1-operand, 2-operand, and 3-operand operations are shown in Listings 3.3, 3.4, and 3.5, respectively. Each file has test cases for five rounding modes, which are:

- RNE: Round to Nearest and tied to even
- RTZ: Round to Zero
- RDN: Round Down towards minus infinite
- RUP: Round Up towards plus infinite
- RMM: Round to Nearest, ties to Max Magnitude

Each test case starts with a 3-letter OPCODE that consists of a #, an operation indicator, and a rounding mode indicator as shown in Table 3.5. The OPCODE is followed by the test case body. The file has five groups of rounding modes.

The test case body for the 1-operand type has three columns, as shown in Listing 3.3. Each column represents a 32-bit or a 5-bit hexadecimal data. The meanings of each column are as follows: (1) The first column is the 32-bit input data, (2) The second column is the expected 32-bit result data, and (3) The third column is the expected 5-bits exception flag, which is explained in Table 3.6.

The test case body for the 2-operands type has four columns, as shown in Listing 3.4. Each column represents a 32-bit or a 5-bit hexadecimal data. The meanings of each column are as follows: (1) The first and second columns are the 32-bit input data, (2) The third column is the expected 32-bit result data, (3) The fourth column is the expected 5-bit exception flag, which is explained in Table 3.6.

The test case body for the 3-operands type has five columns, as shown in Listing 3.5. Each column represents a 32-bit or a 5-bit hexadecimal data. The meanings of each column are as follows: (1) The first, second, and third columns are the 32-bit input data, (2) The fourth column is the expected 32-bit result data, which is calculated as (first * second + third), (3) The fifth column is the expected 5-bit exception flag, which is explained in Table 3.6.

Script Name	Operation	Operand Counts	Generated Test Case File	Total Test Cases	Note
gen_testfloat_A	F32 = F32 + F32	2	testfloat_A	232,320	
gen_testfloat_B	F32 = F32 - F32	2	testfloat_B	232,320	
gen_testfloat_D	F32 = F32 / F32	2	testfloat_D	232,320	
gen_testfloat_F2S	SI32 = F32 (conversion)	1	testfloat_F2S	3,090	Manually added some cases.
gen_testfloat_F2U	UI32 = F32 (conversion)	1	testfloat_F2U	3,000	
gen_testfloat_M	F32 = F32 * F32	2	testfloat_M	232,320	
gen_testfloat_N	F32 = F32 * F32 + F32	3	testfloat_N	30,666,250	Too huge. Need to reduce.
gen_testfloat_Q	F32 = sqrt(F32)	1	testfloat_Q	3,000	
gen_testfloat_S2F	F32 = SI32 (conversion)	1	testfloat_S2F	1,860	
gen_testfloat_U2F	F32 = UI32 (conversion)	1	testfloat_U2F	1,860	

Table 3.4: Scripts for Generating Floating Point Test Cases

Note 1 : The test case file for 3-operands type is indeed very large. It contains more than 6,000,000 test cases for each rounding mode. You can reduce it by selecting some test cases at regular intervals using some tool such as sed or awk. For example, you can use the following command to extract every 250th line from the original file and save it as a new file:

sed -n '1 250p' original_file > new_file

This will reduce the number of test cases for each rounding mode, which is more manageable. You can also adjust the interval according to your preference.

Note 2 : For "testfloat_N" for multiplication and addition, there should be FMADD.S as well as FMSUB.S, FNMSUB.S and FNMADD.S. In the floating-point logic of mmRISC-1, the only differences among these four instructions are how they apply

the sign bit to each input data. Therefore, FMADD can represent the other instructions.

```
#SN
be804000 00000000 00
be804080 00000000 00
be000000 00000000 00
bc800000 00000000 00
...
#SZ
be804000 00000000 00
be804080 00000000 00
be000000 00000000 00
bc800000 00000000 00
...
#SD
be804000 ffffffff 00
be804080 ffffffff 00
be000000 ffffffff 00
bc800000 ffffffff 00
...
#SU
be804000 00000000 00
be804080 00000000 00
be000000 00000000 00
bc800000 00000000 00
...
#SM
be804000 00000000 00
be804080 00000000 00
be000000 00000000 00
bc800000 00000000 00
...
```

Listing 3.3: Example of Generated 1-Operand Test Case (testfloat_F2S)

```
#AN
8683F7FF C07F3FFF C07F3FFF 01
00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F 3E7F7F7F 01
DF7EFFFF 00000000 DF7EFFFF 00
...
#AZ
8683F7FF C07F3FFF C07F3FFF 01
00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F 3E7F7F7E 01
DF7EFFFF 00000000 DF7EFFFF 00
...
#AD
8683F7FF C07F3FFF C07F4000 01
00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F 3E7F7F7E 01
DF7EFFFF 00000000 DF7EFFFF 00
...
#AU
8683F7FF C07F3FFF C07F3FFF 01
00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F 3E7F7F7F 01
DF7EFFFF 00000000 DF7EFFFF 00
...
#AM
8683F7FF C07F3FFF C07F3FFF 01
00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F 3E7F7F7F 01
DF7EFFFF 00000000 DF7EFFFF 00
...
```

Listing 3.4: Example of Generated 2-Operand Test Case (testfloat_A)

```
#NN
8683F7FF C07F3FFF 00000000 07839504 01
00000000 00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F DF7EFFFF DF7EFFFF 01
4F951295 00000000 00000000 00000000 00
...
```

```

#NZ
8683F7FF C07F3FFF 00000000 07839504 01
00000000 00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F DF7EFFFF DF7EFFFF 01
4F951295 00000000 00000000 00000000 00
...
#ND
8683F7FF C07F3FFF 00000000 07839504 01
00000000 00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F DF7EFFFF DF7F0000 01
4F951295 00000000 00000000 00000000 00
...
#NU
8683F7FF C07F3FFF 00000000 07839505 01
00000000 00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F DF7EFFFF DF7EFFFF 01
4F951295 00000000 00000000 00000000 00
...
#NM
8683F7FF C07F3FFF 00000000 07839504 01
00000000 00000000 3C072C85 3C072C85 00
9EDE38F7 3E7F7F7F DF7EFFFF DF7EFFFF 01
4F951295 00000000 00000000 00000000 00
...

```

Listing 3.5: Example of Generated 3-Operand Test Case (testfloat_N)

Test Case File	Operation	OPCODE					Note
		RNE	RTZ	RDN	RUP	RMM	
testfloat_A	FADD.S	#AN	#AZ	#AD	#AU	#AM	
testfloat_B	FSUB.S	#BN	#BZ	#BD	#BU	#BM	
testfloat_D	FDIV.S	#DN	#DZ	#DD	#DU	#DM	
testfloat_F2S	FCVT.W.S	#SN	#SZ	#SD	#SU	#SM	
testfloat_F2U	FCVT.W.U.S	#UN	#UZ	#UD	#UU	#UM	
testfloat_M	FMUL.S	#MN	#MZ	#MD	#MU	#MM	
testfloat_N	FMADD.S	#NN	#NZ	#ND	#NU	#NM	
testfloat_Q	FSQRT.S	#QN	#QZ	#QD	#QU	#QM	
testfloat_S2F	FCVT.S.W	#sN	#sZ	#sD	#sU	#sM	
testfloat_U2F	FCVT.S.W.U	#uN	#uZ	#uD	#uU	#uM	

Table 3.5: OPCODEs in Test Case File

Bit	Symbol	Description	Note
bit4	NV	Invalid Exception	
bit3	DZ	Infinite Exception (Divide by Zero)	
bit2	OF	Overflow Exception	
bit1	UF	Underflow Exception	
bit0	NX	Inexact Exception	

Table 3.6: Exception Flag in Test Case File

3.4.2 Verification of all Test Cases by using FPGA system

There are too many test cases to verify by logic simulation, as it would take a long time. Therefore, we suggest that you prepare the FPGA system of mmRISC-1 as a hardware logic emulator for verification. You can find the details of the FPGA system in the next chapter.

You can use the C program in the directory "mmRISC-1/workspace/mmRISC_Floating/" to verify the test cases. The function "Test_Float()" in ./src/float.c performs the verification. You need to build and run the project, and send each test case text to UART RXD (8N1, 57600bps) with a terminal application. The function decodes OPCODEs (#xx) and executes each operation with test case inputs. It also compares the results and exception flags with the expected values. The terminal will receive verification results from UART TXD (8N1, 57600bps), as shown in Tables 3.6, 3.7, and 3.8. You should save the received ASCII data as a text file named "testfloat_X.out" with the terminal function. Each received line has five columns: the sent test case, the operation result data, the comparison result of the data ("OK" or "NG"), the exception result flag, and the comparison result of the flag ("ok" or "ng"). You need to check the output results to verify the floating-point operations.

Note 1 : The FDIV and FSQRT instructions use the Goldschmidt algorithm, which requires a convergence loop count. This count is specified in the CSR FCONV (0xbe0) and has a maximum value of 15 (0xf). This value ensures the accuracy of each result.

Note 2 : The output data may sometimes show "NG" for the values and "ng" for the flags. This means that there are some differences between the mmRISC-1 and the TestFloat in how they handle NaN and exception flags. For the "NG" values, the bit patterns are slightly different, but they are both valid NaN formats. For the "ng" flags, the exception handling policies are different.

```
#SN
be804000 00000000 00 00000000 OK 00 ok
be804080 00000000 00 00000000 OK 00 ok
be000000 00000000 00 00000000 OK 00 ok
bc800000 00000000 00 00000000 OK 00 ok
...
#SZ
be804000 00000000 00 00000000 OK 00 ok
be804080 00000000 00 00000000 OK 00 ok
be000000 00000000 00 00000000 OK 00 ok
bc800000 00000000 00 00000000 OK 00 ok
...
#SD
be804000 ffffffff 00 FFFFFFFF OK 00 ok
be804080 ffffffff 00 FFFFFFFF OK 00 ok
be000000 ffffffff 00 FFFFFFFF OK 00 ok
bc800000 ffffffff 00 FFFFFFFF OK 00 ok
...
#SU
be804000 00000000 00 00000000 OK 00 ok
be804080 00000000 00 00000000 OK 00 ok
be000000 00000000 00 00000000 OK 00 ok
bc800000 00000000 00 00000000 OK 00 ok
...
#SM
be804000 00000000 00 00000000 OK 00 ok
be804080 00000000 00 00000000 OK 00 ok
be000000 00000000 00 00000000 OK 00 ok
bc800000 00000000 00 00000000 OK 00 ok
...
```

Listing 3.6: Example Result of 1-Operand Test Case (testfloat_F2S)

```
#AN
8683F7FF C07F3FFF C07F3FFF 01 C07F3FFF OK 01 ok
00000000 3C072C85 3C072C85 00 3C072C85 OK 00 ok
9EDE38F7 3E7F7F7F 3E7F7F7F 01 3E7F7F7F OK 01 ok
```

```

DF7EFFFF 00000000 DF7EFFFF 00 DF7EFFFF OK 00 ok
...
#AZ
8683F7FF C07F3FFF C07F3FFF 01 C07F3FFF OK 01 ok
00000000 3C072C85 3C072C85 00 3C072C85 OK 00 ok
9EDE38F7 3E7F7F7F 3E7F7F7E 01 3E7F7F7E OK 01 ok
DF7EFFFF 00000000 DF7EFFFF 00 DF7EFFFF OK 00 ok
...
#AD
8683F7FF C07F3FFF C07F4000 01 C07F4000 OK 01 ok
00000000 3C072C85 3C072C85 00 3C072C85 OK 00 ok
9EDE38F7 3E7F7F7F 3E7F7F7E 01 3E7F7F7E OK 01 ok
DF7EFFFF 00000000 DF7EFFFF 00 DF7EFFFF OK 00 ok
...
#AU
8683F7FF C07F3FFF C07F3FFF 01 C07F3FFF OK 01 ok
00000000 3C072C85 3C072C85 00 3C072C85 OK 00 ok
9EDE38F7 3E7F7F7F 3E7F7F7F 01 3E7F7F7F OK 01 ok
DF7EFFFF 00000000 DF7EFFFF 00 DF7EFFFF OK 00 ok
...
#AM
8683F7FF C07F3FFF C07F3FFF 01 C07F3FFF OK 01 ok
00000000 3C072C85 3C072C85 00 3C072C85 OK 00 ok
9EDE38F7 3E7F7F7F 3E7F7F7F 01 3E7F7F7F OK 01 ok
DF7EFFFF 00000000 DF7EFFFF 00 DF7EFFFF OK 00 ok
...

```

Listing 3.7: Example Result of 2-Operand Test Case (testfloat_A)

```

#NN
00000000 80000004 407FFFFE 407FFFFE 00 407FFFFE OK 00 ok
00000000 57FFD7FF DA000084 DA000084 00 DA000084 OK 00 ok
3E1003FF 7F800024 5F7FEFFF 7FC00024 10 7FC00000 NG 10 ok
7FFFCOFE C59FF7FF 3FFFFFFE 7FFFCOFE 00 7FC00000 NG 10 ng
...
#NZ
00000000 80000004 407FFFFE 407FFFFE 00 407FFFFE OK 00 ok
00000000 57FFD7FF DA000084 DA000084 00 DA000084 OK 00 ok
3E1003FF 7F800024 5F7FEFFF 7FC00024 10 7FC00000 NG 10 ok
7FFFCOFE C59FF7FF 3FFFFFFE 7FFFCOFE 00 7FC00000 NG 10 ng
...
#ND
00000000 80000004 407FFFFE 407FFFFE 00 407FFFFE OK 00 ok
00000000 57FFD7FF DA000084 DA000084 00 DA000084 OK 00 ok
3E1003FF 7F800024 5F7FEFFF 7FC00024 10 7FC00000 NG 10 ok
7FFFCOFE C59FF7FF 3FFFFFFE 7FFFCOFE 00 7FC00000 NG 10 ng
...
#NU
00000000 80000004 407FFFFE 407FFFFE 00 407FFFFE OK 00 ok
00000000 57FFD7FF DA000084 DA000084 00 DA000084 OK 00 ok
3E1003FF 7F800024 5F7FEFFF 7FC00024 10 7FC00000 NG 10 ok
7FFFCOFE C59FF7FF 3FFFFFFE 7FFFCOFE 00 7FC00000 NG 10 ng
...
#NM
00000000 80000004 407FFFFE 407FFFFE 00 407FFFFE OK 00 ok
00000000 57FFD7FF DA000084 DA000084 00 DA000084 OK 00 ok
3E1003FF 7F800024 5F7FEFFF 7FC00024 10 7FC00000 NG 10 ok
7FFFCOFE C59FF7FF 3FFFFFFE 7FFFCOFE 00 7FC00000 NG 10 ng
...
```

Listing 3.8: Example Result of 3-Operand Test Case (testfloat_N)

Chapter 4

mmRISC-1 FPGA Implementation

This chapter describes implementation of mmRISC-1 in FPGA.

4.1 FPGA Implementation

4.1.1 FPGA Configurations

The target FPGA is the Intel MAX 10 10M50DAF484C7G and the target board is the Terasic DE10-Lite Board, as shown in Figure 4.1.

There are two configurations, as shown in Table 4.1: (1) RV32IMAC with selectable 4-wire JTAG or 2-wire cJTAG, (2) RV32IMAFC with selectable 4-wire JTAG or 2-wire cJTAG. You can select each configuration by disabling or enabling the `define switches: RISCV_ISA_RV32F in defines_core.v, which is stored in the ./verilog/common directory.

If the RISC-V ISA in mmRISC-1 does not have floating point instructions (RV32IMAC: `define RISCV_ISA_RV32F is disabled), the FPGA operates at 20MHz (Tcyc = 50ns). However, if it has floating point instructions (RV32IMAFC: `define RISCV_ISA_RV32F is enabled), the FPGA operates at 16.67MHz (Tcyc = 60ns) due to the complexity of logic for floating point operations. In the CHIP_TOP layer, if “`RISCV_ISA_RV32F” is defined, the 16.67MHz output from the PLL is used as the system clock. Otherwise, the 20MHz output from the PLL is used. The examples of FPGA configuration results are shown in Table 4.1.

The FPGA-related resources are stored in the directory "mmRISC-1/fpga". After launching Quartus Prime (Figure 4.2), please open the project file "mmRISC.qpf". The timing constraints file is "mmRISC.sdc".

Please compile the project. Then, you will find a non-volatile MAX 10 FLASH configuration file "mmRISC.pof" and a volatile MAX 10 SRAM configuration file "mmRISC.sof" in the directory "./fpga/output_files". Please download either of them to the FPGA via the USB Blaster Interface using the "Programmer" application of Quartus Prime.

4.1.2 Initialization of Instruction RAM

If you want to make the Instruction RAM have initial data like a ROM, apply the macro keyword "FPGA" in the Quartus synthesis setting. And place the initialization data RAM128KB_DP.mif in the directory "fpga", which can be generated from the Intel Hex format by the hex2mif command prepared in the directory "tools" as shown below. The source code of the tool hex2mif.c is located in the same directory. A “*.hex” is an Intel Hex format binary built from your C program.

```
$ ./tools/hex2mif *.hex > RAM128KB_DP.mif
```

The macro "FPGA" switches ram.v to ram_fpga.v for the Instruction RAM (U_RAMI) in the chip_top layer. The ram_fpga.v refers to the FPGA IP of RAM128KB_DP.v stored in the directory "fpga".

The already prepared RAM128KB_DP.mif has been generated from "mmRISC_TouchLCD" (without floating point instructions) by the above command. If there is no Touch LCD shield on the FPGA board, this demo program toggles LEDs and outputs 3D acceleration sensor data to the UART terminal.

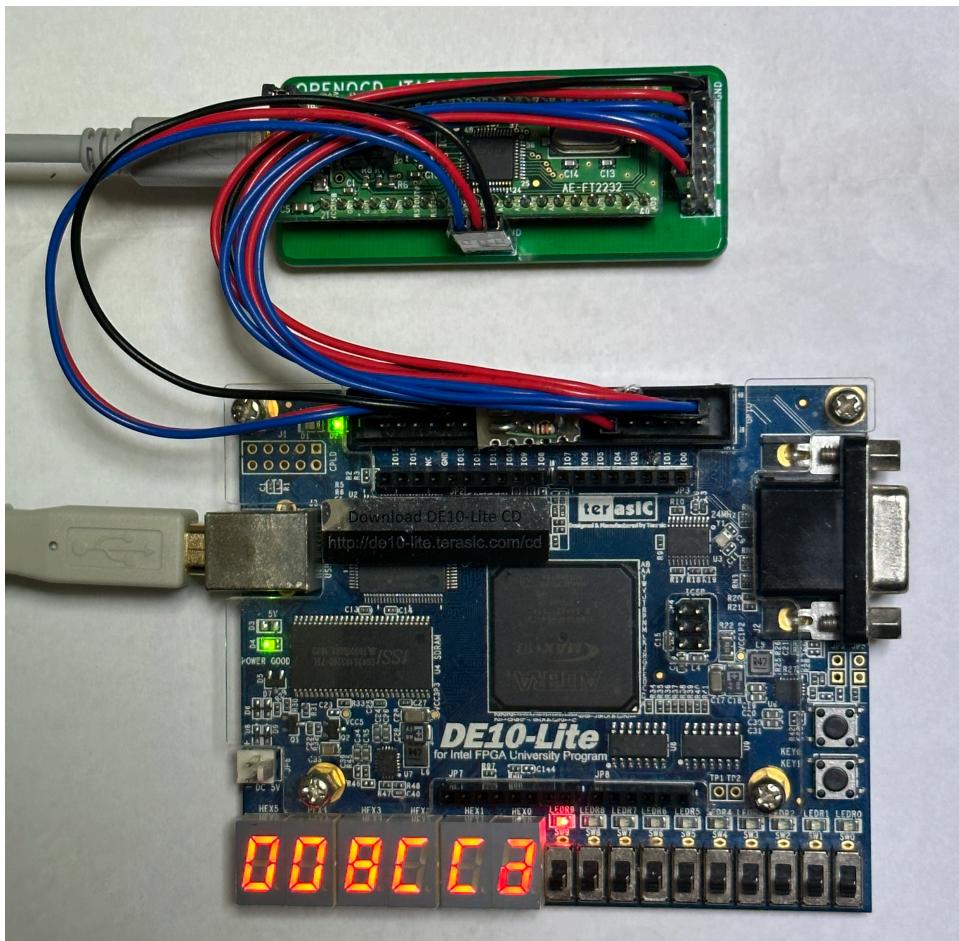


Figure 4.1: Terasic DE10-Lite Board & OpenOCD JTAG Adapter

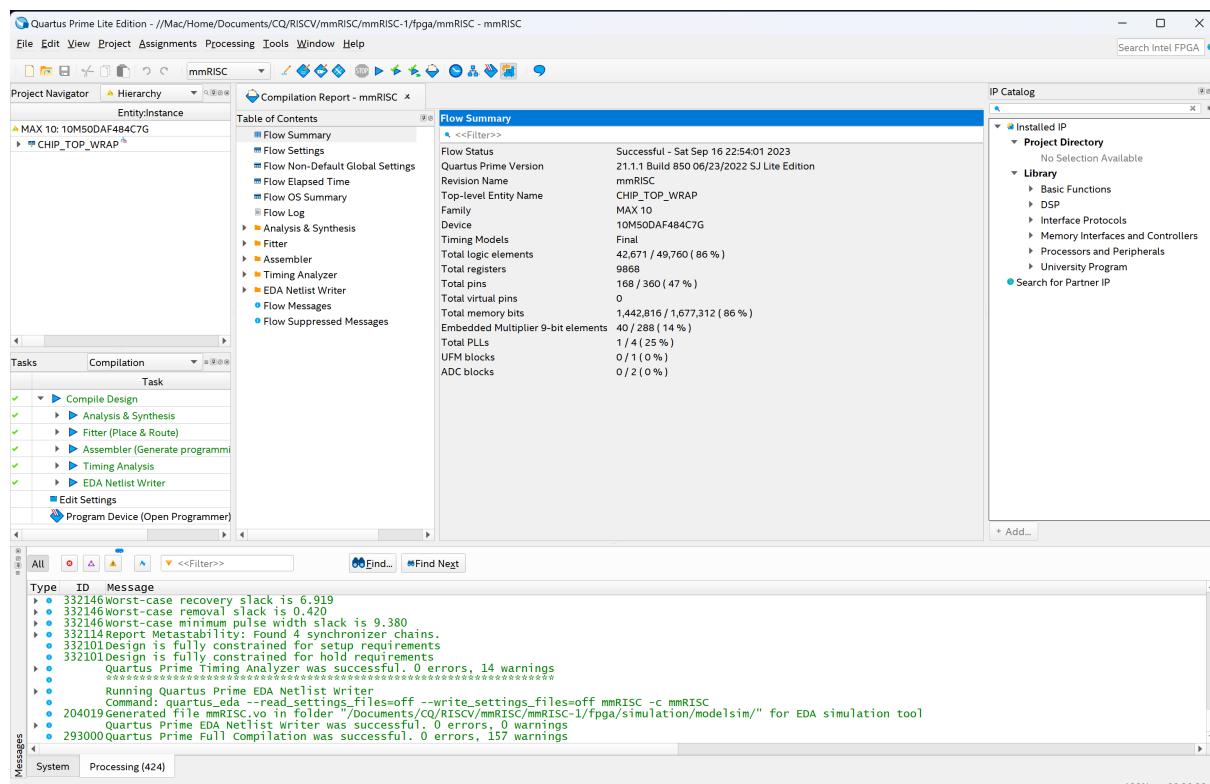


Figure 4.2: Intel Quartus Prime Lite Edition

RISC-V ISA	RV32IMAC (integer)	RV32IMAFC (interger + floating)	Note
Configuration Data	./fpga/output_files/ RV32IMAC_JTAGcJTAG/ mmRISC.pof	./fpga/output_files/ RV32IMAFC_JTAGcJTAG/ mmRISC.pof	
Debug I/F	4-wire JTAG		
Peripherals	Instruction RAM 128KB (initialized by .mif), Data RAM 48KB , Multilayer AHB BUS, UART + Timer + GPIO + I2C(2ch) + SPI + SDRAM I/F		
Device	MAX 10 10M50DAF484C7G		
Quartus Prime	Lite Edition Ver.23.1std.0		
Optimization	Performance (High effort)		
Logic Elements	25266/49760 = 49.8%	43190/49760 = 86.8%	
Interconnect Usage	81,6% (peak)	91.8% (peak)	
Frequency	20MHz	16.67MHz	
Worst Setup Slack	0.964ns (met)	4.662ns (met)	

Table 4.1: FPGA Configuration Results of mmRISC-1

4.1.3 TIPS: How to configure the FPGA on ARM based macOS environment

Even if your environment is an ARM-based macOS machine, such as a MacBook Pro with Apple Silicon M-series, you can install and use Intel Questa Prime on the Windows OS for 64-bit ARM on the Parallels Desktop. However, you cannot configure the FPGA via the USB Blaster cable from the ARM Windows. In such a case, please configure the FPGA from macOS by using UrJTAG and the SVF (Serial Vector Format) file.

STEP 1: Convert configuration file to SVF

To convert the configuration file to SVF, follow these steps:

- (1) In the "Programmer" window, add the "./fpga/output_files/mmRISC.sof" for the MAX 10 SRAM configuration (volatile) or the "./fpga/output_files/mmRISC.pof" for the MAX 10 FLASH configuration (non-volatile).
- (2) For the "mmRISC.pof", do not forget to check the box of "Program/Configure" in the "CFM0" line.
- (3) Select the menu "File" -> "Create JAM, JBC, SVF, or ISC File...", and select "Serial Vector Format (.svf)" from the "File format" drop-down list.
- (4) Enter the file name "mmRISC.sof.svf" to be converted from "mmRISC.sof", or the file name of "mmRISC.pof.svf" to be converted from "mmRISC.pof" in the "File name" field.
- (5) The conversion will start when you click the OK button.

STEP 2: Install UrJTAG on your macOS

To install UrJTAG on your macOS, follow these steps:

- (1) Download the UrJTAG source files from <http://urjtag.org> and extract the tarball.
- (2) % ./configure
- (3) % make (If you encounter a link error during the "make" process, please search for the countermeasures from the internet because they might depend on the UrJTAG

versions.)

(4) % sudo make install

(5) Confirm whether the "jtag" command works on the terminal window.

(6) Find and download the BSDL file "MAX_10_10M50DAF484.bsd1" from the internet.

(7) After launching the "jtag" command, convert the BSDL file to a JTAG information data by using the subcommand shown below in the "jtag" command session.

```
jtag> bsd1 dump MAX_10_10M50DAF484.bsd1
```

And save the displayed data as a text file named "10M50DAF484" in /usr/local/share/urjtag/altera/10m50daf484.

(8) Quit from the "jtag" command session by using the "quit" command.

(9) You can find a text file "PARTS" in /usr/local/share/urjtag/altera. Append the following line at the bottom of the "PARTS" file.

```
0011000100000101 10m50daf484 10M50DAF484
```

(10) Create a text file "STEPPINGS" in /usr/local/share/urjtag/altera/10m50daf484 with the following content:

```
0000 10m50daf484 0
```

(11) Just for your reference, (9) and (10) are based on the following information described in the BSDL file.

```
attribute IDCODE_REGISTER of MAX_10_10M50DAF484 : entity is
  "0000" &           --4-bit Version
  "0011000100000101" &  --16-bit Part Number
  "00001101110" &    --11-bit Manufacturer's Identity
  "1";               --Mandatory LSB
```

STEP 3: Configure the FPGA by UrJTAG

To configure the FPGA, follow these steps:

(1) Connect your FPGA board and Mac via the USB Blaster cable.

(2) Launch the "jtag" command in your terminal.

(3) In the "jtag" session, enter the "cable" command as follows.

```
jtag> cable UsbBlaster
```

(4) Enter the "detect" command as follows. jtag> detect

If successful, you will see the following messages.

```
IR length: 10
Chain length: 1
Device Id: 00000011000100000101000011011101 (0x031050DD)
  Manufacturer: Altera (0x0DD)
  Part(0):      10M50DAF484 (0x3105)
  Stepping:     0
  Filename:     /usr/local/share/urjtag/altera/10m50daf484/10m50daf484}
```

(5) Program the FPGA by using the SVF. For the MAX 10 SRAM configuration (volatile):

```
jtag> svf mmRISC.sof.svf stop progress
```

For the MAX 10 FLASH configuration (non-volatile):

```
jtag> svf mmRISC.pof.svf stop progress
```

(6) Quit from the "jtag" command.

4.1.4 FPGA Pin Assignments

The FPGA pin assignments are shown in Table 4.2 and Table 4.3. The Table 4.4 shows some pins for special functions of the mmRISC-1 tiny SoC, such as cJTAG, stand-by (low power mode), authentication (security), slow clock, and halt-on-reset, which are described

in Section 2.16. The positions on the FPGA board to use the special features are shown in Figure 4.3.

Group	Signal Name	In/Out	FPGA	DE10 Lite Board
Reset Push Switch	RES_N	In	B8	KEY0
Reset Output	RES_OUT_N	Out	F16	ARDUINO RESET_N
Clock	CLK50	In	P11	50MHz Clock
JTAG	TRSTn	In	Y5	Pin Header GPIO_29
JTAG	TCK	In	Y6	Pin Header GPIO_27
JTAG	TMS	In	AA2	Pin Header GPIO_35
JTAG	TDI	In	Y4	Pin Header GPIO_31
JTAG	TDO	Out/Z	Y3	Pin Header GPIO_33
CJTAG	CKKC_pri	OUT	W13	Pin Header GPIO_13
CJTAG	CKKC_rep	IN	AB13	Pin Header GPIO_15
CJTAG	TMSC_pri	OUT	AA14	Pin Header GPIO_12
CJTAG	TMSC_rep	IN	W12	Pin Header GPIO_14
CJTAG	TSMC_PUP_rep	OUT	Y11	Pin Header GPIO_17
CJTAG	TSMC_PDN_rep	OUT	AB12	Pin Header GPIO_16
STBY	STBY_ACK_N	OUT	L19	HEX57 segment DP
7-Segment LED0	GPIO0[0]	In/Out	C14	HEX00 segment A
7-Segment LED0	GPIO0[1]	In/Out	E15	HEX01 segment B
7-Segment LED0	GPIO0[2]	In/Out	C15	HEX02 segment C
7-Segment LED0	GPIO0[3]	In/Out	C16	HEX03 segment D
7-Segment LED0	GPIO0[4]	In/Out	E16	HEX04 segment E
7-Segment LED0	GPIO0[5]	In/Out	D17	HEX05 segment F
7-Segment LED0	GPIO0[6]	In/Out	C17	HEX06 segment G
7-Segment LED0	GPIO0[7]	In/Out	D15	HEX07 segment DP
7-Segment LED1	GPIO0[8]	In/Out	C18	HEX10 segment A
7-Segment LED1	GPIO0[9]	In/Out	D18	HEX11 segment B
7-Segment LED1	GPIO0[10]	In/Out	E18	HEX12 segment C
7-Segment LED1	GPIO0[11]	In/Out	B16	HEX13 segment D
7-Segment LED1	GPIO0[12]	In/Out	A17	HEX14 segment E
7-Segment LED1	GPIO0[13]	In/Out	A18	HEX15 segment F
7-Segment LED1	GPIO0[14]	In/Out	B17	HEX16 segment G
7-Segment LED1	GPIO0[15]	In/Out	A16	HEX17 segment DP
7-Segment LED2	GPIO0[16]	In/Out	B20	HEX20 segment A
7-Segment LED2	GPIO0[17]	In/Out	A20	HEX21 segment B
7-Segment LED2	GPIO0[18]	In/Out	B19	HEX22 segment C
7-Segment LED2	GPIO0[19]	In/Out	A21	HEX23 segment D
7-Segment LED2	GPIO0[20]	In/Out	B21	HEX24 segment E
7-Segment LED2	GPIO0[21]	In/Out	C22	HEX25 segment F
7-Segment LED2	GPIO0[22]	In/Out	B22	HEX26 segment G
7-Segment LED2	GPIO0[23]	In/Out	A19	HEX27 segment DP
7-Segment LED3	GPIO0[24]	In/Out	F21	HEX30 segment A
7-Segment LED3	GPIO0[25]	In/Out	E22	HEX31 segment B
Group	Signal Name	In/Out	FPGA	DE10 Lite Board
7-Segment LED3	GPIO0[26]	In/Out	E21	HEX32 segment C
7-Segment LED3	GPIO0[27]	In/Out	C19	HEX33 segment D
7-Segment LED3	GPIO0[28]	In/Out	C20	HEX34 segment E
7-Segment LED3	GPIO0[29]	In/Out	D19	HEX35 segment F
7-Segment LED3	GPIO0[30]	In/Out	E17	HEX36 segment G
7-Segment LED3	GPIO0[31]	In/Out	D22	HEX37 segment DP
7-Segment LED4	GPIO1[0]	In/Out	F18	HEX40 segment A
7-Segment LED4	GPIO1[1]	In/Out	E20	HEX41 segment B
7-Segment LED4	GPIO1[2]	In/Out	E19	HEX42 segment C
7-Segment LED4	GPIO1[3]	In/Out	J18	HEX43 segment D
7-Segment LED4	GPIO1[4]	In/Out	H19	HEX44 segment E
7-Segment LED4	GPIO1[5]	In/Out	F19	HEX45 segment F
7-Segment LED4	GPIO1[6]	In/Out	F20	HEX46 segment G
7-Segment LED4	GPIO1[7]	In/Out	F17	HEX47 segment DP
7-Segment LED5	GPIO1[8]	In/Out	J20	HEX50 segment A
7-Segment LED5	GPIO1[9]	In/Out	K20	HEX51 segment B
7-Segment LED5	GPIO1[10]	In/Out	L18	HEX52 segment C
7-Segment LED5	GPIO1[11]	In/Out	N18	HEX53 segment D
7-Segment LED5	GPIO1[12]	In/Out	M20	HEX54 segment E
7-Segment LED5	GPIO1[13]	In/Out	N19	HEX55 segment F
7-Segment LED5	GPIO1[14]	In/Out	N20	HEX56 segment G
VGA	GPIO1[15]	In/Out	Y2	VGA_R2
LED	GPIO1[16]	In/Out	A8	LEDR0
LED	GPIO1[17]	In/Out	A9	LEDR1
LED	GPIO1[18]	In/Out	A10	LEDR2
LED	GPIO1[19]	In/Out	B10	LEDR3
LED	GPIO1[20]	In/Out	D13	LEDR4
LED	GPIO1[21]	In/Out	C13	LEDR5
LED	GPIO1[22]	In/Out	E14	LEDR6
LED	GPIO1[23]	In/Out	D14	LEDR7
LED	GPIO1[24]	In/Out	A11	LEDR8
LED	GPIO1[25]	In/Out	B11	LEDR9
General Purpose	GPIO1[26]	In/Out	V10	Pin Header GPIO_0
General Purpose	GPIO1[27]	In/Out	V9	Pin Header GPIO_2
General Purpose	GPIO1[28]	In/Out	V8	Pin Header GPIO_4
General Purpose	GPIO1[29]	In/Out	W8	Pin Header GPIO_5
General Purpose	GPIO1[30]	In/Out	V7	Pin Header GPIO_6
General Purpose	GPIO1[31]	In/Out	W7	Pin Header GPIO_7
Slide Switch	GPIO2[0]	In/Out	C10	SW0
Slide Switch	GPIO2[1]	In/Out	C11	SW1
Slide Switch	GPIO2[2]	In/Out	D12	SW2
Slide Switch	GPIO2[3]	In/Out	C12	SW3

Table 4.2: FPGA Pin Assignments (1)

Group	Signal Name	In/Out	FPGA	DE10 Lite Board
Slide Switch	GPIO2[4]	In/Out	A12	SW4
Slide Switch	GPIO2[5]	In/Out	B12	SW5
Slide Switch	GPIO2[6]	In/Out	A13	SW6 (ENABLE CJTAG)
Slide Switch	GPIO2[7]	In/Out	A14	SW7 (SLOW CLOCK)
Slide Switch	GPIO2[8]	In/Out	B14	SW8 (STBY REQ)
Slide Switch	GPIO2[9]	In/Out	F15	SW9 (DEBUG SECURE)
Push Switch	GPIO2[10]	In/Out	A7	KEY1 (RESET HALT)
General Purpose	GPIO2[11]	In/Out	W6	Pin Header GPIO_8
General Purpose	GPIO2[12]	In/Out	V5	Pin Header GPIO_9
General Purpose	GPIO2[13]	In/Out	W5	Pin Header GPIO_10
General Purpose	GPIO2[14]	In/Out	AA15	Pin Header GPIO_11
General Purpose	GPIO2[15]	In/Out	AB11	Pin Header GPIO_18
General Purpose	GPIO2[16]	In/Out	W11	Pin Header GPIO_19
General Purpose	GPIO2[17]	In/Out	AB10	Pin Header GPIO_20
General Purpose	GPIO2[18]	In/Out	AA10	Pin Header GPIO_21
General Purpose	GPIO2[19]	In/Out	AA9	Pin Header GPIO_22
General Purpose	GPIO2[20]	In/Out	Y8	Pin Header GPIO_23
General Purpose	GPIO2[21]	In/Out	AA8	Pin Header GPIO_24
General Purpose	GPIO2[22]	In/Out	Y7	Pin Header GPIO_25
General Purpose	GPIO2[23]	In/Out	AA7	Pin Header GPIO_26
General Purpose	GPIO2[24]	In/Out	AA6	Pin Header GPIO_28
General Purpose	GPIO2[25]	In/Out	AA5	Pin Header GPIO_30
General Purpose	GPIO2[26]	In/Out	AB3	Pin Header GPIO_32
General Purpose	GPIO2[27]	In/Out	AB2	Pin Header GPIO_34
VGA	GPIO2[28]	In/Out	N1	VGA VS
VGA	GPIO2[29]	In/Out	N3	VGA HS
VGA	GPIO2[30]	In/Out	AA1	VGA R0
VGA	GPIO2[31]	In/Out	V1	VGA R1
SDRAM	SDRAM_CLK	Out	L14	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_CKE	Out	N22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_CSn	Out	U20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQM[0]	Out	V22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQM[1]	Out	J21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_RASn	Out	U22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_CASn	Out	U21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_WEn	Out	V20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_BA[0]	Out	T21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_BA[1]	Out	T22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[0]	Out	U17	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[1]	Out	W19	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[2]	Out	V18	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[3]	Out	U18	64MB (32M x 16bit) SDRAM
Group	Signal Name	In/Out	FPGA	DE10 Lite Board
SDRAM	SDRAM_ADDR[4]	Out	U19	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[5]	Out	T18	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[6]	Out	T19	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[7]	Out	R18	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[8]	Out	P18	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[9]	Out	P19	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[10]	Out	T20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[11]	Out	P20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_ADDR[12]	Out	R20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[0]	In/Out	Y21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[1]	In/Out	Y20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[2]	In/Out	AA22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[3]	In/Out	AA21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[4]	In/Out	Y22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[5]	In/Out	W22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[6]	In/Out	W20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[7]	In/Out	V21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[8]	In/Out	P21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[9]	In/Out	J22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[10]	In/Out	H21	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[11]	In/Out	H22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[12]	In/Out	G22	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[13]	In/Out	G20	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[14]	In/Out	G19	64MB (32M x 16bit) SDRAM
SDRAM	SDRAM_DQ[15]	In/Out	F22	64MB (32M x 16bit) SDRAM
UART	TXD	Out	W10	Pin Header GPIO_1
UART	RXD	In	W9	Pin Header GPIO_3
I2C0	I2C0_SCL	In/Out	AB15	GSENSOR ADXL345
I2C0	I2C0_SDA	In/Out	V11	GSENSOR ADXL345
I2C0	I2C0_ENA	Out	AB16	GSENSOR ADXL345
I2C0	I2C0_ADR	Out	V12	GSENSOR ADXL345
I2C0	I2C0_INT1	In	Y14	GSENSOR ADXL345
I2C0	I2C0_INT2	In	Y13	GSENSOR ADXL345
I2C1	I2C1_SCL	In/Out	AA20	Arduino IO15 (SCL)
I2C1	I2C1_SDA	In/Out	AB21	Arduino IO14 (SDA)
SPI	SPI_CSN[0]	Out	AB19	Arduino IO10 (D10/SS)
SPI	SPI_CSN[1]	Out	AA17	Arduino IO09 (D9)
SPI	SPI_CSN[2]	Out	AB17	Arduino IO08 (D8)
SPI	SPI_CSN[3]	Out	AB9	Arduino IO04 (D4)
SPI	SPI_MOSI	Out	AA19	Arduino IO11 (D11/MISO)
SPI	SPI_MISO	In	Y19	Arduino IO12 (D12/MISO)
SPI	SPI_SCK	In	AB20	Arduino IO13 (D13/SCK)

Table 4.3: FPGA Pin Assignments (2)

Group	Signal Name	In/Out	FPGA	DE10 Lite Board	Description
cJTAG	TCKC_pri	OUT	W13	Pin Header GPIO_13	Connect directly each other
cJTAG	TCKC_rep	IN	AB13	Pin Header GPIO_15	
cJTAG	TMSC_pri	OUT	AA14	Pin Header GPIO_12	Connect directly each other
cJTAG	TMSC_rep	IN	W12	Pin Header GPIO_14	
cJTAG	TSMC_PUP_rep	OUT	Y11	Pin Header GPIO_17	Connect to TMSC_pri through 10k resistor
cJTAG	TSMC_PDN_rep	OUT	AB12	Pin Header GPIO_16	Connect to TMSC_pri through 10k resistor
Slide Switch	GPIO2[6]	In/Out	A13	SW6 (ENABLE cJTAG)	To use cJTAG, slide up SW6; to use JTAG, slide down SW6
Slide Switch	GPIO2[7]	In/Out	A14	SW7 (SLOW CLOCK)	To make the System Clock slow (100KHz), slide up SW7
Slide Switch	GPIO2[8]	In/Out	B14	SW8 (STBY REQ)	To enter STBY mode, slide up SW8
Slide Switch	GPIO2[9]	In/Out	F15	SW9 (DEBUG SECURE)	To enable Secure Authentication, slide up SW9
Push Switch	GPIO2[10]	In/Out	A7	KEY1 (RESET HALT)	At negation of RESET, push KEY1 to enter Halt-on-Reset
STBY	STBY_ACK_N	OUT	L19	HEX57 segment DP	In STBY mode, segment DP of HEX57 becomes ON
VGA	GPIO1[15]	In/Out	Y2	VGA_R2	
VGA	GPIO2[28]	In/Out	N1	VGA_VS	
VGA	GPIO2[29]	In/Out	N3	VGA_HS	
VGA	GPIO2[30]	In/Out	AA1	VGA_R0	
VGA	GPIO2[31]	In/Out	V1	VGA_R1	

Table 4.4: FPGA Special Function Pins

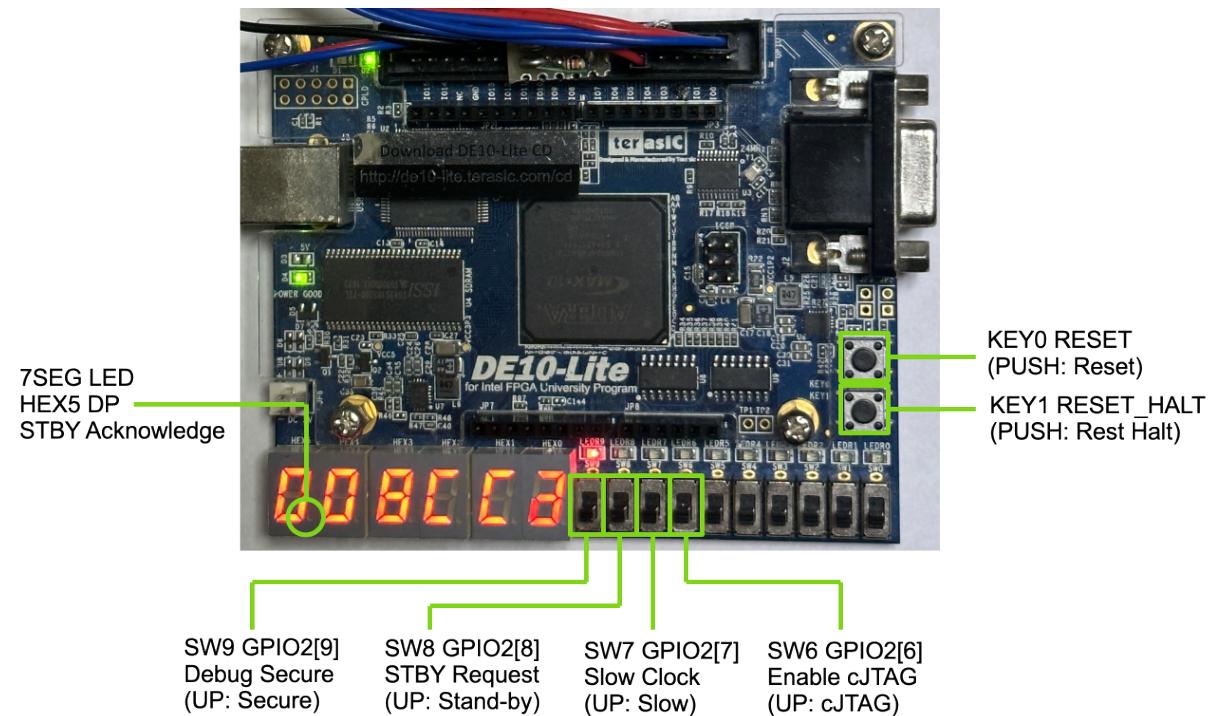


Figure 4.3: Operation Positions to use special features

4.2 JTAG/cJTAG Debugger Interface for OpenOCD

4.2.1 USB to 4-wire JTAG Interface for OpenOCD

You can prepare the USB to 4-wire JTAG interface for OpenOCD by using the FTDI FT2232D chip. To do so, the commercial breakout board of FT2232D is convenient. You can get the board from the following URL.

<https://akizukidenshi.com/catalog/g/gM-02990/>

Figure 4.4 shows the outline of the interface board. The left one is a convenient version made with a breadboard and the breakout board of FT2232D. The right one is made with a printed-circuit board and the breakout board.

NOTE: If you use the breakout board of FT2232D shown above, please set the jumper pins as follows: JP1=short, JP2A=open, and JP2B=open, in order to supply 3.3V to VCCIOA and VCCIOB of the FT2232D chip from the target FPGA board.

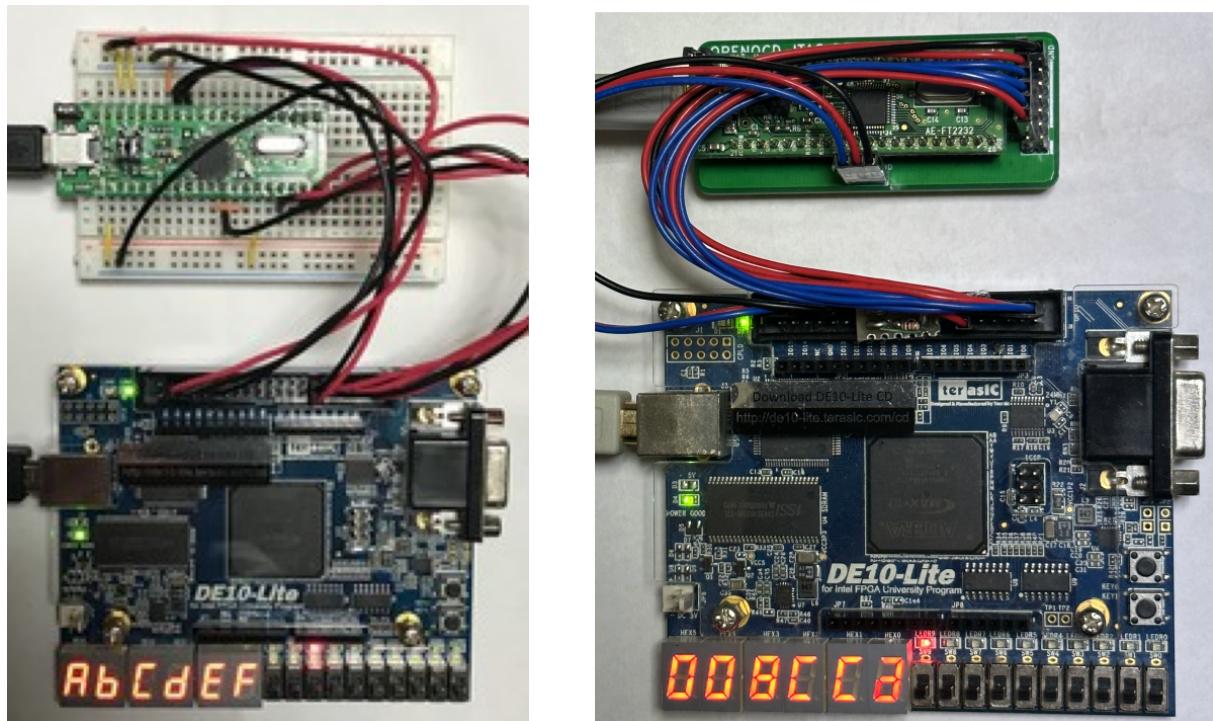


Figure 4.4: Outline of OpenOCD JTAG/cJTAG Adapter (Left:Breadboard Type, Right:Print Board Type)

You can make the interface according to the basic schematic that is shown in Figure 4.5. It is recommended to insert resistors of 200 ohms or less between the FT2232D and the FPGA to avoid hardware damage due to signal contention, and to improve signal integrity by reducing cross talk and signal ringing.

The detailed schematic including the FT2232D breakout is shown in Figure 4.6.

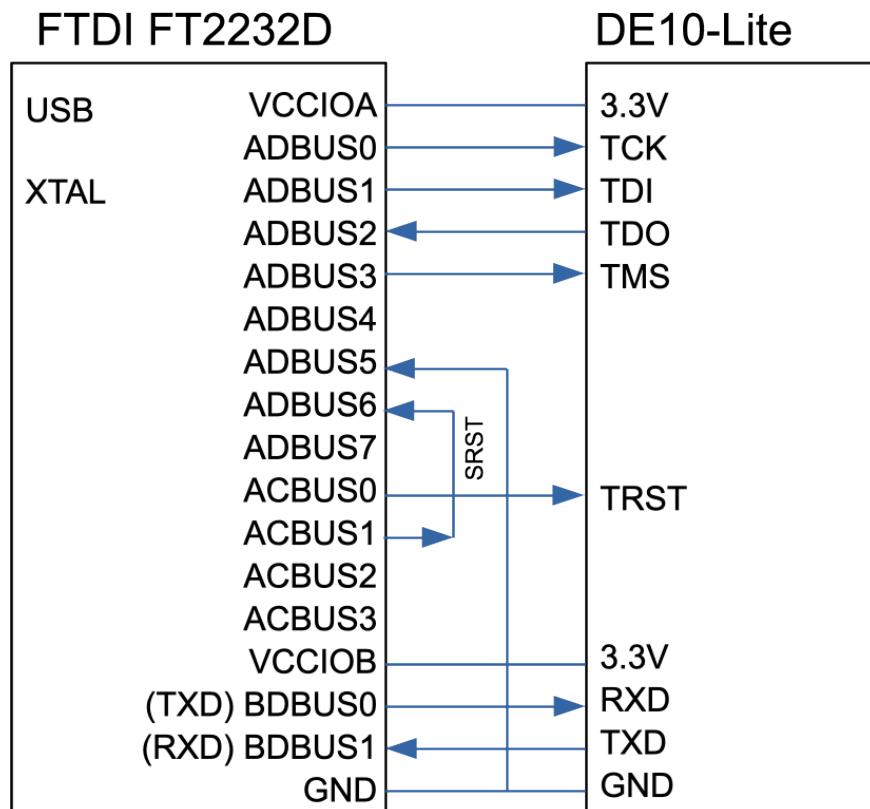


Figure 4.5: Fundamental Connection of OpenOCD JTAG Adapter

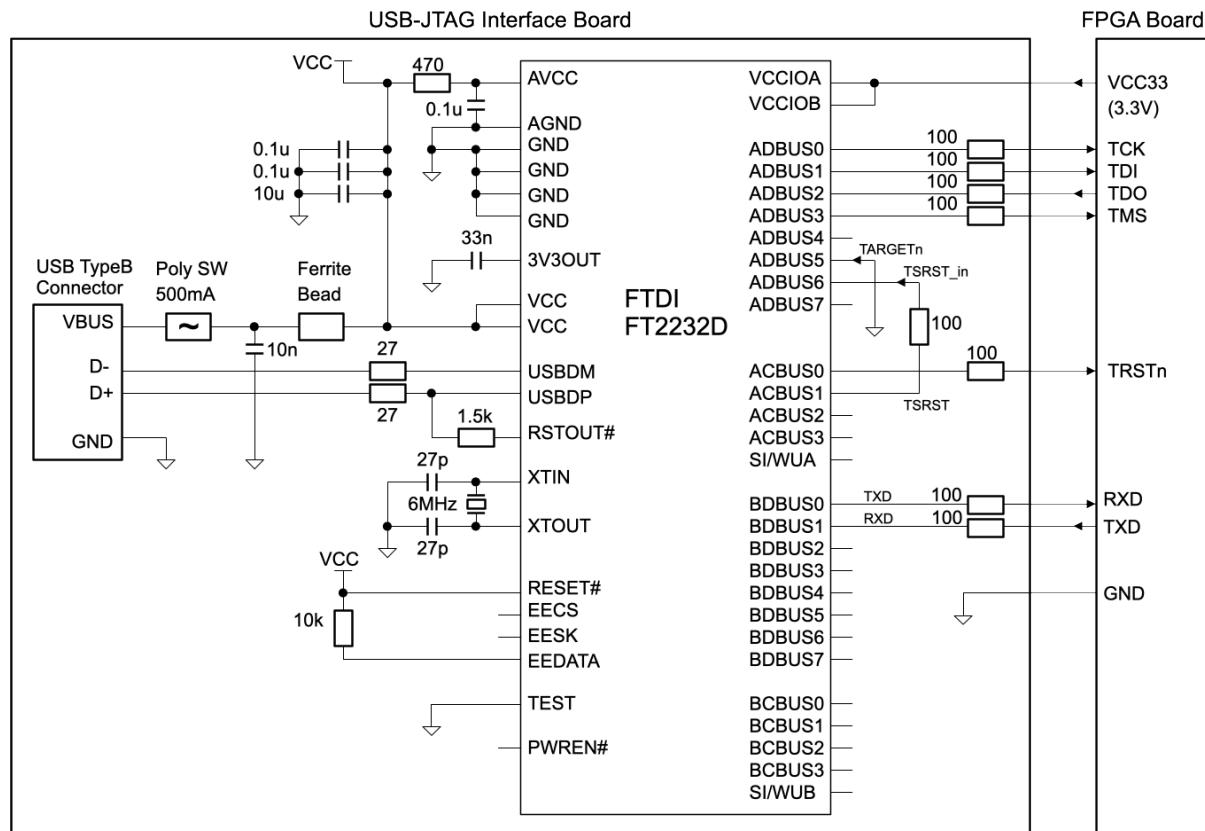


Figure 4.6: Detail Schematics of OpenOCD JTAG Adapter

Of course, you can use the commercial Olimex ARM-USB-OCD(H) as the JTAG

debugger interface for OpenOCD. In the case, it is required to modify the configuration file of OpenOCD. The details are described later.

4.2.2 Connection to the target FPGA as 4-wire JTAG interface

If you set the debug interface of the target FPGA to 4-wire JTAG by sliding down SW6 on the DE10-Lite board , please refer to Figure 4.7 when connecting the JTAG Adapter between the FPGA board and the PC running OpenOCD. For this system, the configuration file of OpenOCD is shown in Listing 4.1.

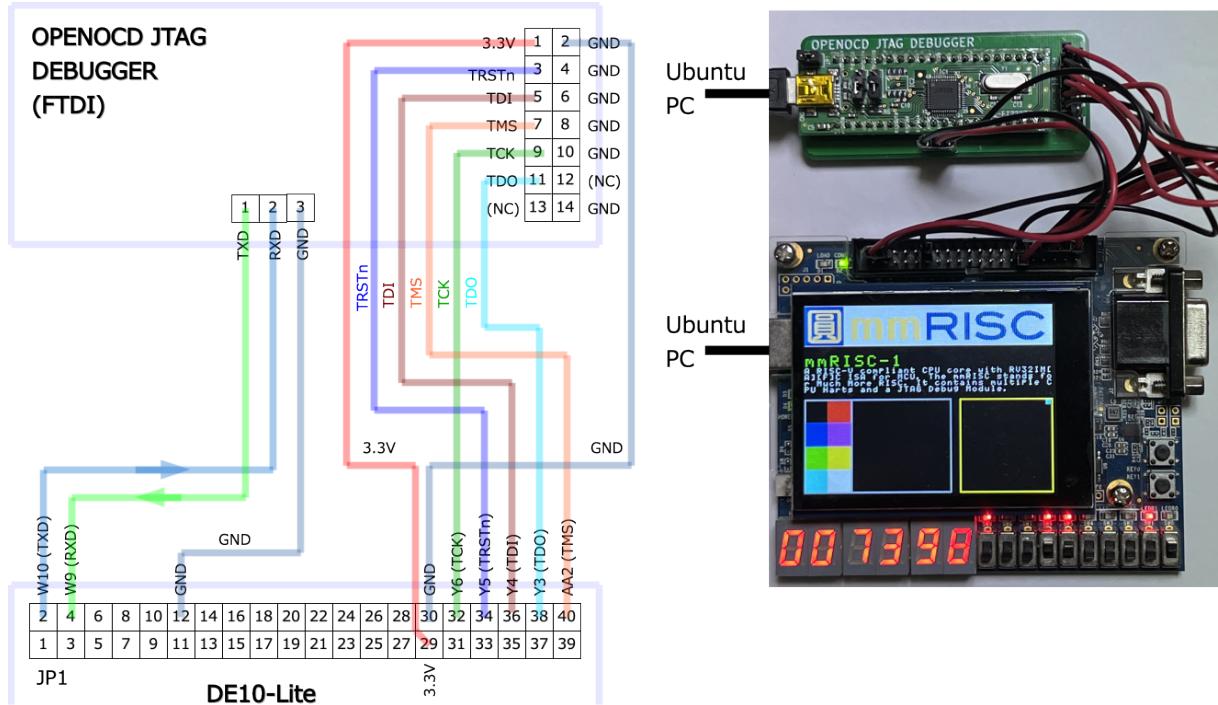


Figure 4.7: Connection of OpenOCD JTAG Adapter and PC

```

adapter driver ftdi
adapter speed 1000
ftdi_device_desc "Dual RS232"
ftdi_vid_pid 0x0403 0x6010

ftdi_layout_init 0x0908 0x0b1b
ftdi_layout_signal nSRST -oe 0x0200
ftdi_layout_signal nTRST -data 0x0100
ftdi_layout_signal LED -data 0x0800

reset_config trst_and_srst

set _chipname riscv
jtag newtap $_chipname cpu -irlen 5

set _targetname $_chipname.cpu
target create $_targetname riscv -endian little -chain-position $_targetname -coreid 0

init

riscv authdata_write 0xbeefcafe

```

Listing 4.1: OpenOCD Configuration File for 4-wire JTAG interface

NOTE: When you use the commercial Olimex ARM-USB-OCD(H), an example of a corresponding OpenOCD configuration file is shown in Listing 4.2.

```

adapter driver ftdi
adapter speed 1000
ftdi_device_desc "Olimex OpenOCD JTAG ARM-USB-OCD-H"
ftdi_vid_pid 0x15ba 0x002b

ftdi_layout_init 0x0908 0x0b1b
ftdi_layout_signal nSRST -oe 0x0200
ftdi_layout_signal nTRST -data 0x0100
ftdi_layout_signal LED -data 0x0800

reset_config trst_and_srst

set _chipname riscv
jtag newtap $_chipname cpu -irlen 5

set _targetname $_chipname.cpu
target create $_targetname riscv -endian little -chain-position $_targetname -coreid 0

init

riscv authdata_write 0xbeefcafe

```

Listing 4.2: OpenOCD Configuration File for Olimex ARM-USB-OCD(H)

4.2.3 Connection to the target FPGA as 2-wire cJTAG interface

If you set the debug interface of the target FPGA to 2-wire cJTAG by sliding up SW6 on the DE10-Lite board, you can connect a generic debug probe with a cJTAG interface to the mmRISC-1. Please connect TCKC of the probe to TCKC_rep of mmRISC-1 SoC, and TMSC of the probe to TMSC_rep of mmRISC-1 SoC, as shown in Figure 4.8.

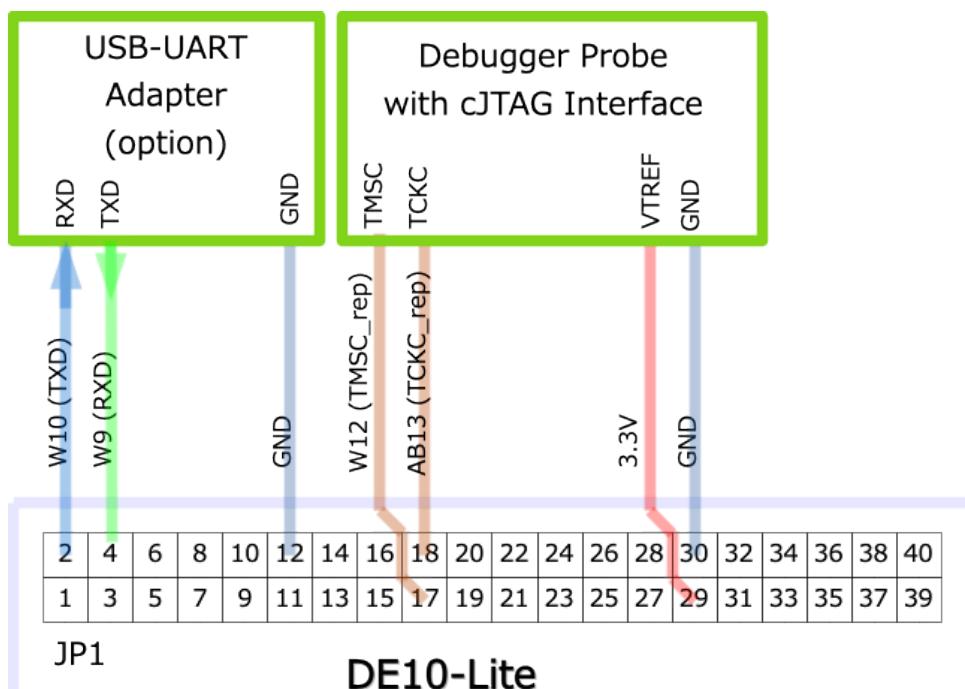


Figure 4.8: Connection between the cJTAG debugger probe and cJTAG signals

In this project, I have not made a dedicated cJTAG adapter. Therefore, both the JTAG to cJTAG converter and the cJTAG to JTAG converter are implemented into one FPGA, and the cJTAG signals are connected at outside of the FPGA; that are the pairs

of TCKC_pri - TCKC_rep and TMSC_pri - TMSC_rep.

First of all, please connect the 4-wire JTAG Adapter as described in previous Section 4.2.2.

Next, please make and install a small circuit board as shown in Figure 4.9. The small board includes the supplemental circuits shown below as described in Section 1.31.6.

1. External connection between TCKC_pri (from JTAG/cJTAG converter in the FPGA) and TCKC_rep (to cJTAG/JTAG converter in the FPGA).
2. External connection between TMSC_pri (from/to JTAG/cJTAG converter in the FPGA) and TMSC_rep (to/from cJTAG/JTAG converter in the FPGA).
3. External pull-up and pull-down resistors on the TMSC pin to control pull-up or bus-keeper.

For this system, the configuration file of OpenOCD is shown in Listing 4.3.

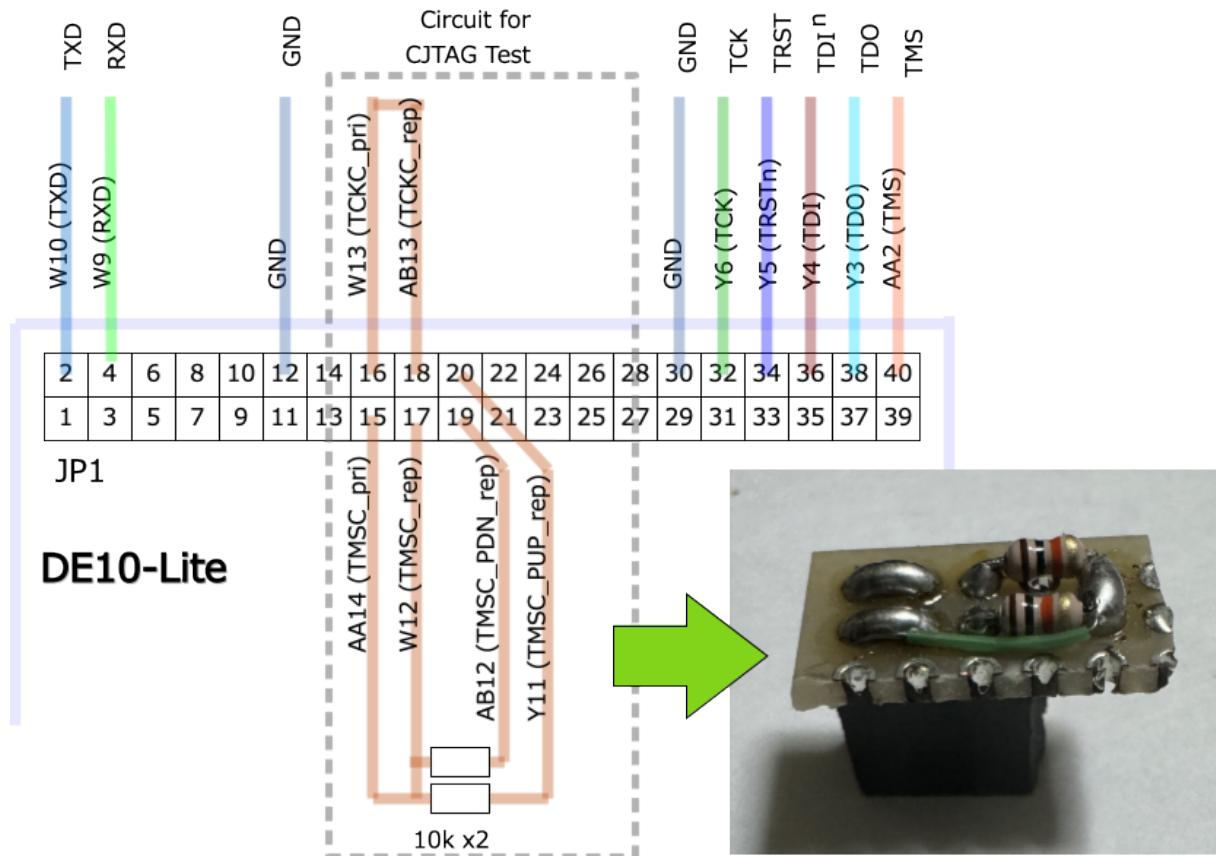


Figure 4.9: Supplement for cJTAG

```

adapter driver ftdi
adapter speed 1000
ftdi oscan1_mode on
set protocol cjttag
ftdi device_desc "Dual RS232"
ftdi vid_pid 0x0403 0x6010

ftdi layout_init 0x0808 0xb1b
ftdi layout_signal nTRST -data 0x0100
ftdi layout_signal nSRST -oe 0x0200

```

```

ftdi layout_signal JTAG_SEL -data 0x0800

ftdi layout_signal TCK -data 0x0001
ftdi layout_signal TDI -data 0x0002
ftdi layout_signal TDO -input 0x0004
ftdi layout_signal TMS -data 0x0008

reset_config srst_only

set _CHIPNAME riscv
jtag newtap $_CHIPNAME cpu -irlen 5

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME riscv -endian little -chain-position $_TARGETNAME -coreid 0

init

riscv authdata_write Oxbeefcafe

```

Listing 4.3: OpenOCD Configuration File for 2-wire cJTAG interface

4.2.4 Raspberry Pi 4 Model B as a Development Environment

If you have a Raspberry Pi 4 Model B, it can be a comprehensive development environment for RISC-V including mmRISC-1. Please note that the GPIO pins of the Raspberry Pi can be a direct OpenOCD 4-wire JTAG interface, so you do not need to prepare a dedicated USB to JTAG interface any more. However, the OpenOCD for Raspberry Pi does not support 2-wire compact JTAG (cJTAG) yet, so please configure the FPGA as the mmRISC-1 with the 4-wire JTAG interface.

You need to install RISC-V development tools on the 64bit Raspberry Pi OS, because the Eclipse requires a 64bit OS. To install all the necessary tools, please follow the steps shown below.

- (1) Install "Eclipse IDE for Embedded C/C++ Developers" aarch64 version.
- (2) Build and Install GNU Tool Chains as follows.

```

$ sudo apt-get install autoconf automake autotools-dev curl python3 \
    libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex \
    texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev
$ git clone --recursive https://github.com/riscv-collab/riscv-gnu-toolchain.git
$ ./configure --prefix=/opt/riscv --enable-multilib
$ sudo make

```

- (3) Install OpenOCD tool as follows. Very simple.

```
$ sudo apt install openocd
```

- (4) Download mmRISC-1 resources from Github, and check the OpenOCD Configuration file "mmRISC-1/openocd/openocd_rpi.cfg" listed in Listing 4.4 which configures the Raspberry Pi GPIO as JTAG port as shown in Table 4.5.

- (5) Connect the Raspberry Pi to the FPGA board as shown in Figure 4.10.

```

adapter driver bcm2835gpio
adapter speed 1000
transport select jtag

#Raspberry Pi 3B/3B+
#bcm2835gpio_peripheral_base 0x3F000000

#Raspberry Pi 4B
bcm2835gpio_peripheral_base 0xFE000000

# Transition delay calculation: SPEED_COEFF/khz - SPEED_OFFSET
# These depend on system clock, calibrated for stock 700MHz
# bcm2835gpio speed SPEED_COEFF SPEED_OFFSET
bcm2835gpio_speed_coeffs 236181 60

```

```

# Each of the JTAG lines need a gpio number set: tck tms tdi tdo
# Header pin numbers: 37 29 33 31
bcm2835gpio_jtag_nums 26 5 13 6

# If you define trst or srst, use appropriate reset_config
# Header pin numbers: TRST - 35, SRST - 40
bcm2835gpio_trst_num 19

# reset_config trst_only
bcm2835gpio_srst_num 21

# reset_config srst_only srst_push_pull
# or if you have both connected,
reset_config trst_and_srst srst_push_pull

proc init_targets {} {
    set _CHIPNAME riscv
    jtag newtap $_CHIPNAME cpu -irlen 5

    set _TARGETNAME $_CHIPNAME.cpu
    target create $_TARGETNAME riscv -endian little -chain-position $_TARGETNAME -coreid 0
}

```

Listing 4.4: OpenOCD Configuration file for Raspberry Pi

BCM2835 GPIO	Header Pin No.	Signal	Note
GPIO14	Pin 8	TXD (output)	/dev/serial0
GPIO15	Pin 10	RXD (input)	/dev/serial1
GPIO05	Pin 29	TMS	
GPIO06	Pin 31	TDO	
GPIO13	Pin 33	TDI	
GPIO19	Pin 35	TRSTn	
GPIO26	Pin 37	TCK	
GPIO21	Pin 40	SRSTn	not used
GND	Pin 6, 9, 14, 20, 25, 30, 34, 39	GND	

Table 4.5: Assignments of GPIO of Raspberry Pi 4 Model B as an OpenOCD JTAG Debugger

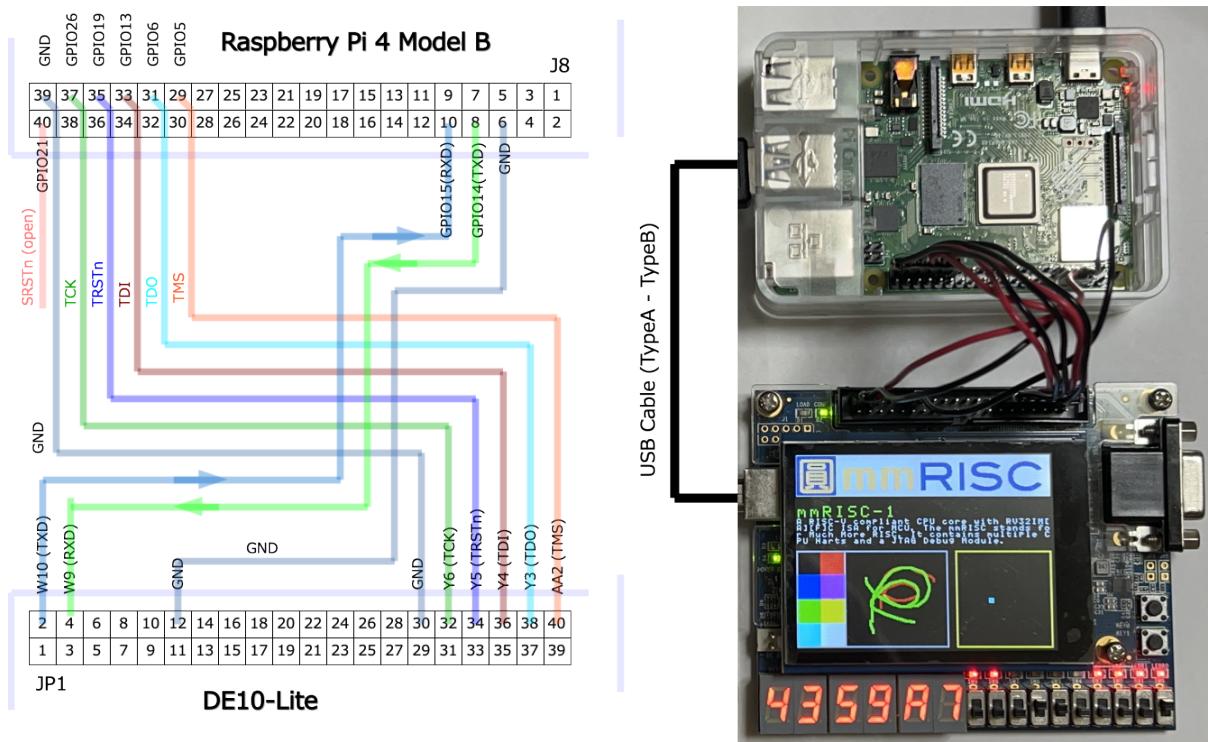


Figure 4.10: Connection of Raspberry Pi and the FPGA

4.2.5 Generic Commercial Development Environments

You can use the generic commercial development environments such as the IAR Embedded Workbench for RISC-V. You can connect the debug probe to the mmRISC-1 SoC.

For the 4-wire JTAG, please connect TRSTn, TCK, TMS, TDI and TDO of the probe to corresponding terminals of mmRISC-1 SoC configured as 4-wire JTAG, respectively.

For the 2-wire cJTAG, please connect TCKC and TMSC of the probe to TCKC_{rep} and TMSC_{rep} of mmRISC-1 SoC configured as 2-wire cJTAG, respectively. In this case, you need not use supplemental circuits shown in Figure 4.9.

4.3 Sample Programs

Sample Programs are provided as the projects of Eclipse in the directory "mmRISC-1/workspace". Please import all the projects in the Eclipse Project Explorer in advance.

4.3.1 Integer Instructions: "mmRISC_SampleCPU"

First, please build the program and launch a terminal application on Ubuntu, such as the minicom. You can also use the serial terminal windows in the debug perspective of the Eclipse IDE, if you have it. The communication parameters are 115200 bps and 8N1 (8bit, Non-Parity and 1 Stop bit).

Before running and debugging the program, change the Eclipse perspective to debug and select the menu of "Run"→"Debug Configuration". Create a debug configuration "mmRISC_SampleCPU" in GDB OpenOCD Debugging. Please fill in "./Debug/mmRISC_SampleCPU.elf" in the C/C++ Application field and click the "Debug" button. The program will break at the beginning of main(). Click the resume button (green right arrow) to restart the program. You can use any debug operations in the Eclipse windows as shown in Figure 4.11.

The MTIME timer requests periodic interrupts to increment the binary number displayed on 10 discrete LEDs. The LED incrementing value is controlled by GPIO2[9:0] and the direction is controlled by GPIO2[10]. Most of GPIO2[9:0] are connected to the slide switches (SW0-SW9) and GPIO2[10] is connected to the push switch (KEY).

Additionally, the sample program reads the 3D Acceleration Sensor ADXL345 on the MAX10-Lite board via the I2C interface. It displays some messages and acceleration data in the terminal by using the printf() function.

When you type any characters in the terminal, the UART receives each ASCII code and requests a RX interrupt. The software handler for the interrupt displays the code on the left 2-digits of the 7-segment LED and increments a number on the right 4-digits 7-segment LED.

The FPGA's operating frequency depends on whether the floating point ISA is implemented or not. Therefore, the program reads the CSR MISA to check if the "F" instruction exists or not and sets the required UART baud rate according to the system clock frequency.

4.3.2 Floating Point Instructions: "mmRISC_SampleFPU"

In this Eclipse project property, select "C/C++ Build"→"Settings"→"Tool Settings"→"Target Processor". Then, choose "Single precision extension (RVF)" in the Floating point field and "Single precision (f)" in the Floating point ABI field.

This program periodically calculates the floating sinf() functions and displays the results in the serial terminal by using printf() and on the 7-segment LED as a decimal number. During this operation, the periodic MTIME interrupt increments 10 discrete LEDs like in the "mmRISC_SampleCPU" program.

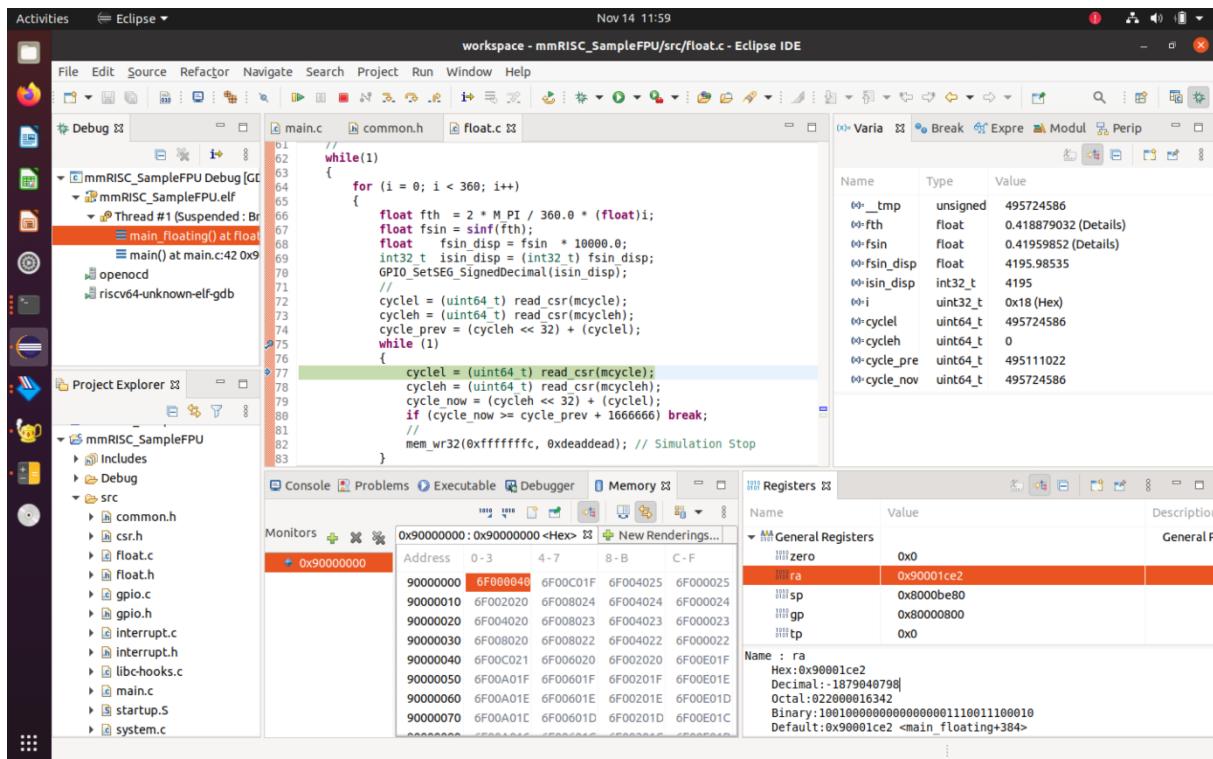


Figure 4.11: Eclipse Windows

4.3.3 Verifications of Floating Point Instructions: "mmRISC_Floating"

This project is a program to verify the Floating Point Operations. Please refer to Section 3.4 for more detailed information.

4.3.4 FreeRTOS Porting "mmRISC_FreeRTOS"

This program demonstrates the FreeRTOS porting for mmRISC-1. It implements a typical demo application of the FreeRTOS "Blinky" as a main program. The application sources are in the directory "src_app" and the RTOS kernel sources are in the directory "src_kernel". Please launch the terminal window before running this demo.

4.3.5 Dhrystone benchmark "mmRISC_Dhrystone"

This is the Dhrystone benchmark program ported to mmRISC-1. Please launch the terminal window before running this benchmark. You will see the benchmark result on it. The calculated scores are shown in Table 4.6.

4.3.6 Coremark benchmark "mmRISC_Coremark"

This is the Coremark benchmark program ported to mmRISC-1. Please launch the terminal window before running this benchmark. You will see the benchmark result on it. The calculated scores are shown in Table 4.6.

Bench-mark	Compiler	Compile Options	Result	Memory
Dhrystone	riscv64-unknown -elf-gcc 10.2.0	-O3	1.55 DMIPS/MHz	RAM: 1cyc R/W
	riscv64-unknown -elf-gcc 10.2.0	-O3 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -flto	3.60 DMIPS/MHz	RAM: 1cyc R/W
	riscv64-unknown -elf-gcc 12.2.0	-O3 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -flto	3.25 DMIPS/MHz	RAM: 1cyc R/W
Coremark	riscv64-unknown -elf-gcc 10.2.0	-Ofast -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las	2.76 Core-mark/MHz	RAM: 1cyc R/W
	riscv64-unknown -elf-gcc 12.2.0	-Ofast -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las	2.51 Core-mark/MHz	RAM: 1cyc R/W
	IAR Embedded Workbench IDE RISC-V 3.30.1 (Evaluation Version)	Level=High Speed, lb no size constraints	3.30 Core-mark/MHz	RAM: 1cyc R/W

Table 4.6: Scores of Benchmarks of mmRISC-1 core

4.3.7 A Retro Text Video Game "mmRISC_StarTrek"

This is the retro text video game StarTrek. You can play it on the terminal window as shown in Listing 4.5. This program is based on "Super Star Trek Classic (v1.1)" ported by Chris Nystrom in 1996.

```
*****
*          *
*          *
*      * * Super Star Trek * *
*          *
*          *
*****
```

Do you need instructions (y/n): y

```

...
-----*-----
----- / /
\----- --
-----'

The USS Enterprise --- NCC - 1701

Your orders are as follows:

Destroy the 41 Klingon warships which have invaded
the galaxy before they can attack Federation Headquarters
on stardate 2641. This gives you 41 days. There is
1 starbase in the galaxy for resupplying your ship.

Hit any key to accept command.
Your mission begins with your starship located
in the galactic quadrant Rigel I.

Combat Area Condition Red
Shields Dangerously Low
-----
<*>      *      Stardate      2600
          Condition    *RED*
          Quadrant     2, 1
+K+       Sector      1, 2
          Photon Torpedoes -2013265672
*          Total Energy   10
          Shields      0
          Klingons Remaining 41
-----
Command?

```

Listing 4.5: Star Trek Game

4.3.8 A Touch LCD Demo "mmRISC_TouchLCD"

The FPGA board DE10-Lite can accommodate the Arduino shield. This demo program supports the following two shields.

(1) Adafruit 2.8" TFT Touch Shield for Arduino with Resistive Touch Screen (Product ID 1651); LCD Controller=ILI9341 (SPI I/F), Resistive Touch Controller=STMPE610 (SPI I/F)

(2) Adafruit 2.8" TFT Touch Shield for Arduino with Capacitive Touch Screen (Product ID 1947); LCD Controller=ILI9341 (SPI I/F), Capacitive Touch Controller=FT6206 (I2C I/F)

At the early stage of the demo program execution, it checks the existence of the LCD Touch shield and the type of touch controller by accessing the control chip registers. If no LCD Touch shield is found, this demo program toggles LEDs and outputs 3D Acceleration Sensor data to the UART terminal, instead. For the LCD Touch shield with resistive touch, if you press KEY1 at startup, a screen for touch panel calibration will appear. Please touch the center of each small cross section mark that appears on each corner of the LCD screen. For the LCD Touch shield with capacitive touch, calibration is not required.

The following graphic routines are prepared in "touchlcd_tft.c". You can easily understand how to use these functions by reading the source code.

Pixel	:	LCD_Draw_Pixel()
Line	:	LCD_Draw_Line()
Rectangle	:	LCD_Draw_Rect(),
		LCD_Fill_Rect()

```

Circle           : LCD_Draw_Circle() ,           LCD_Fill_Circle()
Circle Quadrant: LCD_Draw_Circle_Quadrant() , LCD_Fill_Circle_Quadrant()
Round Rectangle: LCD_Draw_Round_Rect() ,       LCD_Fill_Round_Rect()
Triangle        : LCD_Draw_Triangle() ,        LCD_Fill_Triangle()
Text            : LCD_Draw_Char() ,           LCD_Draw_String()
Image (BMP)     : LCD_Draw_Image()

```

For the image data, you can generate the C source that has the array data from BMP24 by using a perl script mmRISC_TouchLCD/img/bmp2rgb.pl.

Figure 4.12 shows how the demo program works. The image on the top of the screen is generated from ./img/logo.bmp and drawn by LCD_Draw_Image(). The strings on the middle of the screen are drawn by LCD_Draw_String() and each character is an 8x8 size font defined in ./src/font.h. The bottom-left portion of the screen is a simple painting demo, and the bottom-right portion of the screen shows a bouncing ball controlled by the 3D Acceleration sensor output. This demo supports both panels shown in Figure 4.13.

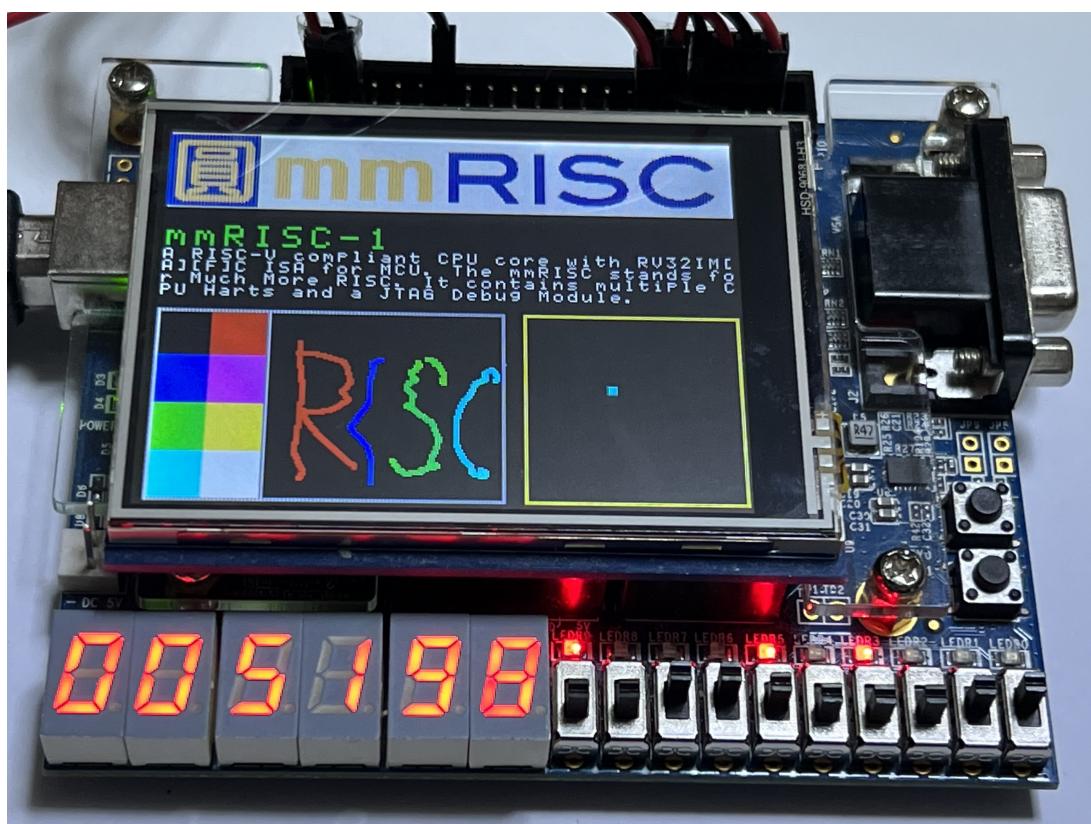


Figure 4.12: Demo Program "mmRISC_TouchLCD"

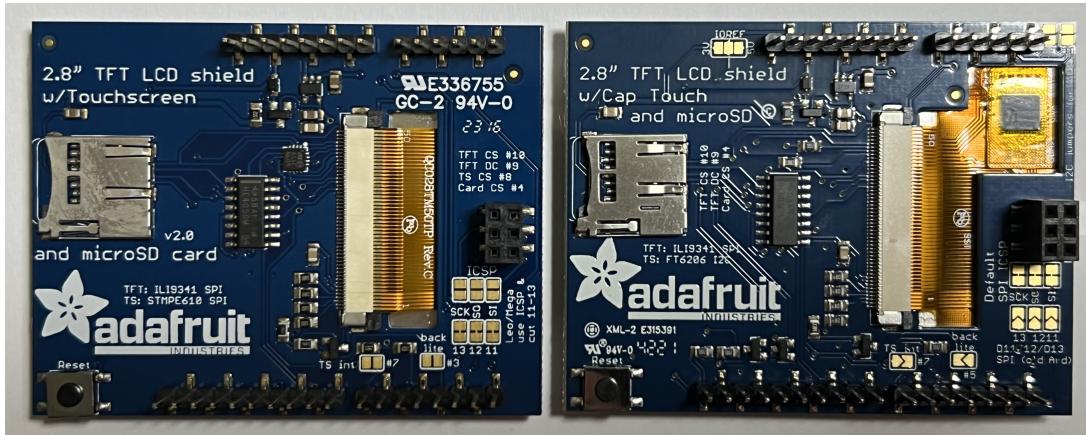


Figure 4.13: Adafruit 2.8" TFT Touch Shield for Arduino (Left: Product ID 1651 Resistive Touch Version; Right: Product ID 1947 Capacitive Touch Version)

4.3.9 A Touch LCD Demo "mmRISC_TicTacToe"

This demo program demonstrates a Tic-Tac-Toe AI game; human vs computer. It also supports the Touch LCD shield of the Arduino, either the Resistive Touch type (Product ID 1651) or the Capacitive Touch type (Product ID 1947). At the early stage of the demo program execution, it checks the existence and the type of the LCD Touch shield by accessing the control chip registers. If the LCD Touch shield is detected, you can play the game on the LCD screen with touch interface as shown in Figure 4.14 and Figure 4.15. If the LCD Touch shield is not detected, you can play the game on UART Terminal (8N1, 115200bps) as shown in Listing 4.6.



Figure 4.14: Startup Screen of Tic-Tac-Toe Game

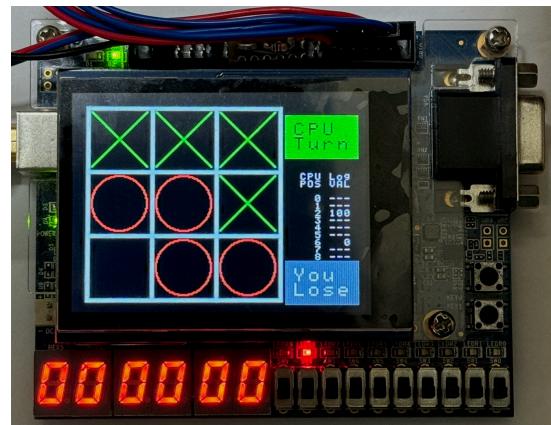


Figure 4.15: Ending Screen of Tic-Tac-Toe Game

```
==== Tictactoe Game ====
1st or 2nd [1-2] ? 1
... (012)
... (345)
... (678)
[You] Move to [0-8] ? 4
... (012)
... (345)
... (678)
[CPU] Move to 0
X.. (012)
```

```
.0.  (345)
...  (678)

...
[You] Move to [0-8] ? 8
X.0  (012)
00X  (345)
X.0  (678)
[CPU] Move to 1
XX0  (012)
00X  (345)
X.0  (678)
[You] Move to [0-8] ? 7
XX0  (012)
00X  (345)
X00  (678)
-----
--- Draw! ---
```

Listing 4.6: Tic-Tac-Toe on UART Terminal

Chapter 5

Referenced Documents

5.1 Referenced Documents

- [1] The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213 : <https://riscv.org/technical/specifications/>
- [2] The RISC-V Instruction Set Manual Volume II: Privileged Architecture Document Version 20190608-Priv-MSU-Ratified : <https://riscv.org/technical/specifications/>
- [3] RISC-V External Debug Support Version 0.13.2 : <https://riscv.org/technical/specifications/>
- [4] AMBA® 3 AHB-Lite Protocol v1.0, June 2006, ARM Ltd.
- [5] DE10-Lite User Manual v1.4, Nov 2016, Terasic Inc.
- [6] FT2232D DUAL USB TO SERIAL UART/FIFO IC Datasheet, Document No. FT_000173, Version 2.05, Future Technology Devices International Limited
- [7] ARM-USB-OCD rev 1.00/07-2006, OLIMEX Ltd.
- [8] OpenCores Simple Asynchronous Serial Controller (SASC) : <https://opencores.org/projects/sasc>
- [9] OpenCores I2C Master controller : <http://www.opencores.org/projects/i2c/>
- [10] OpenCores SPI Interface : https://opencores.org/projects/simple_spi
- [11] ModelSim® Starter Edition Command Reference Manual, Software Version 10.4b , 2015, Mentor Graphics Corporation
- [12] Adafruit 2.8" TFT Touch Shield v2 - Capacitive or Resistive, 2021-11-15 : <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-2-8-tft-touch-shield-v2.pdf>

