mmRISC-1

Technical Reference Manual

RISC-V RV32IMAFC Core for MCU

Rev.01

Munetomo Maruyama

Twitter : @Proccessing_Unit

History

Rev.	Date	Author	Description	Note
01	Nov.19, 2021	Munetomo Maruyama	1st Release	

Index

1 mmRISC-1 CPU Core

1.1 Overview

“mmRISC_1” is a RISC-V compliant CPU core with RV32IM[A][F]C ISA for MCU. Overview of mmRISC-1 specifications are 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/FMADD	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 Selectable Convergence Loops (1 to 16, typ 4) FDIV: 11 cycles (loops = 4), FDQRT: 19 cycles (loops = 4)	
Debug Support	External Debug Support Ver.0.13.2 with JTAG Interface Run / Stop / Step Abstract Command to access Register and Memory System Bus Access for Memory Hardware Break Points (Instruction / Data) x 4 Instruction Count Break Point x 1	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 Bus multiplexed with Debugger Abstract Command LR/SC Monitor Bus Debugger System Bus	AHB-Lite
RTL	Verilog-2001 / System Verilog	

Table 1.1:mmRISC-1 Overview

1.2 Block Diagram

Block diagram of mmRISC-1 is shown in Figure 1.1. The mmRISC-1 contains multiple CPU Harts and a Debug Module with JTAG interface. Each Hart has instruction fetch bus, data access bus and monitor bus for LR/SC operation. The Debug Module has dedicated system bus access capability and controls each Hart (run / stop / step, etc.). The Debug Module can access resources in each Hart such as programer'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 instruction fetch unit with pre-fetch buffer, pipeline unit with instruction decode, datapath unit for integer operations, CSR resources, debugger support and floating point operation unit. The monitor bus for LR/SC operation exists only when Atomic ISA is enabled. The floating point operation unit exists only when floating point ISA is enabled.

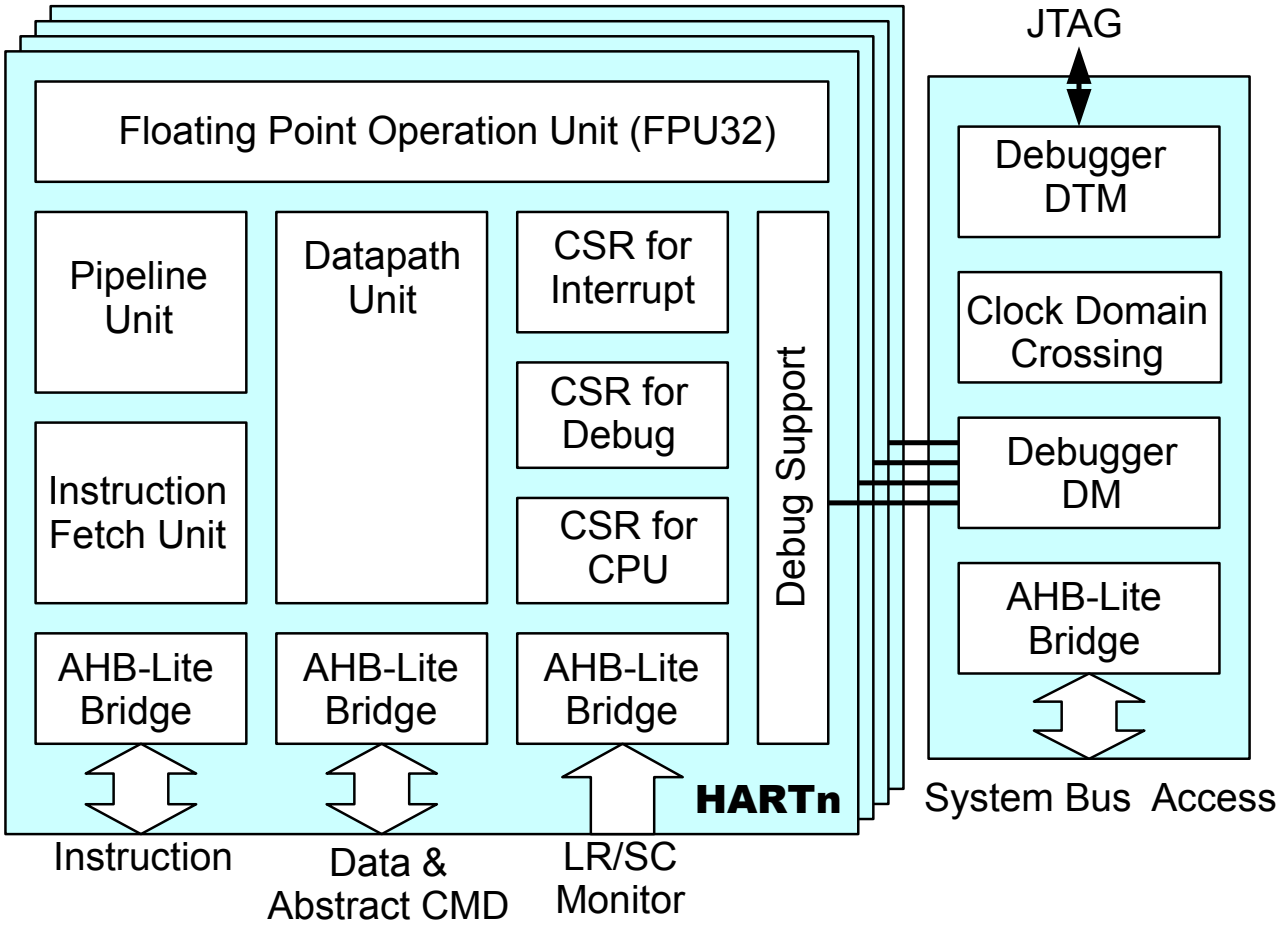

Figure 1.1: Block Diagram

1.3 Programmer's Model

Programmer's model of this core is shown in Figure 1.2. Integer register xn (or sometimes expressed as XRn) and PC are 32bit length. Floating Point register fn (or sometimes expressed as FRn) are also 32bit length which exist in the core only when Floating Point ISA is enabled. The core's XLEN= 32 and FLEN = 32. In addition, Figure 1.3 shows Registers specified by 3bits number for Compressed Instruction Set.

Integer Registers

31		0	
x0	(zero)		Hard wired Zero
x1	(ra)		Return Address
x2	(sp)		Stack Pointer
x3	(gp)		Global Pointer
x4	(tp)		Thread Pointer
x5	(t0)		Temporary
x6	(t1)		Temporary
x7	(t2)		Temporary
x8	(s0/fp)		Saved Register / Frame Pointer
x9	(s1)		Saved Register
x10	(a0)		Function Argument / Return Value
x11	(a1)		Function Argument / Return Value
x12	(a2)		Function Argument
x13	(a3)		Function Argument
x14	(a4)		Function Argument
x15	(a5)		Function Argument
x16	(a6)		Function Argument
x17	(a7)		Function Argument
x18	(s2)		Saved Register
x19	(s3)		Saved Register
x20	(s4)		Saved Register
x21	(s5)		Saved Register
x22	(s6)		Saved Register
x23	(s7)		Saved Register
x24	(s8)		Saved Register
x25	(s9)		Saved Register
x26	(s10)		Saved Register
x27	(s11)		Saved Register
x28	(t3)		Temporary
x29	(t4)		Temporary
x30	(t5)		Temporary
x31	(t6)		Temporary

Floating Point Registers

31		0	
f0	(ft0)		FP Temporary
f1	(ft1)		FP Temporary
f2	(ft2)		FP Temporary
f3	(ft3)		FP Temporary
f4	(ft4)		FP Temporary
f5	(ft5)		FP Temporary
f6	(ft6)		FP Temporary
f7	(ft7)		FP Temporary
f8	(fs0)		FP Saved Register
f9	(fs1)		FP Saved Register
f10	(fa0)		FP Argument / Return Value
f11	(fa1)		FP Argument / Return Value
f12	(fa2)		FP Argument
f13	(fa3)		FP Argument
f14	(fa4)		FP Argument
f15	(fa5)		FP Argument
f16	(fa6)		FP Argument
f17	(fa7)		FP Argument
f18	(fs2)		FP Saved Register
f19	(fs3)		FP Saved Register
f20	(fs4)		FP Saved Register
f21	(fs5)		FP Saved Register
f22	(fs6)		FP Saved Register
f23	(fs7)		FP Saved Register
f24	(fs8)		FP Saved Register
f25	(fs9)		FP Saved Register
f26	(fs10)		FP Saved Register
f27	(fs11)		FP Saved Register
f28	(ft8)		FP Temporary
f29	(ft9)		FP Temporary
f30	(ft10)		FP Temporary
f31	(ft11)		FP Temporary

pc	Program Counter

Figure 1.2:Programmer's Model

RVC Reg#	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 1.3:Registers specified by 3bits number for Compressed Instruction Set

1.4 Configurable Options

Several specifications of mmRISC-1 core are configurable as shown in Table 1.2. Each configuration is specified by parameter definition (`define) or logic level of input signals of mmRISC-1 core.

Where	Class	Name	Description	Default
defines.v	`define	`RISCV_ISA_RV32F	Single Floating Point ISA	enabled
defines.v	`define	`RISCV_ISA_RV32A	Atomic Access ISA	enabled
defines.v	`define	`JTAG_IDCODE	JTAG ID Code	32'h16d6d001 Version = 0x1 PartNo.=0x6d6d(“mm”) ManuID = 0
defines.v	`define	`MVENDORID	CSR MVENDORID	32'h00000000
defines.v	`define	`MARCHID	CSR MARCHID	32'h6d6d3031
defines.v	`define	`MIMPID	CSR MIMPID	32'h00000010
defines.v	`define	`TRG_CH_BUS	Count of Type 2 Trigger ch (Type 3 Trigger is 1ch)	4
defines.v	`define	`HART_COUNT	Hart Counts in mmRISC	1
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	1’b0 (not secure)
mmRISC.v	input signal	DEBUG_SECURE_CODE	Debug Secure Code to be matched with CSR AUTHDATA	32’h12345678

Table 1.2: Configurable Options

1.5 RTL Files and Structure

RTL files of mmRISC-1 are shown in Table 1.3.

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

Table 1.3: RTL files

RTL structure in mmRISC-1 is shown in Figure 1.4. The top layer of mmRISC-1 (mmRISC.V) has two large domains, one is multiple CPU (Hart) blocks (cpu_top.v) and the another is Debug block (debug_top.v). As for the periodic interrupt timer MTIME, it is defined in CSR but it is outside of CPU core because it is connected system bus as a memory mapped peripheral.

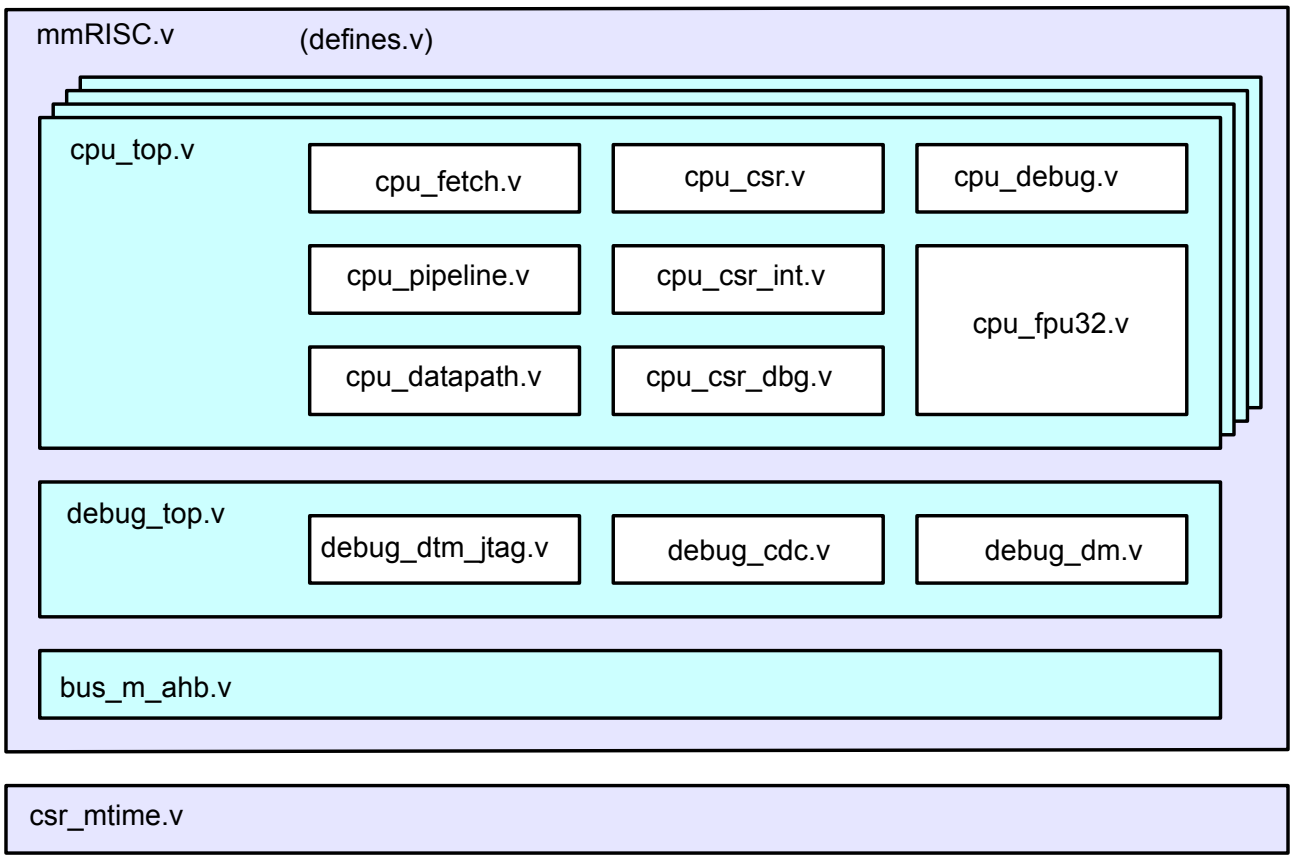

Figure 1.4: RTL Structure

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, the hart is shown as unavailable.	

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	

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	Debug Secure Code to be matched with CSR AUTHDATA	

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

Group	Direction	Width	Name	Description	Note
Instr Bus	output		CPUI_M_SEL [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Select	
Instr Bus	output	[1:0]	CPUI_M_HTRANS [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Transfer	
Instr Bus	output		CPUI_M_HWRITE [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Write	
Instr Bus	output		CPUI_M_HMASTLOCK [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Master Lock	
Instr Bus	output	[2:0]	CPUI_M_HSIZE [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Size	
Instr Bus	output	[2:0]	CPUI_M_HBURST [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Burst	
Instr Bus	output	[3:0]	CPUI_M_HPROT [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Protection	
Instr Bus	output	[31:0]	CPUI_M_HADDR [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Address	
Instr Bus	output	[31:0]	CPUI_M_HWDATA [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Write Data	
Instr Bus	output		CPUI_M_HREADY [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Ready Output	
Instr Bus	input		CPUI_M_HREADYOUT [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Ready Input	
Instr Bus	input	[31:0]	CPUI_M_HRDATA\ [0:`HART_COUNT-1]	Instruction Bus; AHB-Lite Read Data	
Instr 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_HMASTLOCK [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_HRDATA [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	
Atomic	input		CPUM_S_HREADYOUT [0:`HART_COUNT-1]	Atomic Monitor Bus; AHB-Lite Readyout	

Table 1.9: Input / Output Signals of mmRISC-1 (Atomic LR/SC Monitor, available when `RISCV_ISA_RV32A is defined)

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

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

Group	Direction	Width	Name	Description	Note
Interrupt	input		IRQ_EXT	External Interrupt	
Interrupt	input		IRQ_MSOFT	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)

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 Control)

1.7 Clock and 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).

Reset Structure of mmRISC-1 is slightly complicated as shown in Figure 1.5. RES_ORG is an 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 whole chip and peripherals. HART_RESET[] are connected each CPU (Hart) and can independently reset each hart. In each CPU (Hart), one of the HART_RESET is renamed to RES_CPU, and the RES_CPU resets whole area in CPU block. In chip top layer, SRSTn is bi-directional inout signal. The SRSTn input is used to reset whole chip except for debug block, and SRSTn output is generated from debug block.

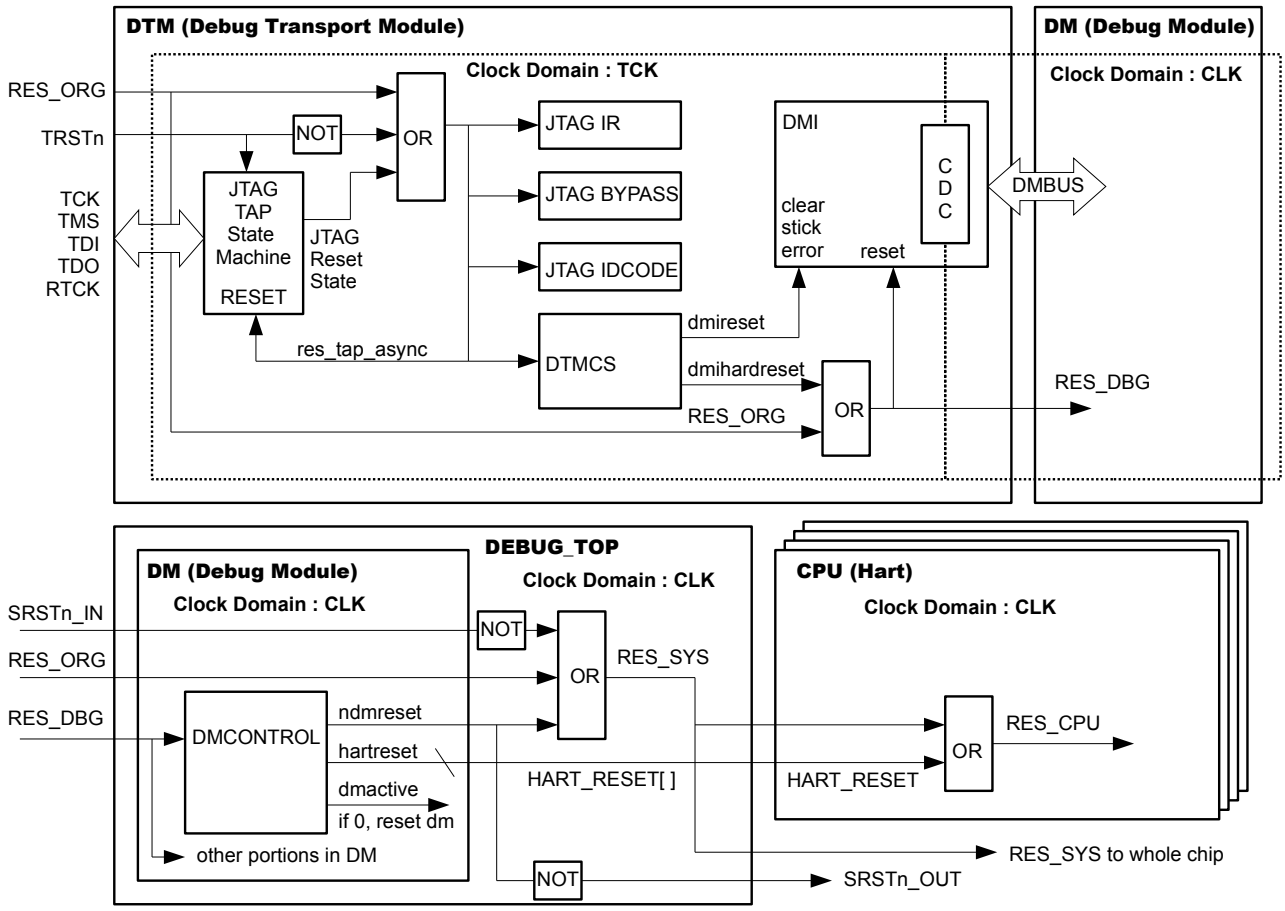

Figure 1.5: Reset Structure

***NOTE**: An input signal STBY (Stand-by) is prepared to inform CPU that the chip enters low power mode, and the STBY is reflected to hart available status implemented in DMSTATUS register of DM (Debug Module), that is if STBY is 1, then hart status becomes "unavailable". Please note that DMSTATUS register and JTAG DTM (Debug Transport Module) require toggling CLK (System Clock), so if CLK stops during STBY state, JTAG Port can not access DMSTATUS. The specification of Stand-by state with stopped CLK still needs more consideration.*

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 is invoked by instructions, and the other is asynchronous exception which is 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 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) MTVAL is set specific data corresponding to each exception.

(6) MCAUSE is set 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 MTVAL 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 manner.

Cause and Vector						
Group	MCAUSE		Vector Offset		Description	MTVAL
	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	0x0b	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
	0x03	Load/Store/AMO Address Break Point
	0x06	Store/AMO(Atomic Memory Operation) Address Misaligned
Lower	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 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 IRQ_EXT > IRQ_SOFT > IRQ_MTIME > IRQ[63:0].

64 IRQ inputs are controlled by INTC (Interrupt Controller) shown in Figure 1.6. 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. An example of startup routine including exception hander and interrupt hander is shown inList 1.1. Note that the sample routine ignores saving and storing Floating Point Registers FRx.

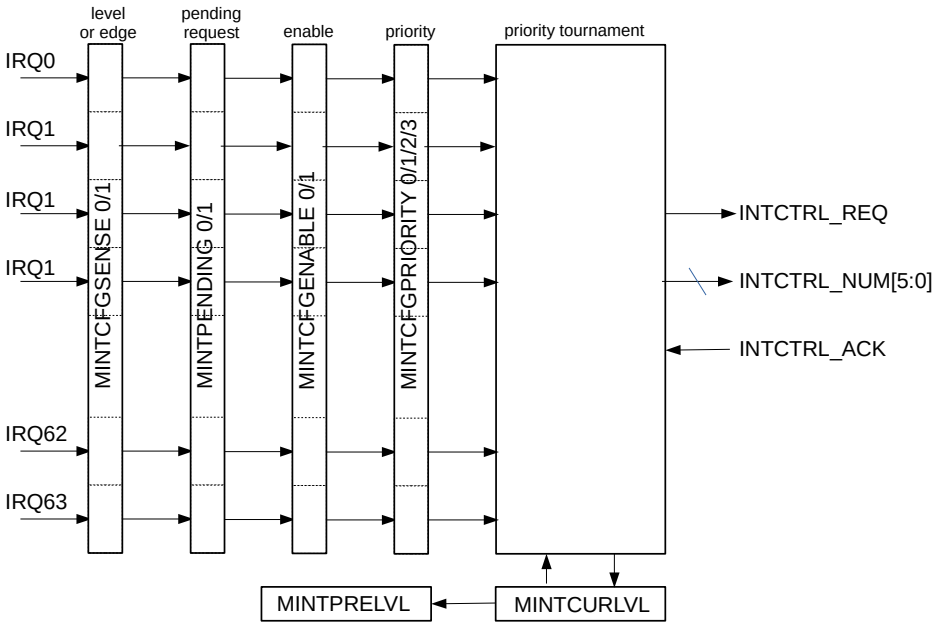

Figure 1.6: INTC (Interrupt Controller)

```
_reset:
    j    _start
_vector_base:
    j    _trap_exception // 00:Exception
    j    _trap_illegal   // 01:Reserved
    j    _trap_illegal   // 02:Reserved
    j    _trap_int_soft  // 03:IRQ_MSOFT
    j    _trap_illegal   // 04:Reserved
    j    _trap_illegal   // 05:Reserved
    j    _trap_illegal   // 06:Reserved
    j    _trap_int_timer // 07:IRQ_MTIME
    j    _trap_illegal   // 08:Reserved
    j    _trap_illegal   // 09:Reserved
    j    _trap_illegal   // 10:Reserved
    j    _trap_int_ext   // 11:IRQ_EXT
    j    _trap_illegal   // 12:Reserved
    j    _trap_illegal   // 13:Reserved
    j    _trap_illegal   // 14:Reserved
    j    _trap_illegal   // 15:Reserved
    j    _trap_irq       // 16:IRQ00
    j    _trap_irq       // 16:IRQ01
    j    _trap_irq       // 16:IRQ02
    j    _trap_irq       // 16:IRQ03
    ...
    j    _trap_irq       // 16:IRQ63
//
_trap_irq:
    csrrw ra, mscratch, ra  // swap ra and macratch
    jal   _trap_handler_save
    jal   _trap_handler_irq
    jal   _trap_handler_load
    csrrw ra, mscratch, ra  // swap ra and macratch
    mret
//
_trap_exception:
    csrrw x1, mscratch, x1  // swap ra and macratch
    jal   _trap_handler_save
    jal   _trap_handler_exception
    jal   _trap_handler_load
    csrrw ra, mscratch, ra  // swap ra and macratch
    mret
//
_trap_int_soft:
    csrrw ra, mscratch, ra  // swap ra and macratch
    jal   _trap_handler_save
    jal   _trap_handler_int_soft
    jal   _trap_handler_load
    csrrw ra, mscratch, ra  // swap ra and macratch
    mret
//
_trap_int_timer:
    csrrw ra, mscratch, ra  // swap ra and macratch
    jal   _trap_handler_save
    call  INT_Timer_Handler
    jal   _trap_handler_load
    csrrw ra, mscratch, ra  // swap ra and macratch
    mret
//
_trap_int_ext:
    csrrw ra, mscratch, ra  // swap ra and macratch
    jal   _trap_handler_save
    jal   _trap_handler_int_ext
    jal   _trap_handler_load
    csrrw ra, mscratch, ra  // swap ra and macratch
    mret
//
_trap_illegal:
   j     .
//
_trap_handler_save:
    csrrw ra, mscratch, ra  // swap ra and macratch
    addi  sp, sp, -124
    sw    x1,    0(sp) // ra
    sw    x2,    4(sp) // sp
    sw    x3,    8(sp)
    sw    x4,   12(sp)
    ...
    sw    x30, 116(sp)
    sw    x31, 120(sp)
    csrrw ra, mscratch, ra  // swap ra and macratch
    ret
//
_trap_handler_load:
    csrrw ra, mscratch, ra  // swap ra and macratch
    lw    x1,    0(sp) // ra
    lw    x2,    4(sp) // sp
    lw    x3,    8(sp)
    lw    x4,   12(sp)
    ...
    lw    x30, 116(sp)
    lw    x31, 120(sp)
    addi  sp, sp, 124
    csrrw ra, mscratch, ra  // swap ra and macratch
    ret
_trap_handler_irq:
    // save PC, MSTATUS,mintprelvl
    csrr  x8, mepc
    csrr  x9, mstatus
    csrr  x10, 0xbf1 //mintprelvl
    addi  sp, sp, -16
    sw    x1 , 0(sp)
    sw    x8 , 4(sp)
    sw    x9 , 8(sp)
    sw    x10,12(sp)
    // enable global interrupt (set mie)
    csrrsi x15, mstatus, 0x08
    // allow enested interrupt
    call   INT_IRQ_Handler         // Defined in C Program
    // disable global interrupt (clear mie)
    csrrci x15, mstatus, 0x08
    // load MSTATUS, PC, mintcurlvl
    lw    x1 , 0(sp)
    lw    x8 , 4(sp)
    lw    x9 , 8(sp)
    lw    x10,12(sp)
    addi  sp, sp, 16
    csrw  0xbf0, x10 // mintcurlvl
    csrw  mstatus, x9
    csrw  mepc, x8
    // Return
    ret
//
_trap_handler_exception:
    // As well as _trap_handler_irq
    ret
//
_trap_handler_int_soft:
    // As well as _trap_handler_irq
    ret

_trap_handler_int_timer:
    // As well as _trap_handler_irq
    ret

_trap_handler_int_ext:
    // As well as _trap_handler_irq
    ret
//
// Start Up Routine
//
_start:
    //
    // Init GP and SP
    la  gp, __GLOBAL_PTR__
    la  sp, __STACK_TOP__
    //
    // Copy Initial Data
    la  a0, __ROM_INIT_BGN__  // Defined in Linker Script
    la  a1, __RAM_INIT_BGN__  // Defined in Linker Script
    la  a2, __RAM_INIT_END__  // Defined in Linker Script
    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    3
```

List 1.1:An Example of Startup Routine including Exception / Interrupt Handlers

1.10 Memory Model

Memory Model of mmRISC-1 is based on Little Endian manner as shown in Figure 1.7.

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 memory alignment exception.

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

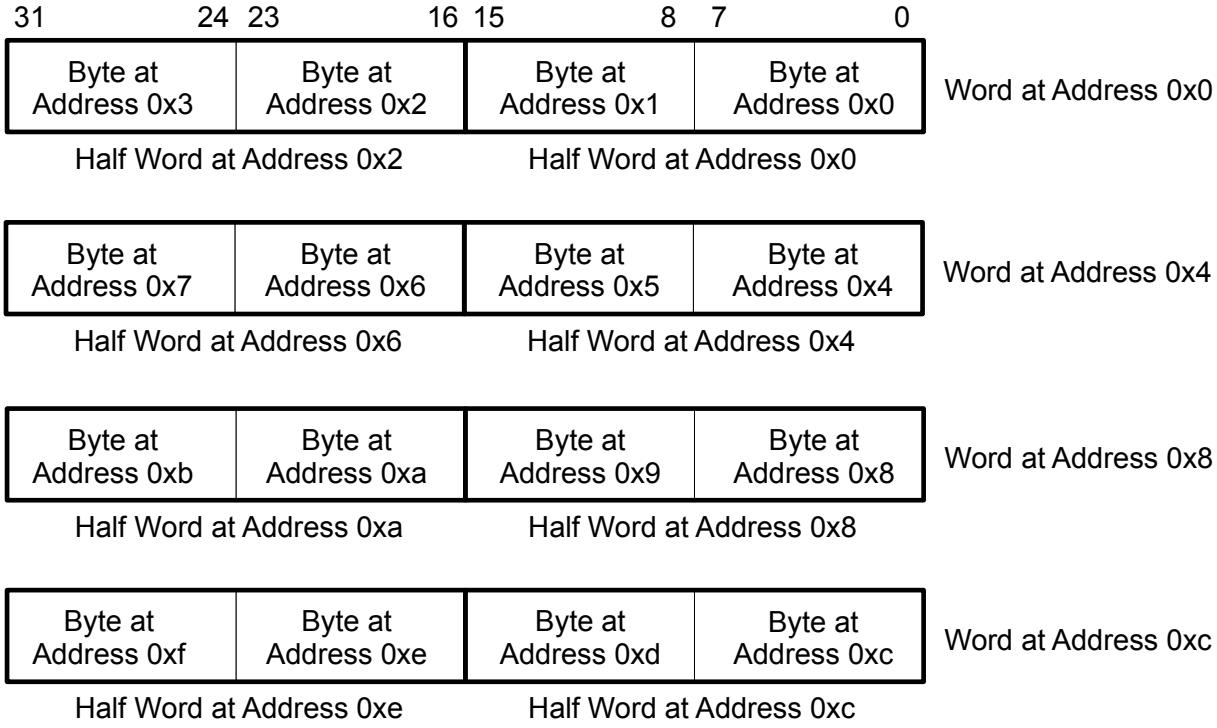

Figure 1.7:Little Endian Memory Organization

1.11 CSR Map

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	MCYCLEH	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	CYCLEH	Read-Only Mirror of MCYCLEH
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:Debug 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
0x7a3	TINFO	Trigger Info

Table 1.19:Debug Trigger Module

1.12 Other Register Map except for CSR

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	MTIMECMPH	MTIME Timer Ticks Comparator (higher 32bits)
0x0018	MSOFTIRQ	Software Interrupt Request

Table 1.20:Machine Mode MTIME (Memory Mapped Registers; So far, recommend 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	HALTSUM0	Halt Summary 0
0x13	HALTSUM1	Halt Summary 1
0x34	HALTSUM2	Halt Summary 2
0x35	HALTSUM3	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) Registers : Abstract Command; 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) Registers : System Bus Access; 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 sate 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 periodic interrupt timer, is defined as memory mapped peripheral. Timer counter is 64bit length; its high side is MTIMEH and lower side is MTIME. Count increment clock source can be selected from internal system clock or external clock of the IP. Timer comparator is also 64bit length; its high side is MTIMECMPH and lower side is

MTIMECMP. When 64bit timer counter is matched to timer comparator, interrupt request is generated. MTIME_CTRL has timer enable bit, clock select bit, interrupt enable bit and interrupt status bit. MTIME_DIV is pre-scalar configuration of count clock. MSOFTIRQ can be used to generate IRQ_SOFT interrupt request.

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

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

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

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

Table 1.44:MTIMECMP / MTIMECMPH

Offset Address	Name	Description			
0x0018	MSOFTIRQ	Software Interrupt Request			
Bit	Field	Initial	Access	Description	Note
31:1	0	31'h0	R	Always 0.	
0	msip	1'b0	R/W	Machine Software Interrupt Pending (Request) 0: Negate Software Interrupt 1: Assert Software Interrupt	

Table 1.45:MSOFTIRQ

1.15 INTC (Interrupt Controller) CSR

Address	Name	Description			
0xbf0	MINTCURLVL	Machine Interrupt Current Level			
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			
0xbf1	MINTPRELVL	Machine Interrupt Previous Level			
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			
0xbf2	MINTCFGENABLE0	Machine Interrupt Configuration Enable 0 (x = 0)			
0xbf3	MINTCFGENABLE1	Machine Interrupt Configuration Enable 1 (x = 1)			
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			
0xbf4	MINTCFGSENSE0	Machine Interrupt Configuration Sense 0 (x = 0)			
0xbf5	MINTCFGSENSE1	Machine Interrupt Configuration Sense 1 (x = 1)			
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			
0xbf6	MINTPENDING0	Machine Interrupt Pending 0 (x = 0)			
0xbf7	MINTPENDING1	Machine Interrupt Pending 1 (x = 1)			
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			
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)			
Bit	Field	Initial	Access	Description	Note
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:MINTCFGPRIORITY 0/1/2/3/4/5/6/7

1.16 Floating Point CSR

Floating Point Unit of mmRISC-1 uses Goldschmidt's Algorithm for FDIV.S and FSQRT.S instruction. A CSR FCONV, which is an dedicated CSR of mmRISC-1, configures convergence loop count for each FDIV and FSQRT.

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			
0x7b0	DCSR	Debug Control and Status			
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	mprven	1'b0	R	Always 0. 0: MPRV in mstatus is ignored in Debug Mode.	
3	nmip	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			
0x7b1	DPC	Debug PC			
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. ie. 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 re. 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			
0x7a0	TSELECT	Trigger Select			
Bit	Field	Initial	Access	Description	Note
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			
0x7a1	TDATA1	Trigger Data 1			
Bit	Field	Initial	Access	Description	Note
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			
0x7a2	TDATA2	Trigger Specific Data			
Bit	Field	Initial	Access	Description	Note
31:0	data	32'h0	R/W	Trigger-specific data associated to mcontrol.	

Table 1.60: TDATA2

Address	Name	Description			
0x7a3	TINFO	Trigger Info			
Bit	Field	Initial	Access	Description	Note
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 con-taining 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 JTAG register is specified by 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.	
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			
0x11	DMI	Debug Module Interface Access This register allows access to the Debug Module Interface (DMI). In Update-DR, the DTM starts the operation specified in op unless the current status reported in op is sticky. In Capture-DR, the DTM updates data with the result from that operation, updating op if the current op isn't sticky.			
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	When the debugger writes this field, it has the following meaning: 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. 1: Read from address. (read) 2: Write data to address. (write) 3: Reserved. When the debugger reads this field, it means the following: 0: The previous operation completed successfully. 1: Reserved. 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. 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.	

Table 1.67:DMI

1.20 DM (Debug Module) Register

The DM (Debug Module) registers are accessed by DMI bus from DTM. The address is specified by DMI register according to 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	ackhaveresest	-	W1	0: No efect. 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 clrresethaltreq 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 clrresethaltreq to clear it. Writes apply to the new value of hartsel and hasel. If hasresethaltreq is 0, this field is not implemented.	
2	clrresethaltreq	-	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 Address	Name Name	Description 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 DMI bus from DTM. The address is specified by DMI register according to JTAG access.

Address	Name	Description			
0x17	COMMAND	Abstract Command : Access Register			
Bit	Field	Initial	Access	Description	Note
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 1.78:COMMAND for Access Register

Address	Name	Description			
0x17	COMMAND	Abstract Command : Access Memory			
Bit	Field	Initial	Access	Description	Note
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 1.79:COMMAND for Access Memory

Address	Name	Description			
0x16	ABSTRACTCS	Abstract Control and Status			
Bit	Field	Initial	Access	Description	Note
31:29	0	3'b000	R	Always 0.	
28:24	progbufsize	5'b00000	R	Always 0. Program Buffer size 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 writ- ing 1 to them. No abstract command is started until the value is reset to 0. This eld 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 1.80:ABSTRACTCS

1.22 DM (Debug Module) Register : System Bus Access

The DM (Debug Module) registers are accessed by DMI bus from DTM. The address is specified by DMI register according to 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 eld 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 1.81: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 1.82: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 1.83:SBDATA0

1.23 CPU Pipeline Structure

For mmRISC-1 core, the RV32IMAC instruction pipelines consist of 3-stages to 5-stage as shown in Figure 1.8. Instructions are pre-fetched and queued in instruction fetch unit which is expressed as “F”, and sent to next decode state “D”. Execution stage is shown by “E”, data memory access stage is “M” and 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 non-restoring method and take several clocks to complete.

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

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

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

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

Detail CPU pipeline logic structure is shown in Figure 1.9.

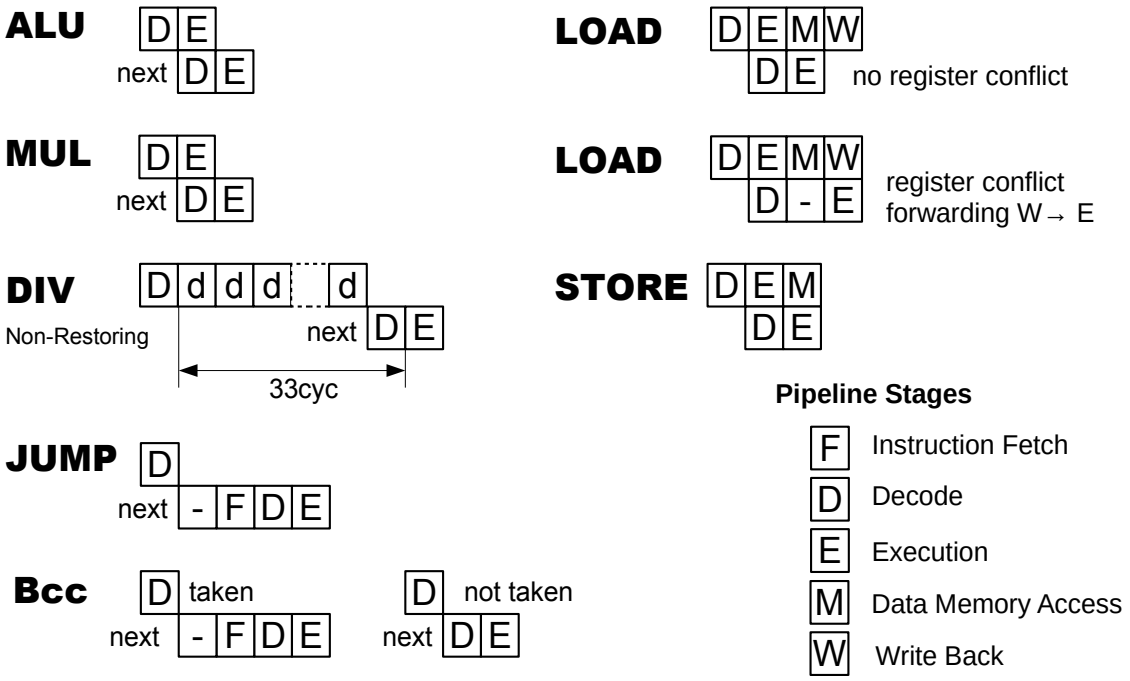

Figure 1.8: CPU Pipeline

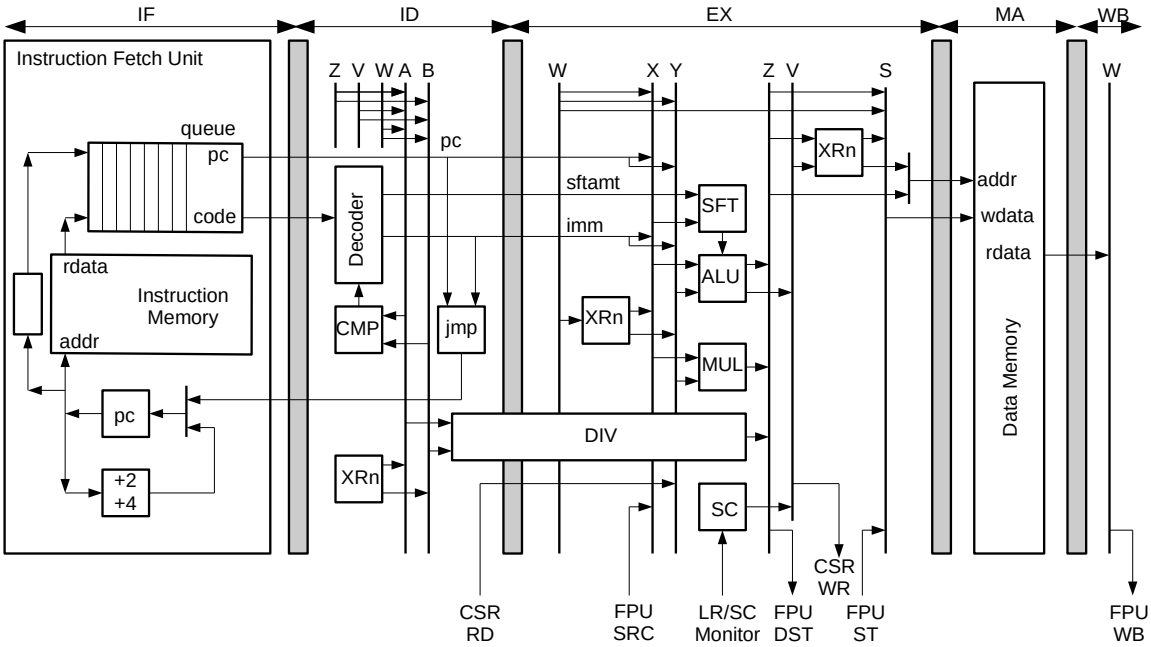

Figure 1.9: CPU Pipeline Logic Structure

1.24 FPU Pipeline Structure

For mmRISC-1 core, RV32F/RV32FC (FPU) instructions have independent pipeline to CPU (integer). If there are no register conflict between CPU and 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. 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 1.10, FPU pipelines consist of 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 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 near future. The “a” stage execute addition and subtraction between the internal format. The “m” stage execute multiplication between the internal format. The “f” stage means “finalization” that converts internal number format to 32bit IEEE754 format and stores the result to 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, FNMADD.S and FNMSUB.S instructions consist of “i”, “m”, “a” and “f” stages. Among FMADD.S, FMSUB.S, FNMADD.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”, combination of multiple “a” and “m”, and final “f” stage. Each convergence loop counts can be configured by CSR FCONV. For default configuration, 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 dedicated stage for conversion.

Pipelines of instruction for operations between FRn and FRn (FSGNJ.S, FSGNJN.S, FSGNJX.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.

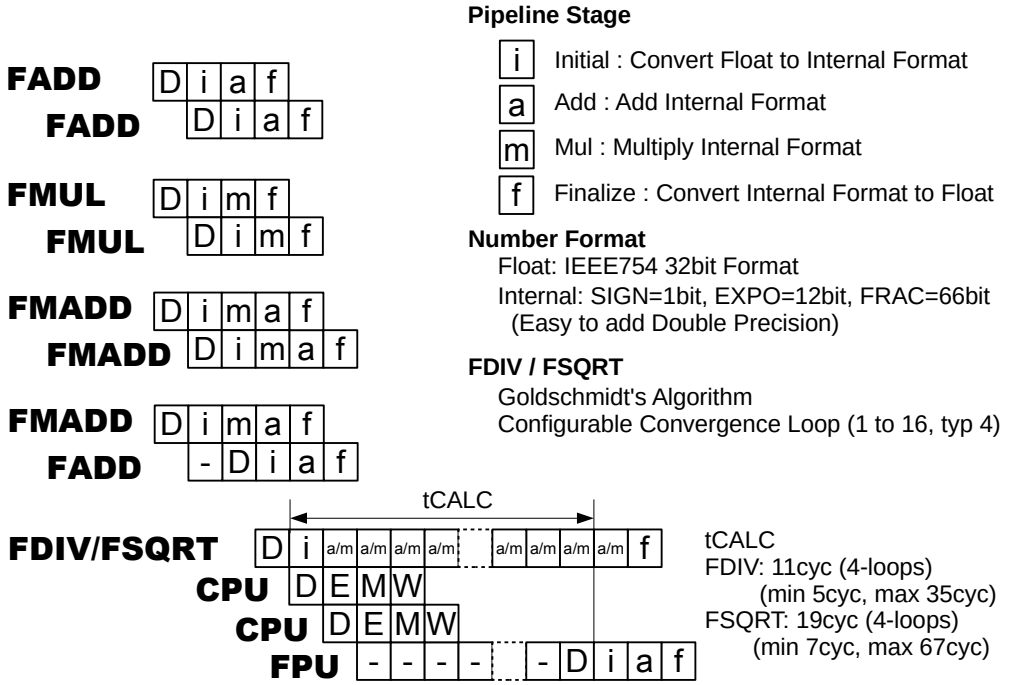

Figure 1.10: FPU Pipeline

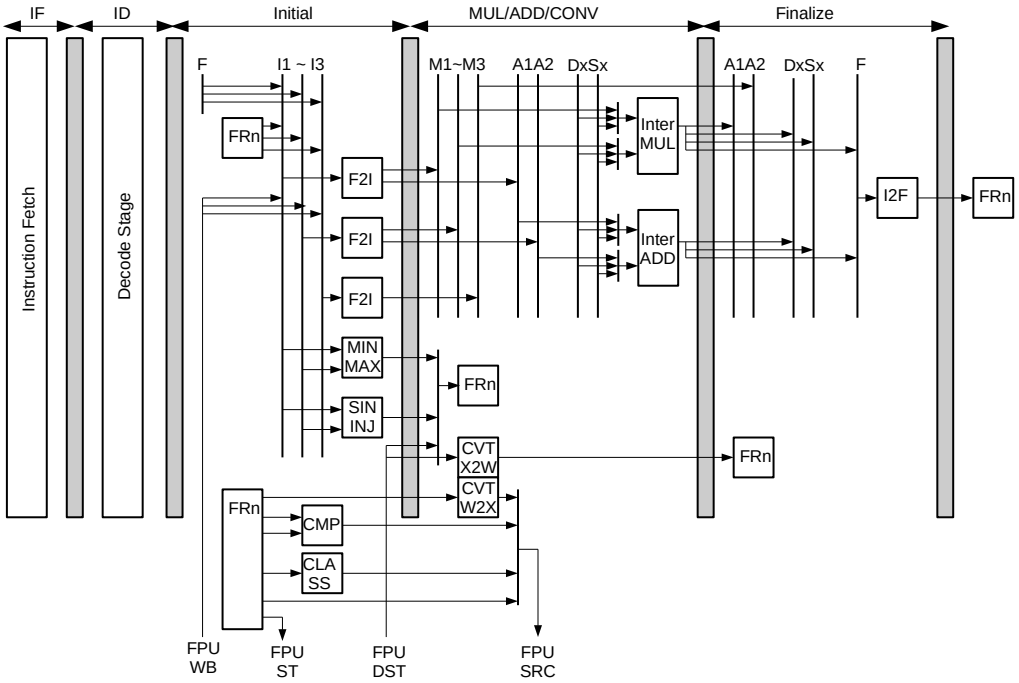

Figure 1.11: FPU Pipeline Logic Structure

1.25 32bit ISA Specifications

RV32I Base Instruction Set			
Class	Mnemonic	Meaning	Operation
Arithmetic	LUI rd, imm	Load Upper Immediate	rd ← {imm, 12'b0}
Arithmetic	AUIPC rd, imm	Add Upper Immediate to PC	rd ← {imm, 12'b0} + PC
Arithmetic	ADD rd, rs1, rs2	Add	rd ← rs1 + rs2
Arithmetic	SUB rd, rs1, rs2	Subtract	rd ← rs1 - rs2
Arithmetic	ADDI rd, rs1, imm	Add Immediate	rd ← rs1 + imm(SE)
Shift	SLL rd, rs1, rs2	Shift Left Logicak	rd ← rs1 << rs2[4:0] Logical
Shift	SRL rd, rs1, rs2	Shift Right Logical	rd ← rs1 >> rs2[4:0] Logical
Shift	SRA rd, rs1, rs2	Shift Right Arithmetic	rd ← rs1 >> rs2[4:0] Arithmetic
Shift	SLLI rd, rs1, shamt	Shift Left Immediate Logical	rd ← rs1 << shamt Logical
Shift	SRLI rd, rs1, shamt	Shift Right Immediate Logical	rd ← rs1 >> shamt Logical
Shift	SRAI rd, rs1, shamt	Shift Right Immediate Arithmetic	rd ← rs1 >> shamt Arithmetic
Logical	XOR rd, rs1, rs2	XOR	rd ← rs1 ^ rs2
Logical	OR rd, rs1, rs2	OR	rd ← rs1 | rs2
Logical	AND rd, rs1, rs2	AND	rd ← rs1 & rs2
Logical	XORI rd, rs1, imm	XOR Immediate	rd ← rs1 ^ imm(SE)
Logical	ORI rd, rs1, imm	OR Immediate	rd ← rs1 | imm(SE)
Logical	ANDI rd, rs1, imm	AND Immediate	rd ← rs1 & imm(SE)
Compare	SLT rd, rs1, rs2	Set Less Than (S)	rd ← (rs1 < rs2) (S)
Compare	SLTU rd, rs1, rs2	Set Less Than (U)	rd ← (rs1 < rs2) (U)
Compare	SLTI rd, rs1, imm	Set Less Than Immediate (S)	rd ← (rs1 < imm(SE)) (S)
Compare	SLTIU rd, rs1, imm	Set Less Than Immediate (U)	rd ← (rs1 < imm(SE)) (U)
Branch	BEQ rs1, rs2, imm	Branch Equal	PC ← PC + imm(SE) if rs1 == rs2
Branch	BNE rs1, rs2, imm	Branch Not Equal	PC ← PC + imm(SE) if rs1 != rs2
Branch	BLT rs1, rs2, imm	Branch Less Than (S)	PC ← PC + imm(SE) if rs1 < rs2 (S)
Branch	BGE rs1, rs2, imm	Branch Greater or Equal (S)	PC ← PC + imm(SE) if rs1 >= rs2 (S)
Branch	BLTU rs1, rs2, imm	Branch Less Than (U)	PC ← PC + imm(SE) if rs1 < rs2 (U)
Branch	BGEU rs1, rs2, imm	Branch Greater or Equal (U)	PC ← PC + imm(SE) if rs1 >= rs2 (U)
Jump & Link	JAL rd, imm	Jump and Link	rd ← PC+4, PC ← PC + imm(SE)
Jump & Link	JALR rd, imm	Jump and Link Register	rd ← PC+4, PC ← rs1 + imm(SE)
Load	LB rd, rs1, imm	Load Byte	rd ← MB[rs1 + imm(SE)] with SE
Load	LH rd, rs1, imm	Load Halfword	rd ← MH[rs1 + imm(SE)] with SE
Load	LW rd, rs1, imm	Load Word	rd ← MW[rs1 + imm(SE)]
Load	LBU rd, rs1, imm	Load Byte Unsigned	rd ← MB[rs1 + imm(SE)] with ZE
Load	LHU rd, rs1, imm	Load Halfword Unsigned	rd ← MH[rs1 + imm(SE)] with ZE
Store	SB rs1, rs2, imm	Store Byte	MB[rs1 + imm(SE)] ← rs2
Store	SH rs1, rs2, imm	Store Halfword	MH[rs1 + imm(SE)] ← rs2
Store	SW rs1, rs2, imm	Store Word	MW[rs1 + imm(SE)] ← 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.ebreamk == 0 MEPC←PC, MSTATUS.mpie←mie ,mie←0, MCAUSE←0x03, MTVAL←PC, PC←MTVEC

Table 1.84: 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 1.85: RV32 Zifencei Specification

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

Table 1.86: 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 1.87: 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 1.88: 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	FSGNJN.S fd, fs1, fs2	Floating Sign Injection Opposite	fd ← {~fs2[31], fs1[30:0]}
F Sign	FSGNJX.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.WU.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 1.89: RV32F Single Precision Floating Point Operation Specification

1.26 16bit ISA Specifications

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

Table 1.91: RV32FC Floating Point Compressed Instruction Specification

C Arithmetic	C.LUI	rd, imm	Load Upper Immediate	rd ← {imm(SE), 12'b0}
C Arithmetic	C.ADD	rd, rs2	Add	rd ← Rd + rs2
C Arithmetic	C.SUB	rd', rs2'	Subtract	rd' ← rd' - rs2'
C Arithmetic	C.ADDI	rd, imm	Add Immediate	rd ← rd + imm(SE)
C Arithmetic	C.ADDI4SPN	rd', uzuimm	Add SP and Imm * 4	rd' ← x2 + imm(ZE)*4
C Arithmetic	C.ADDI16SP	sp, sp, imm	Add SP and Imm * 16	sp(x2) ← xp(x2) + imm(SE)
C Shift	C.SLLI	rd, shamt	Shift Left Logical Immediate	rd' ← rd' << shamt Logical
C Shift	C.SRLI	rd', shamt	Shift Right Logical Immediate	rd' ← rd' >> shamt Logical
C Shift	C.SRAI	rd', shamt	Shift Right Arithmetic Immediate	rd' ← rd' >> shamt Arithmetic
C Logical	C.XOR	rd', rs2'	XOR	rd' ← rd' ^ rs2'
C Logical	C.OR	rd', rs2'	OR	rd' ← rd' | rs2'
C Logical	C.AND	rd', rs2'	AND	rd' ← rd' & rs2'
C Logical	C.ANDI	rd', imm	AND Immediate	rd' ← rd' & imm(SE)
C Move	C.MV	rd, rs2	Move	rd ← rs2
C Branch	C.BEQZ	rs1', imm	Branch Equal Zero	PC ← PC + imm(SE) if rs1' == 0
C Branch	C.BNEZ	rs1', imm	Branch Equal Not Zero	PC ← PC + imm(SE) if rs1' != 0
C Jump	C.J	imm	Jump	PC ← PC + imm(SE)
C Jump	C.JR	rs1	Jump Register	PC ← rs1
C Jump & Link	C.JAL	imm	Jump and Link	x1 ← PC+2, PC ← PC + imm(SE)
C Jump & Link	C.JALR	rs1	Jump Register and Link	x1 ← PC+2, PC ← rs1
C Load	C.LW	rd', rs1', imm	Load Word	rd' ← MW[rs1' + imm(ZE) * 4]
C Load	C.LWSP	rd, imm	Load Word Stack Pointer	rd ← MW[x2 + imm(ZE) * 4]
C Store	C.SW	rs1', rs2', imm	Store Word	MW[rs1' + imm(ZE) * 4] ← rs2'
C Store	C.SWSP	rs2, imm	Store Word Stack Pointer	MW[x2 + imm(ZE) * 4] ← rs2
C System	C.EBREAK		Environment Break	if DCSR.ebreakm == 1, DCSR.cause←1, DPC←PC, Enter to Debug Mode if DCSR.ebreamk == 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 1.90: RV32C 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	0010111
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	000000000000				00000	000	00000	1110011
System	EBREAK	000000000001				00000	000	00000	1110011

Table 1.92: 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 1.93: 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	CSRRWI 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 1.94: 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 1.95: 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 1.96: 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	1000111
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	FSGNJN.S fd, fs1, fs2	0010000			fs2	fs1	001	fd	1010011
F Sign	FSGNJX.S fd, fs1, fs2	0010000			fs2	fs1	010	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 1.97: RV32F Single Precision Floating Point Operation 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:nzuimm=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	11	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	nzuimm[5:4|9:6|2|3]				rd'	00	RES:nzuimm=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	11	rd'	01	rs2'	01	
C Logical	C.OR	rd', rs2'	100	0	11	rd'	10	rs2'	01	
C Logical	C.AND	rd', rs2'	100	0	11	rd'	11	rs2'	01	
C Logical	C.ANDI	rd', imm	100	imm[5]	10	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 1.98: 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 1.99: RV32FC Floating Point Compressed Instruction Code

2 mmRISC-1 Tiny SoC for FPGA

2.1 Overview

“mmRISC_1 Tiny SoC for FPGA” is a sample design of mmRISC-1 application. The overview are shown in Table 2.1. Target FPGA is Intel MAX 10.

Item	Description	Note
CPU Core	mmRISC-1 (1 hart) RV32IMAC or RV32IMAFC	
Debug Interface	RISC-V standard JTAG Debug (Rev.0.13.2)	
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	
MTIME (Timer)	Memory mapped MTIME which generates periodic IRQ_MTIME and software interrupt IRQ_MSOFT	
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	
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)	
RTL	Verilog-2001 / System Verilog	

Table 2.1:mmRISC-1 Tiny SoC for FPGA Overview

2.2 Block Diagram

Block diagram of mmRISC-1 Tiny SoC for FPGA is shown in Figure 2.1. 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. UART is a simple asynchronous Tx/Rx device with configurable baud rate generator. GPIO is general purpose Input Output with 32bits x 3ports. 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.

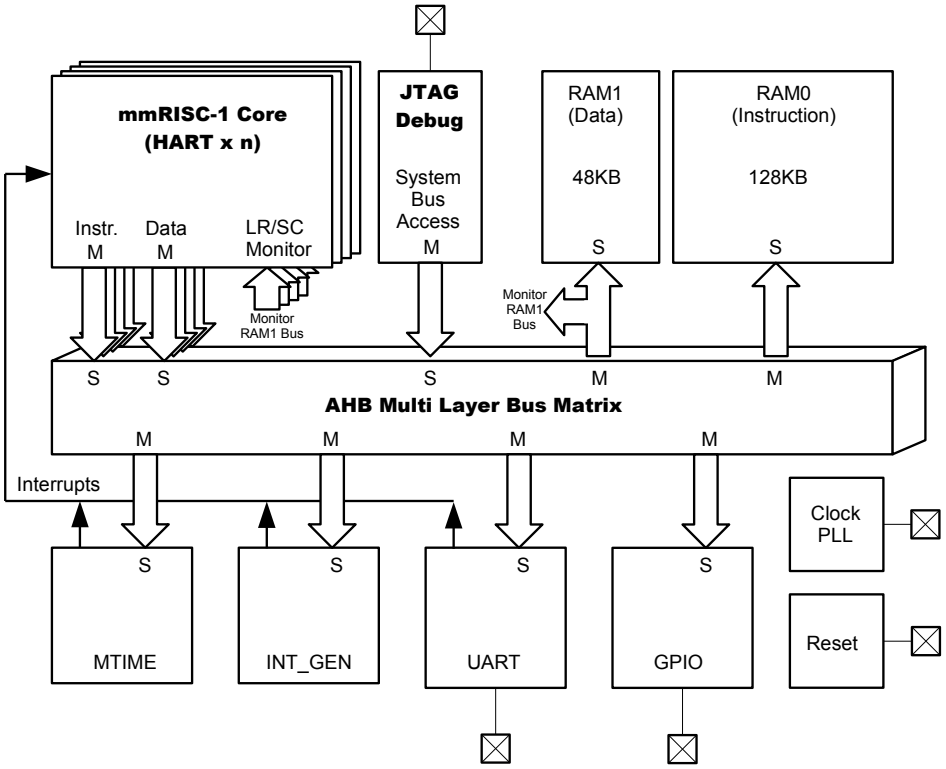

Figure 2.1: Block Diagram of mmRISC-1 Tiny SoC for FPGA

2.3 RTL Files and Structure

RTL files of mmRISC-1 Tiny SoC for FPGA are shown in Table 2.2.

Directory	File Name	Description	Note
verilog/chip	chip_top.v	Top Layer of Chip	
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/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/port	port.v	GPIO	
mmRISC-1 RTLs	Table 1.3	mmRISC-1 Block	

Table 2.2: RTL files of mmRISC-1 Tiny SoC for FPGA

RTL structure of the Tiny SoC is shown in Figure 2.2.

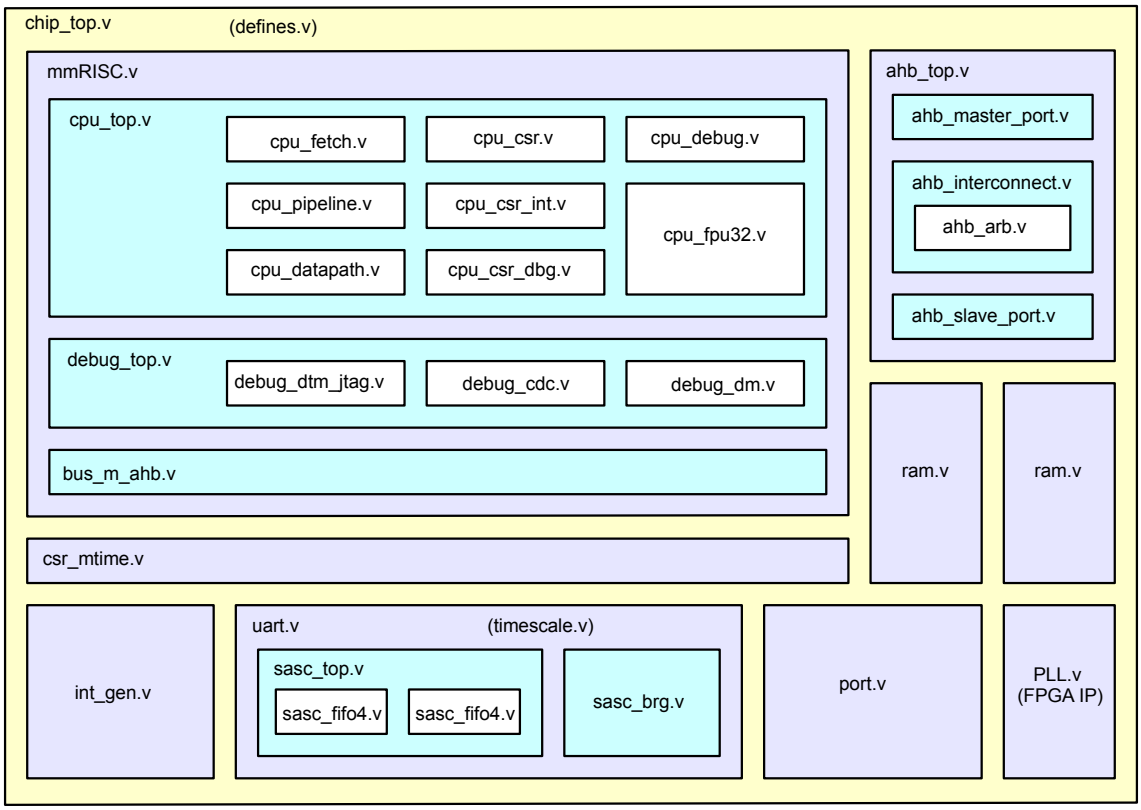

Figure 2.2: RTL Structure of Tiny SoC

2.4 Top Layer of Tiny SoC (FPGA)

Input / Output Signals of Tiny SoC Top Layer 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	inout		SRSTn	System Reset In / Out (Open Drain)	Only for Simulation
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	JTAG Data Output (3-state)	
JTAG	output		RTCK	Return Clock	Only for Simulation
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	

Table 2.3: Input / Output Signals of Tiny SoC Top Layer

2.5 Reset and Clock

Reset structure in the SoC is shown in Figure 2.3. Power on reset circuit is created using counter logic which is initialized at startup by 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 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 outputted from mmRISC block is used as reset of peripheral blocks. Reset structure in mmRISC block is shown in Figure 1.5.

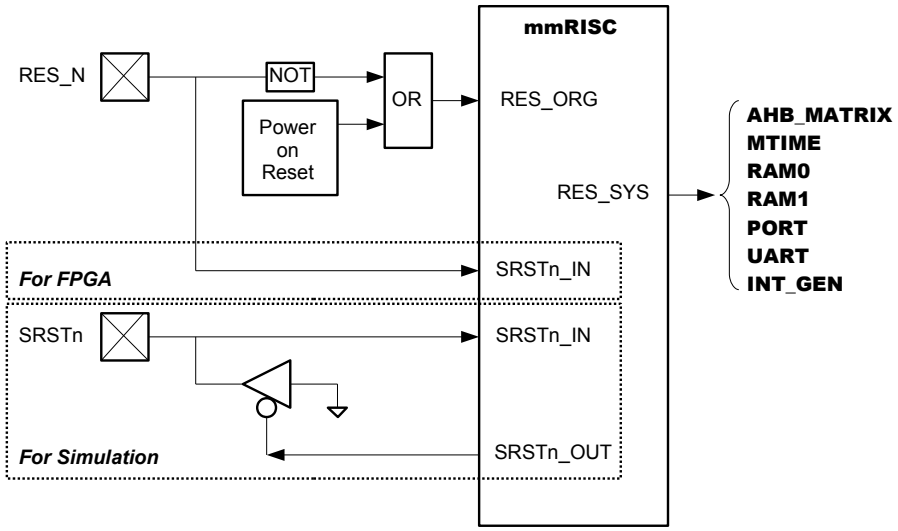

Figure 2.3: Reset Structure in Tiny SoC

Clock structure of SoC is shown in Figure 2.4. For simulation, internal system clock clk is directly connected to external clock input CLK50. For FPGA, clk is generated by PLL, and the frequency is 20MHz if RV32F/RV32FC ISA is not implemented, whereas 16MHz if RV32F/RV32FC ISA is implemented.

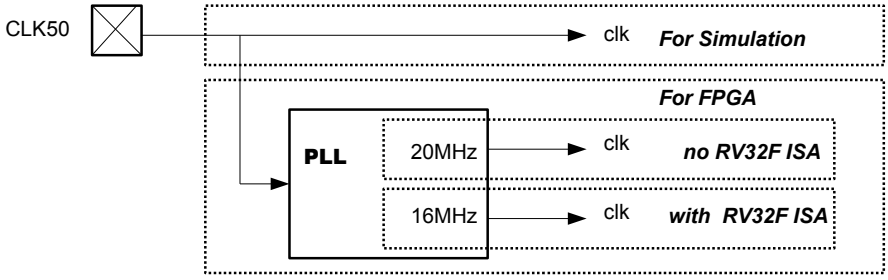

Figure 2.4: Clock Structure in Tiny SoC

2.6 Memory Map

Memory map of the SoC is shown in Table 2.4. All master port of mmRISC block can access all peripheral resources assigned at same addresses.

Address Range	Instruction Bus	Data Bus	System Bus Access	Note
0x00000000-0x48ffffff	reserved	reserved	reserved	
0x49000000-0x4900001f	MTIME	MTIME	MTIME	
0x49000020-0x7fffffff	reserved	reserved	reserved	
0x80000000-0x8fffffff	RAM0 (Data)	RAM0 (Data)	RAM0 (Data)	
0x90000000-0x9fffffff	RAM1 (Instruction)	RAM1 (Instruction)	RAM1 (Instruction)	
0xa0000000-0xafffffff	PORT	PORT	PORT	
0xb0000000-0xbfffffff	UART	UART	UART	
0xc0000000-0xcfffffff	INT_GEN	INT_GEN	INT_GEN	
0xd0000000-0xffffffff	reserved	reserved	reserved	

Table 2.4: Memory Map of Tiny SoC

2.7 Interrupt Assignment

Interrupt assignment in Tiny SoC is shown in Table 2.5. IRQ0 is made by logical OR of interrupt request from UART and interrupt request from INT_GEN.

Interrupt Name	Source Module	Exception Code (MCAUSE)	Vector Address (Vectored Mode)	Note
IRQ_MSOFT	MTIME	0x03	base + 0x0000000c	
IRQ_MTIME	MTIME	0x07	base + 0x0000001c	
IRQ_EXT	INT_GEN	0x0b	base + 0x0000002c	
IRQ0	INT_GEN | UART	0x10	base + 0x00000040	
IRQ2	INT_GEN	0x11	base + 0x00000044	
...	...	...	...	
IRQ63	INT_GEN	0x4f	base + 0x0000013c	

Table 2.5: Interrupt Assignment in Tiny SoC

2.8 Multi-Layer AHB Bus Matrix

(1) 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 any number by module parameters (MASTERS, SLAVES) defined at instantiation of this module in upper layer, respectively. The Bus Matrix supports priority of master access on each slave port from either fixed or round-robin.

(2) 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 array with element counts same as each port counts (MASTERS and SLAVES).

(3) 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 master port counts MASTERS = 8, and M_PRIORITY[0:7] (each element has 3bit width) are specified as follows:

```
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
```

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. Following settings are not guaranteed.

```
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
```

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_HSIZE[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_HPROT[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_HSIZE[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_HPROT[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

(4) 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 follows:

```
S_HADDR_BASE[0] = 32'h01000000
S_HADDR_MASK[0] = 32'hfff00000
```

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.

(5) 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 access to the slave port is stalled until the same master access with HMASTLOCK=0 finished. This feature supports atomic memory access.

(6) 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 the case, output values of access commands related to burst information such as HTRANS (IDLE, BUSY, NOSEQ, SEQ), HBURST and HSIZE on the slave port is completely same as the ones what 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)

(1) 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 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. When logic simulation, the contents of RAM can be initialized by $readmemh from file “rom.memh”.

(2) Input / Output Signals

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

2.10 Peripheral : Interrupt Generator (INT_GEN)

(1) Overview

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

(2) Input / Output Signals

Input / Output signals of INT_GEN are shown in Table 2.8.

(3) Control Registers

Controls registers of INT_GEN are shown in Table 2.9 to Table 2.11.

Group	Direction	Width	Name	Description	Note
Config	Parameter		RAM_SIZE	RAM Size in bytes	
System	input		RES	Reset	
System	input		CLK	System Clocks	
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

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clocks	
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	outout	[63:0]	IRQ	Interrupt Request	

Table 2.8: Input / Output Signals of INT_GEN

Offset Address	Name	Description			
0x0000	IRQ_EXT	External Interrupt Request			
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 2.9: IRQ_EXT

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 2.10: IRQ0

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 2.11: IRQ1

2.11 Peripheral : UART

(1) Overview

UART is a simple asynchronous Tx/Rx device with configurable baud rate generator and 4-depths FIFO buffer for each Tx and Rx. Core logic of the UART is based on SASC (Simple Asynchronous Serial Controller) published on OpenCores site. The UART can generate interrupt request when Tx FIFO buffer has rooms to write (not full) or Rx buffer has at least one data to read (not empty).

(2) Input / Output Signals

Input / Output signals of UART are shown in Table 2.12.

(3) Control Registers

Controls registers of UART are shown in Table 2.14 to Table 2.16.

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clocks	
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
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 2.12: Input / Output Signals of UART

Offset Address	Name	Description			
0x00	UART_TXD UART_RXD	UART TxD Data Register (When Writing) UART RxD Data Register (When Reading)			
Bit	Field	Initial	Access	Description	Note
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 2.13: UART_TXD / UART_RXD

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	Interrput = 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 2.14: UART_CSR

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 2.15: UART_BG0

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 2.16: UART_BG1

2.12 Peripheral : GPIO

(1) Overview

GPIO (module name PORT) has 3 x 32bit input / output ports. Each direction can be configured. When a port bit is configured as input, read value of data register is corresponding input signal level. When a port bit is configured as output, read value of data register is the register value, and write value is stored in the data register and it is output to external pin.

(2) Input / Output Signals

Input / Output signals of GPIO (PORT) are shown in Table 2.17.

Group	Direction	Width	Name	Description	Note
System	input		RES	Reset	
System	input		CLK	System Clocks	
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 2.17: Input / Output Signals of GPIO (PORT)

(3) Control Registers

Controls registers of GPIO (PORT) are shown in Table 2.18 to Table 2.23.

Offset Address	Name	Description			
0x0000	PDR0	Port Data Register 0			
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 2.18: PDR0

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 2.19: PDR1

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 2.20: PDR2

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 2.21: PDD0

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 2.22: PDD1

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 2.23: PDD2

3 Verifications and Implementations

3.1 Development Environment and Tools

Please prepare Development Environment and Tools shown in Table 3.1.

Logic simulation tool ModelSim can be obtained by installing Intel Quartus Prime, but the simulator's executable binary is still 32bit code and does not run on 64bit OS. So please install necessary 32bit libraries according to suggestions on internet describing countermeasures to run the ModelSim on 64bit Linux OS.

Please build and install the Tool Chains and the OpenOCD obtained from each Github.

Please add all executable binary position in environment variable $PATH.

Details of USB-JTAG interface hardware is describe later in this document.

Item	Description	Note
Platform	Ubuntu 64bit	
Simulator	Mentor ModelSim Intel FPGA Started Edition	Included in Quartus 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	
IDE	Eclipse IDE for Embedded C/C++ Developers	
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)	Detail described later.

Table 3.1: Development Environment and Tools

3.2 Github Repository Files

You can get full related files of mmRISC-1 from Github by following command.

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

Major files in the repository are shown in Table 3.2.

Group	Directory	File / Directory Name	Description	Note
Document	mmRISC-1/doc	mmRISC-1_TRM_RevXX.pdf	Technical Reference Manual	
RTL	mmRISC-1/verilog/*	*.v	RTL for mmRISC-1 and Tiny SoC	
Simulation	mmRISC-1/simulation/ modelsim/mmRISC_Simulation	tb_TOP.v	Test Bench of mmRISC-1 Tiny SoC	
		sim_TOP.do	Command Script of ModelSim	
		rom.memh	Initial Data of Instruction RAM generated by hex2v command	
	mmRISC-1/simulation/ modelsim/riscv-arch-test	do_riscv-arch-test.py	Python Script to simulate RISC-V Compliance Test "riscv-arch-test" for I/C/M/Zifence/Privileged	
		flist	File list of RTL codes	
		tb_TOP.v	Test Bench of mmRISC-1 Tiny SoC	
		sim_TOP.do	Command Script of ModelSim	
		./riscv-arch-test/	Directory of RISC-V Architectural Testing Framework	
		./riscv-arch-test/work/ rv32_m/	Directory of Build Binary Files (*.elf) and Expected Signature Output Files (*.signature.output)	
	mmRISC-1/simulation/ modelsim/riscv-tests	do_riscv-tests.py	Python Script to simulate unit tests of RISC-V Processors "riscv-tests" including Atomic and Floating	
		flist	File list of RTL codes	
		tb_TOP.v	Test Bench of mmRISC-1 Tiny SoC	
		sim_TOP.do	Command Script of ModelSim	
		./riscv-tests/work/isa/RV32*/	Directory of Build Binary Files (*)	
Tools	mmRISC-1/tools	hex2v.c	Converter from Intel Hex format to Initial RAM data for verilog $readmemh()	
Sample Programs	mmRISC-1/workspace/ mmRISC_SampleCPU	./src/*.c, *.h, *.S	C Source Files, C Header Files and Startup Assembler Source File for Sample CPU Application	
		link.ld	Linker Script	
		.cproject, .projct	Eclipse Project Files	
	mmRISC-1/workspace/ mmRISC_SampleFPU	./src/*.c, *.h, *.S	C Source Files, C Header Files and Startup Assembler Source File for Sample FPU Application	
		link.ld	Linker Script	
		.cproject, .projct	Eclipse Project Files	
	mmRISC-1/workspace/ mmRISC_FreeRTOS	./src_app/inc/*.h ./src_app/src/*.c, *.S	C Source Files, C Header Files and Startup Assembler Source File for FreeRTOS Blinky Application	
		./src_kernel/inc/*.h ./src_kernel/src/*.c, *.S	C Source Files, C Header Files and Startup Assembler Source File for FreeRTOS Kernel	
		link.ld	Linker Script	
		.cproject, .projct	Eclipse Project Files	
	mmRISC-1/workspace/ mmRISC_Dhrystone	./src/*.c, *.h, *.S	C Source Files, C Header Files and Startup Assembler Source File for Dhrystone Benchmark	
		link.ld	Linker Script	
		.cproject, .projct	Eclipse Project Files	
	mmRISC-1/workspace/ mmRISC_Coremark	./src/*.c, *.h, *.S	C Source Files, C Header Files and Startup Assembler Source File for Coremark Benchmark	
		link.ld	Linker Script	
		.cproject, .projct	Eclipse Project Files	
OpenOCD	mmRISC-1/openocd	openocd.cfg	OpenOCD Configuration File	
FPGA	mmRISC-1/fpga	mmRISC.pqf	Quartus Prime Project File	
		PLL.*	PLL IP Files	
		mmRISC.sdc	Design Constraints File	
		./output_files	Directory for Generated Files	

Table 3.2: Major Files in Github Repository

3.3 RTL Verification Methods

(1) Vector Simulation

A directory “mmRISC-1/simulation/modelsim/mmRISC_Simulation” contains a sample set of vector simulation environment.

a) Common Preparation (Do Once)

Before your try simulation, please build “tools/hex2v.c” once by following command, and add position of generated executable binary “hex2v” in the environment variable $PATH.

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

b) Preparation of Script Command for ModelSim

Please edit command script file “sim_TOP.do” if necessary. Default script instructs following operations.

- To Convert Intel Hex binary program code “*.hex” to “rom.memh” which initializes instruction RAM by $readmemh() in RTL code, by the command “hex2v” prepared above. Please edit binary code file name if you want to change the codes to be stored in instruction RAM.

- To Compile RTL files for verification by the command “vlog”. Please edit RTL file list if you change RTL file structure.

- To Launch Simulator by the command “vsim”.

- To Add Signals to display in Wave Window by ModelSim internal command “add wave”. Please edit the command list if you want to change the waveform display.

- To Start Logic Simulation by ModeSim internal command “run”.

c) Preparation of Test Bench RTL

Please edit test bench RTL code “tb_TOP.v” if necessary. In the test bench, followings are supported.

- Generation of Reset and Clock

- Instantiation of CHIP_TOP (chip_top.v)

- Simulation Time Out (`TB_STOP)

- Initialization of nodes in Power-On-Reset circuit.

- Simulation Stop Condition (Both the end of Stimulus and writing Memory; address = 0xfffffffc, wdata = 0xdeaddead)

- Dump Instruction Decode Sequence to “dump.txt” (if `DUMP_ID_STAGE is defined)

- Tasks for JTAG Access (JTAG Reset, Shift-In/Out IR/DR, DMI Write and Read)

- Stimulus (if `JTAG_OPERATION is defined) : Read IDCODE, JTAG Reset, Debugger Command for Stop/Step/Run)

- If you want to insert wait cycle in Instruction RAM access and Data RAM access, define "`RAM_WAIT" in "define.v". The wait cycles are determined by bit [3:2] of 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.v". Each hart has different reset vector, but all harts fetch instruction codes from same Instruction RAM due to shadowed addressing.

d) Run the Logic Simulation

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

e) Check Waveforms and Dump File (if any)

Hopefully you will get success ! ModelSim Screen Shot is shown in Figure 3.1.

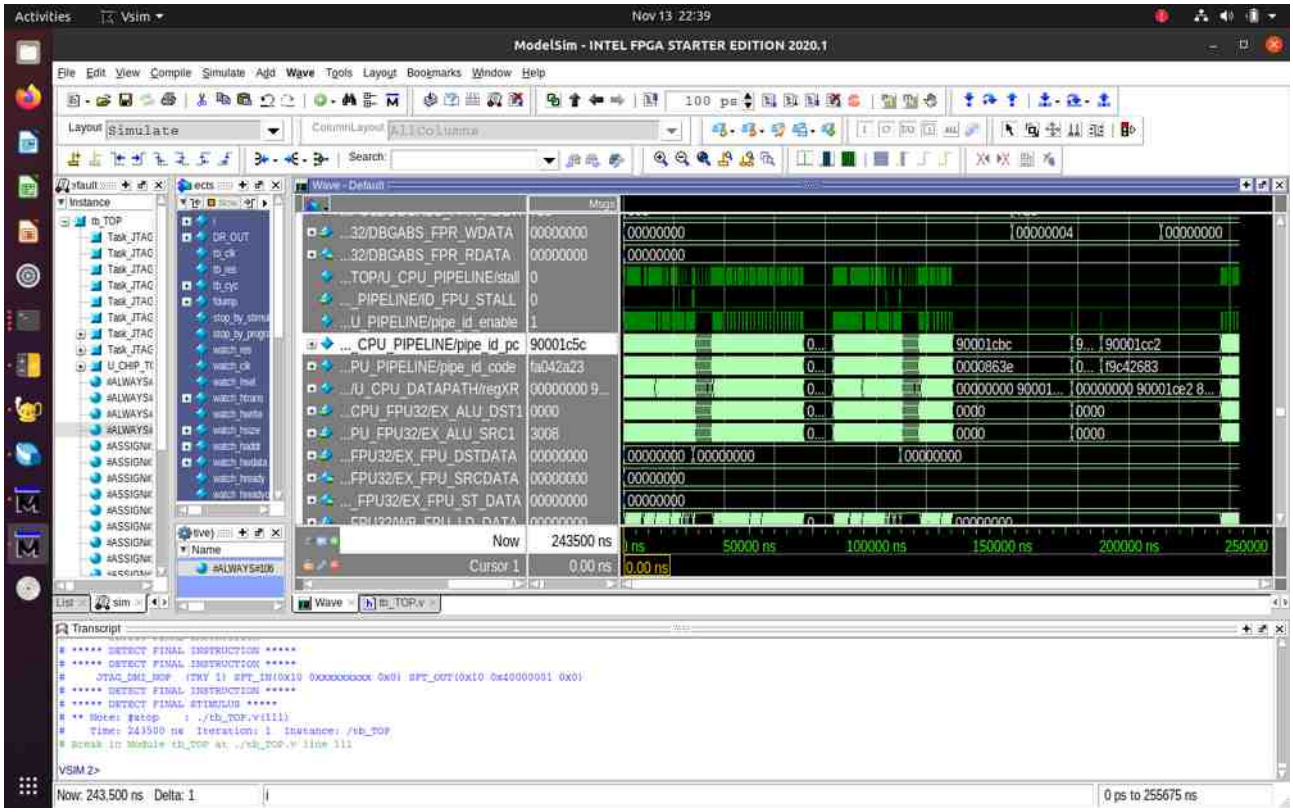

Figure 3.1: ModelSim Screen

(2) RISC-V Compliance Test “riscv-arch-test” for I/C/M/Zifence/Privileged

A directory “mmRISC-1/simulation/modelsim/riscv_arch_test” contains a sample set of Compliance Test “riscv-arch-test” for I/C/M/Zifence/Privileged instructions

a) Preparation of “riscv_arch_test”

The mmRISC-1 repository includes completed test binaries and signatures. But if you want to prepare by yourself, please prepare RISC-V ISA Simulator “Spike” in advance. Then please get “riscv-arch-test” from Github shown in Table 3.1. Following its documentation, run makefile to generate elf-binaries with dis-assemble list and simulate to generate expecting signatures. Let all elf-binaries, dis-assemble list and signatures place under directory “riscv-arch-test/work/rv32i_m/*” with classifying in each ISA group. Finally move the directory “riscv-arch-test” under the directory “mmRISC-1/simulation/modelsim/riscv_arch_test”.

b) Running Simulations

To run the test automatically, python script “do_riscv_arch_test.py” is prepared. It simulates mmRISC-1 Tiny SoC under four conditions that are; (1) No RAM Wait Cycles + 1 Hart, (2) No RAM Wait Cycles + 4 Harts, (3) With RAM Wait Cycles + 1 Hart, (4) With RAM Wait Cycles + 4 Harts. Instruction sequence Binary (elf) to be simulated is taken from “riscv-arch-test/riscv-arch-test/work/rv32i_m/*/*.elf”. Each program code stores signature data to address from <begin_signature> to <end_signature>, and stores 0x00000001 to address <tohost> when reaching at end of program. These <*> parameters can be obtained from corresponding dis-assemble list “*.objdump” of each binary code. Once the simulation finished, test bench RTL outputs stored data from <begin_signature> to <end_signature> as dump list file “dump.signature”. Then the python script compares the dump and prepared expecting signatures. If matched, the python script continues to simulate next program code. If unmatched, the python script stops. You will see reports as shown in List 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
...
```

List 3.1: Output of “riscv-arch-test”

(3) RISC-V Unit Test “riscv-tests” including Atomic and Floating Point ISA

A directory “mmRISC-1/simulation/modelsim/riscv_tests” contains a sample set of Unit Test “riscv-tests” including Atomic and Floating Point ISA.

a) Preparation of “riscv_tests”

The mmRISC-1 repository includes completed test binaries. But if you want to prepare by yourself, please get “riscv-tests” from Github shown in Table 3.1. Following its documentation, run makefile to generate elf-binaries and dis-assemble list. Let all elf-binaries and dis-assemble list place under directory “riscv-tests/work/isa/*” with classifying in each ISA group. Finally move the directory “riscv-test” under the directory “mmRISC-1/simulation/modelsim/riscv_tests”.

b) Running Simulations

To run the test automatically, python script “do_riscv_tests.py” is prepared. It simulates mmRISC-1 Tiny SoC under four conditions that are; (1) No RAM Wait Cycles + 1 Hart, (2) No RAM Wait Cycles + 4 Harts, (3) With RAM Wait Cycles + 1 Hart, (4) With RAM Wait Cycles + 4 Harts. Instruction sequence Binary (elf) to be simulated is taken from “riscv-arch-test/riscv-arch-test/work/isa/RV32*/*”. In the script file, if you verify RV32IMC, specify RV32IMC directory, if you verify RV32A, specify RV32IMAC directory, and if you verify RV32F and RV32FC, specify RV32IMAFC directory. Each program code stores 0x00000001 to address <tohost> when reaching at success point of program, or stores other data to address <tohost> if failed. The <tohost> parameter can be obtained from corresponding dis-assemble list “*.dump” of each binary code. Once the simulation finished, the test bench RTL outputs “result.txt” which contents is “PASS” or “FAIL” text. Then the python script check the result text, and if PASS, the python script continues to simulate next program code, or if FAIL, the python script stops. You will see report as shown in List 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
...
```

List 3.2: Output of "riscv-tests"

3.4 FPGA Implementation

Target FPGA is Intel MAX 10 10M50DAF484C7G and the target board is Terasic DE10-Lite Board (Figure 3.2). FPGA Pin Assignment is shown in Table 3.4.

FPGA related resources are stored in directory “mmRISC-1/fpga”. From Quartus Prime (Figure 3.3), open project file “mmRISC.qpf”. Timing constraints file is prepared as “mmRISC.sdc”. Compile the project. Configuration file “mmRISC.pof” will be generateed in directory “output_files”. Program the FPGA via USB Blaster Interface.

If Floating Point ISA is not implemented, the FPGA operates in 20MHz (Tcyc = 50ns), wheres if implemented, in 16.67MHz (Tcyc = 60ns). In CHIP_TOP layer, if “`RISCV_ISA_RV32F” is defined, 16MHz output from PLL is used as system clock, wheres if not, 20MHz output from PLL is used. FPGA Configuration results are shown in Table 3.3.

RISC-V Spec	RV32IMAC	RV32IMAFC
Peripherals	Instruction RAM 128KB Data RAM 48KB Multilayer BUS UART Timer GPIO	Instruction RAM 128KB Data RAM 48KB Multilayer BUS UART Timer GPIO
Device	MAX 10 10M50DAF484C7G	MAX 10 10M50DAF484C7G
Quartus Prime	Lite Edition Ver.20.1.1	Lite Edition Ver.20.1.1
Optimization	Balanced (Normal Flow)	Balanced (Normal Flow)
Logic Elements	23061/49760 = 46%	41662/49760 = 84%
Interconnect Usage	29%	39%
Frequency	20MHz	16.67MHz
Worst Setup Slack	3.344ns (met)	0.199ns (met)

Table 3.3: FPGA Configuration Results

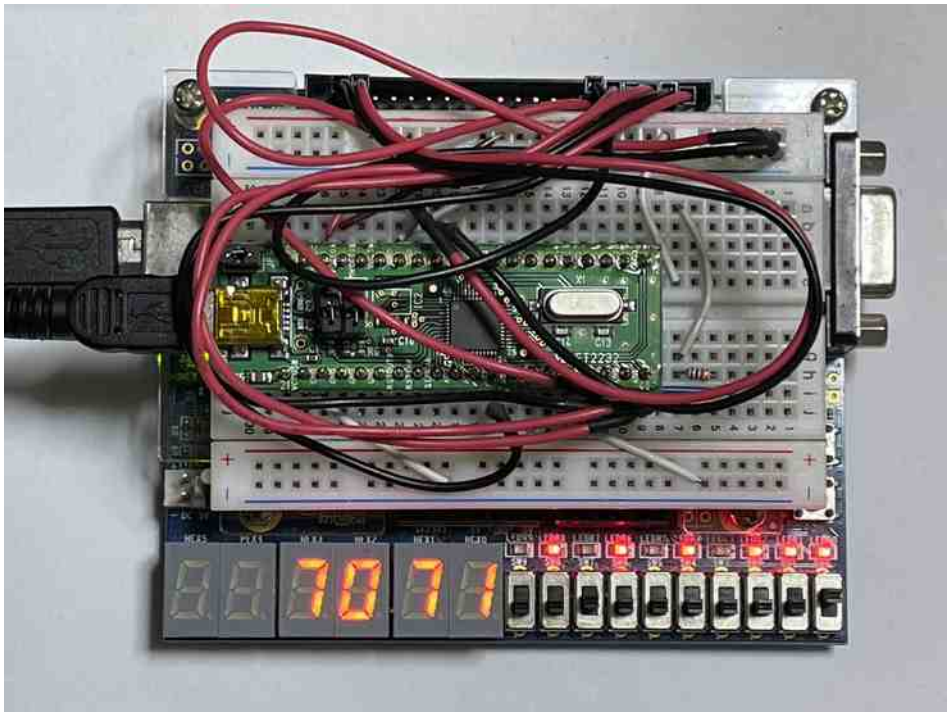

Figure 3.2: DE10-Lite Board (Debugger Interface is on the board)

Group	Signal Name	FPGA	DE10 Lite Board
Reset Push Switch	RES_N	B8	KEY0
JTAG	TRSTn	Y5	Pin Header GPIO_29
JTAG	TCK	Y6	Pin Header GPIO_27
JTAG	TMS	AA2	Pin Header GPIO_35
JTAG	TDI	Y4	Pin Header GPIO_31
JTAG	TDO	Y3	Pin Header GPIO_33
7-Segment LED0	GPIO0[0]	C14	HEX00 segment A
7-Segment LED0	GPIO0[1]	E15	HEX01 segment B
7-Segment LED0	GPIO0[2]	C15	HEX02 segment C
7-Segment LED0	GPIO0[3]	C16	HEX03 segment D
7-Segment LED0	GPIO0[4]	E16	HEX04 segment E
7-Segment LED0	GPIO0[5]	D17	HEX05 segment F
7-Segment LED0	GPIO0[6]	C17	HEX06 segment G
7-Segment LED0	GPIO0[7]	D15	HEX07 segment DP
7-Segment LED1	GPIO0[8]	C18	HEX10 segment A
7-Segment LED1	GPIO0[9]	D18	HEX11 segment B
7-Segment LED1	GPIO0[10]	E18	HEX12 segment C
7-Segment LED1	GPIO0[11]	B16	HEX13 segment D
7-Segment LED1	GPIO0[12]	A17	HEX14 segment E
7-Segment LED1	GPIO0[13]	A18	HEX15 segment F
7-Segment LED1	GPIO0[14]	B17	HEX16 segment G
7-Segment LED1	GPIO0[15]	A16	HEX17 segment DP
7-Segment LED2	GPIO0[16]	B20	HEX20 segment A
7-Segment LED2	GPIO0[17]	A20	HEX21 segment B
7-Segment LED2	GPIO0[18]	B19	HEX22 segment C
7-Segment LED2	GPIO0[19]	A21	HEX23 segment D
7-Segment LED2	GPIO0[20]	B21	HEX24 segment E
7-Segment LED2	GPIO0[21]	C22	HEX25 segment F
7-Segment LED2	GPIO0[22]	B22	HEX26 segment G
7-Segment LED2	GPIO0[23]	A19	HEX27 segment DP
7-Segment LED3	GPIO0[24]	F21	HEX30 segment A
7-Segment LED3	GPIO0[25]	E22	HEX31 segment B
7-Segment LED3	GPIO0[26]	E21	HEX32 segment C
7-Segment LED3	GPIO0[27]	C19	HEX33 segment D
7-Segment LED3	GPIO0[28]	C20	HEX34 segment E
7-Segment LED3	GPIO0[29]	D19	HEX35 segment F
7-Segment LED3	GPIO0[30]	E17	HEX36 segment G
7-Segment LED3	GPIO0[31]	D22	HEX37 segment DP
7-Segment LED4	GPIO1[0]	F18	HEX40 segment A
7-Segment LED4	GPIO1[1]	E20	HEX41 segment B
7-Segment LED4	GPIO1[2]	E19	HEX42 segment C
7-Segment LED4	GPIO1[3]	J18	HEX43 segment D
7-Segment LED4	GPIO1[4]	H19	HEX44 segment E
7-Segment LED4	GPIO1[5]	F19	HEX45 segment F
7-Segment LED4	GPIO1[6]	F20	HEX46 segment G
7-Segment LED4	GPIO1[7]	F17	HEX47 segment DP
7-Segment LED5	GPIO1[8]	J20	HEX50 segment A
7-Segment LED5	GPIO1[9]	K20	HEX51 segment B
7-Segment LED5	GPIO1[10]	L18	HEX52 segment C
7-Segment LED5	GPIO1[11]	N18	HEX53 segment D
7-Segment LED5	GPIO1[12]	M20	HEX54 segment E
7-Segment LED5	GPIO1[13]	N19	HEX55 segment F
7-Segment LED5	GPIO1[14]	N20	HEX56 segment G
7-Segment LED5	GPIO1[15]	L19	HEX57 segment DP

Group	Signal Name	FPGA	DE10 Lite Board
Clock	CLK50	P11	50MHz Clock
SDRAM	SDRAM_CLK	L14	Fixed to Low to disable SDRAM
SDRAM	SDRAM_CKE	N22	Fixed to Low to disable SDRAM
SDRAM	SDRAM_CSn	U20	Fixed to Low to disable SDRAM
UART	TXD	W10	Pin Header GPIO_1
UART	RXD	W9	Pin Header GPIO_3
LED	GPIO1[16]	A8	LEDR0
LED	GPIO1[17]	A9	LEDR1
LED	GPIO1[18]	A10	LEDR2
LED	GPIO1[19]	B10	LEDR3
LED	GPIO1[20]	D13	LEDR4
LED	GPIO1[21]	C13	LEDR5
LED	GPIO1[22]	E14	LEDR6
LED	GPIO1[23]	D14	LEDR7
LED	GPIO1[24]	A11	LEDR8
LED	GPIO1[25]	B11	LEDR9
General Purpose	GPIO1[26]	V10	Pin Header GPIO_0
General Purpose	GPIO1[27]	V9	Pin Header GPIO_2
General Purpose	GPIO1[28]	V8	Pin Header GPIO_4
General Purpose	GPIO1[29]	W8	Pin Header GPIO_5
General Purpose	GPIO1[30]	V7	Pin Header GPIO_6
General Purpose	GPIO1[31]	W7	Pin Header GPIO_7
Slide Switch	GPIO2[0]	C10	SW0
Slide Switch	GPIO2[1]	C11	SW1
Slide Switch	GPIO2[2]	D12	SW2
Slide Switch	GPIO2[3]	C12	SW3
Slide Switch	GPIO2[4]	A12	SW4
Slide Switch	GPIO2[5]	B12	SW5
Slide Switch	GPIO2[6]	A13	SW6
Slide Switch	GPIO2[7]	A14	SW7
Slide Switch	GPIO2[8]	B14	SW8
Slide Switch	GPIO2[9]	F15	SW9
Push Switch	GPIO2[10]	A7	KEY1
General Purpose	GPIO2[11]	W6	Pin Header GPIO_8
General Purpose	GPIO2[12]	V5	Pin Header GPIO_9
General Purpose	GPIO2[13]	W5	Pin Header GPIO_10
General Purpose	GPIO2[14]	AA15	Pin Header GPIO_11
General Purpose	GPIO2[15]	AA14	Pin Header GPIO_12
General Purpose	GPIO2[16]	W13	Pin Header GPIO_13
General Purpose	GPIO2[17]	W12	Pin Header GPIO_14
General Purpose	GPIO2[18]	AB13	Pin Header GPIO_15
General Purpose	GPIO2[19]	AB12	Pin Header GPIO_16
General Purpose	GPIO2[20]	Y11	Pin Header GPIO_17
General Purpose	GPIO2[21]	AB11	Pin Header GPIO_18
General Purpose	GPIO2[22]	W11	Pin Header GPIO_19
General Purpose	GPIO2[23]	AB10	Pin Header GPIO_20
General Purpose	GPIO2[24]	AA10	Pin Header GPIO_21
General Purpose	GPIO2[25]	AA9	Pin Header GPIO_22
General Purpose	GPIO2[26]	Y8	Pin Header GPIO_23
General Purpose	GPIO2[27]	AA8	Pin Header GPIO_24
General Purpose	GPIO2[28]	Y7	Pin Header GPIO_25
General Purpose	GPIO2[29]	AA7	Pin Header GPIO_26
General Purpose	GPIO2[30]	AA6	Pin Header GPIO_28
General Purpose	GPIO2[31]	AA5	Pin Header GPIO_30

Table 3.4: FPGA Pin Assignment (DE10-Lite Board)

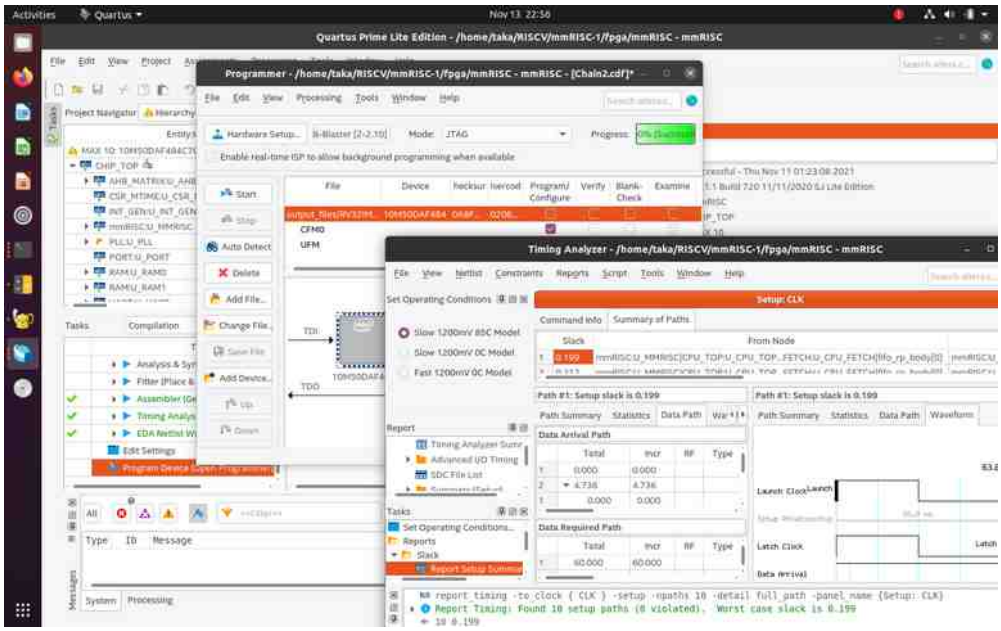

Figure 3.3: Intel Quartus Prime Lite Edition

3.5 JTAG Debugger Interface for OpenOCD

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.

To make the interface hardware, commercial breakout board of FT2232D is convenient. You can get the board from https://akizukidenshi.com/catalog/g/gM-02990/. Connect between the board and FPGA as shown in Figure 3.4. If you use above breakout board, please confirm that jumper pin JP1=short, JP2A=open and JP2B=open.

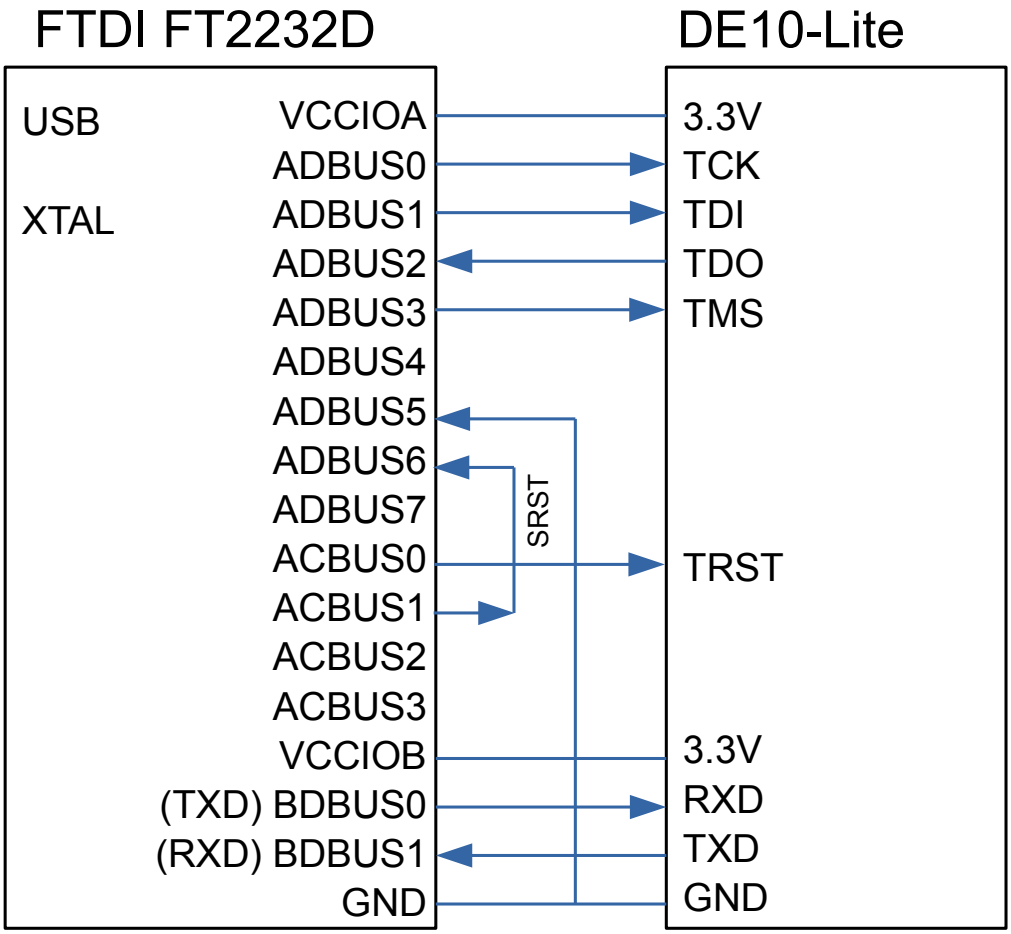

Figure 3.4: Connection of FT2232D and FPGA for JTAG Interface

3.6 Sample Programs

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

(1) Integer Instruction Program “mmRISC_SampleCPU”

First please build the program, and launch a terminal application on Ubuntu such as minicom. Communication parameters are 9600 bps and 8N1 (8bit, Non-Parity and 1 Stop bit). Before run and debug the program, change Eclipse perspective to debug, and select menu of “Run”→”Debug Configuration”. Create debug configuration “mmRISC_SampleCPU” in GDB OPenOCD Debugging. Please fill “./Debug/mmRISC_SampleFPU.elf” in C/C++ Application field. And push “Debug” button. The program breaks at beginning of main(). Push resume button (green right arrow) to restart the program. You can use several debug operations in Eclipse windows (Figure 3.5).

In this program, periodic interrupt is requested from MTIME timer to increment binary number displayed on 10 discrete LEDs, and the incrementing speed and direction can be set by slide switch (SW0-SW9) and push switch (KEY1). Note that if SW0-SW9 are all off, then the LED increment stops. Moreover, after some messages are displayed in terminal by printf() function, please type any characters in terminal. When each ASCII code is received by UART, the RX interrupt is requested and its software handler displays the code on 7-segment LED. The operating frequency of FPGA is different whether floating point ISA is implemented or not, therefore the program recognizes system clock frequency by reading CSR MISA to check “F” instruction exists or not.

(2) Floating Point Instruction Program “mmRISC_SampleFPU”

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

In this program, 10,000 times as sine function is calculated and the results are displayed on 7-segment LED as decimal number. During this operation, periodic MTIME interrupt increment 10 discrete LEDs like as sample program “ mmRISC_SampleCPU”.

(3) FreeRTOS Porting “mmRISC_FreeRTOS”

This program demonstrates FreeRTOS porting on mmRISC-1. Typical demo application of FreeRTOS “Blinky” is implemented as main program. Application sources are gathered in directory “src_app” and RTOS kernel sources are in directory “src_kernel”. Before running this demo, launch terminal window.

(4) Dhrystone benchmark “mmRISC_Dhrystone”

This is the Dhrystone benchmark program ported to mmRISC-1. Before running this benchmark, launch terminal window. You will see benchmark result. Calculated scores are shown in Table 3.5.

(5) Coremark benchmark “mmRISC_Coremark”

This is the Coremark benchmark program ported to mmRISC-1. Before running this benchmark, launch terminal window. You will see benchmark result. Calculated scores are shown in Table 3.5.

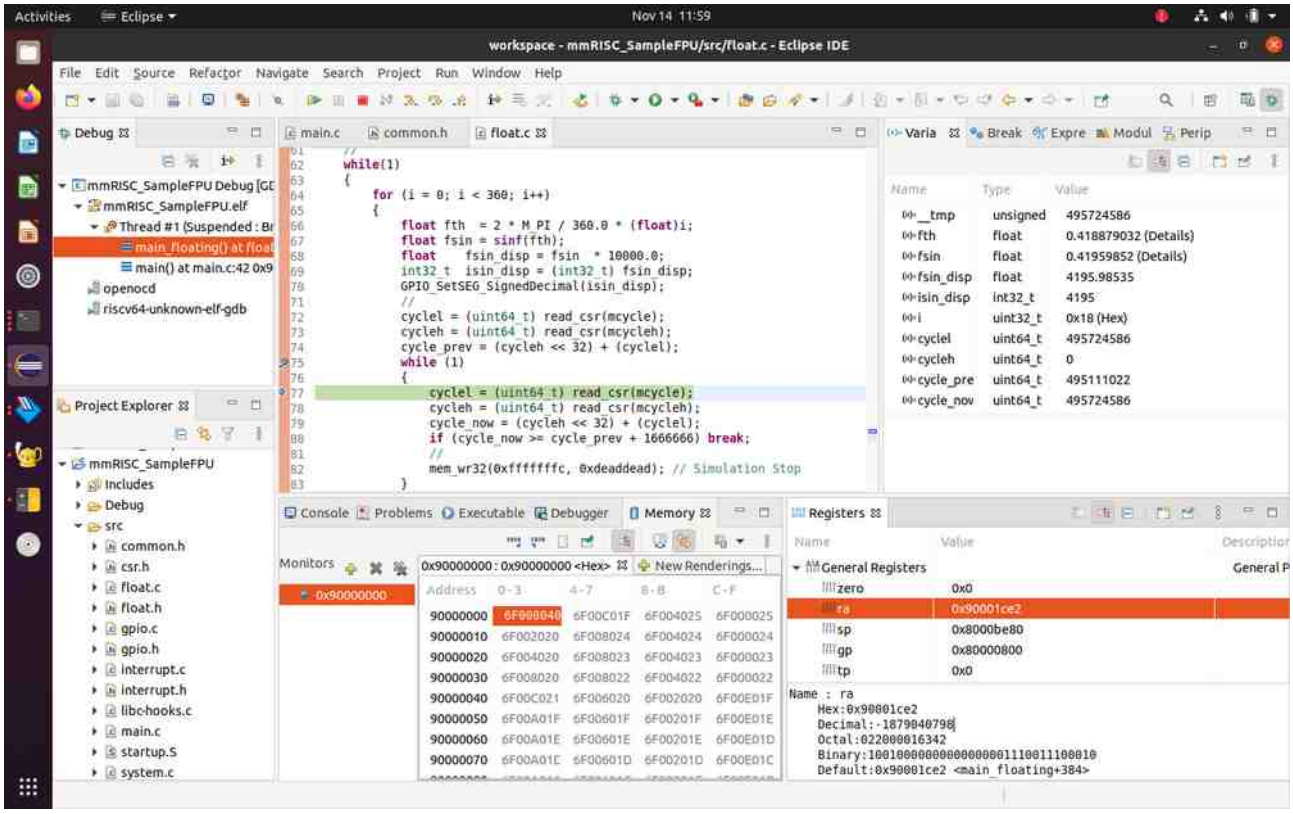

Figure 3.5: Eclipse Window

Benchmark	Compiler	Compiler Option	Result	Memory
Dhrystone	riscv64-unknown-elf-gcc 10.2.0	-O3	1.55 DMIPS/MHz	1cyc Read, 1cyc Write RAM
	riscv64-unknown-elf-gcc 10.2.0	-O3 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -flto	2.26 DMIPS/MHz	1cyc Read, 1cyc Write RAM
Coremark	riscv64-unknown-elf-gcc 10.2.0	-O3 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las	2.76 Coremark/MHz	1cyc Read, 1cyc Write RAM

Table 3.5: Scores of Benchmarks of mmRISC-1 core

4 Referenced Documents

4.1 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] Simple Asynchronous Serial Controller (SASC)

https://opencores.org/projects/sasc

[9] ModelSim® Starter Edition Command Reference Manual, Software Version 10.4b , 2015, Mentor Graphics Corporation