HB0801

MIV_RV32IMAF_L1_AHB v2.1 Handbook

Microsemi
a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 2.0

The following is a summary of the changes made in revision 2.0 of this document.

- Revision 2.0 is the updated MIV_RV32IMAF_L1_AHB document, which adds ECC support for PolarFire and RTG4.
- Added Chapter 5 Register Map and Descriptions and updated to include more information on the address space within the core.
- Added Chapter 6 Interrupts which covers the Interrupts supported by this core and their address space.
- Added Chapter 10 SoftConsole and updated with the compiler flags that need to be set in SoftConsole for the MIV_RV32IMAF_L1_AHB core.

1.2 Release 1.0

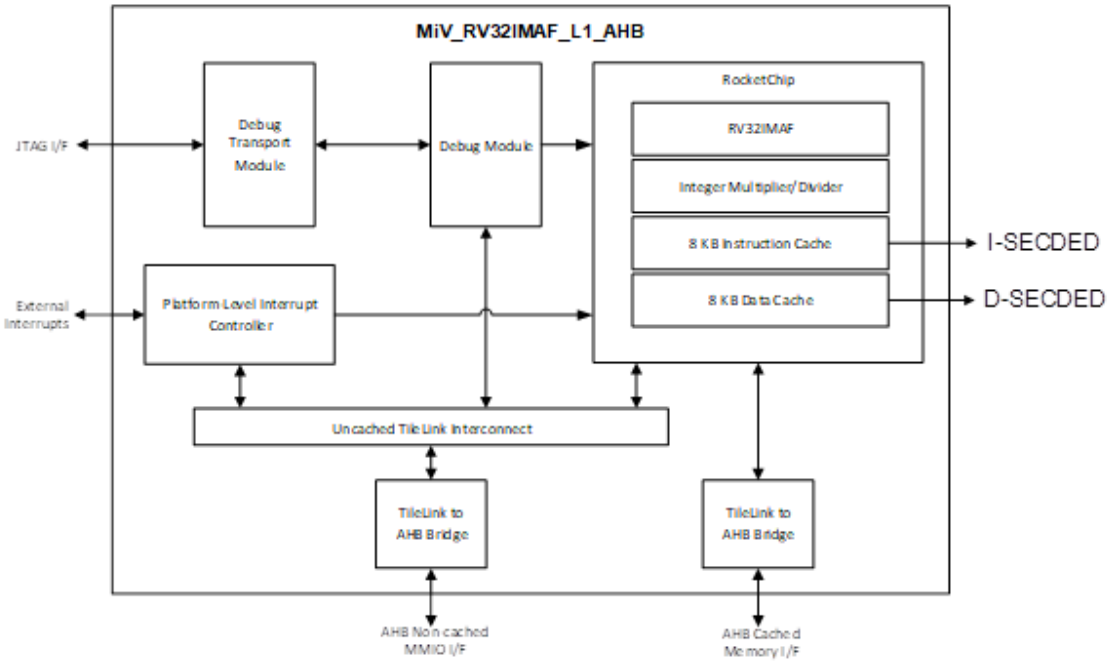
Revision 1.0 was the first publication of this document. Created for MIV_RV32IMAF_L1_AHB v2.0.

2 Introduction

2.1 Overview

The MIV_RV32IMAF_L1_AHB is a softcore processor designed to implement the RISC-V instruction set for use in Microsemi FPGAs. The processor is based on Rocket-Chip, which contains a high-performance single-issue in order execution pipeline 32-bit RISC-V core. This core includes, an industry-standard JTAG interface to facilitate debug access, along with separate AHB bus interfaces for memory and IO access, Error-Correcting Code (ECC) cache memory availability and support for 31 dedicated interrupt ports.

Figure 1 • **MIV_RV32IMAF_L1_AHB Block Diagram**



2.2 Features

- Designed for low power ASIC microcontroller and FPGA soft-core implementations.
- Integrated 8Kbytes instructions cache and 8Kbytes data cache.
- A Platform-Level Interrupt Controller (PLIC) supports up to 31 programmable interrupts with a single priority level 0. The 31 interrupt inputs are serviced from 0 to 31 in ascending order.
- Supports the RISC-V standard RV32IMAF ISA.
- On-Chip debug unit with a JTAG interface.
- Two external AHB interfaces for IO and memory.
- Support for Error-Correcting Code (ECC) cache on RTG4 and PolarFire.

2.3 Core Version

This Handbook applies to MIV_RV32IMAF_L1_AHB version 2.1.

Note: The accompanying manuals for this core are:

- The RISC-V Instruction Set Manual, Volume 1, User Level ISA, Version 2.2
- The RISC-V Instruction Set Manual, Volume 2, Privileged Architecture, Version 1.10
- The RISC-V External Debug Support, Version 0.13

2.4 Supported Families

- PolarFire®
- RTG4™
- IGLOO®2
- SmartFusion®2

2.5 Device Utilization and Performance

Utilization and performance data is listed in Table 1 for the supported device families. The data listed in this table is indicative only. The overall device utilization and performance of the core is system dependent.

Table 1 • **Device Utilization and Performance**

Family	Sequential	Combinatorial	uSRAM	LSRAM	Math	Frequency (MHz)	Min Timing	Max Timing
SmartFusion2	9033	20646	8	8	6	61.00	Pass	Pass
IGLOO2	9033	20653	8	8	6	63.00	Pass	Pass
RTG4	9072	21113	2	14	6	59.00	Pass	Pass
RTG4 (ECC)	10143	23251	0	14	0	53.00	Pass	Pass
PolarFire	9138	21013	6	22	6	105.00	Pass	Pass
PolarFire (ECC)	10159	23310	0	22	0	105.00	Pass	Pass

Note:

1. Performance numbers are for standard speed devices, except for PolarFire, which uses the -1 speed.
2. Multi-pass place-and-route setting is set to 5.

3 Functional Description

3.1 MIV_RV32IMAF_L1_AHB Architecture

Figure 2 • **MIV_RV32IMAF_L1_AHB Block Diagram**

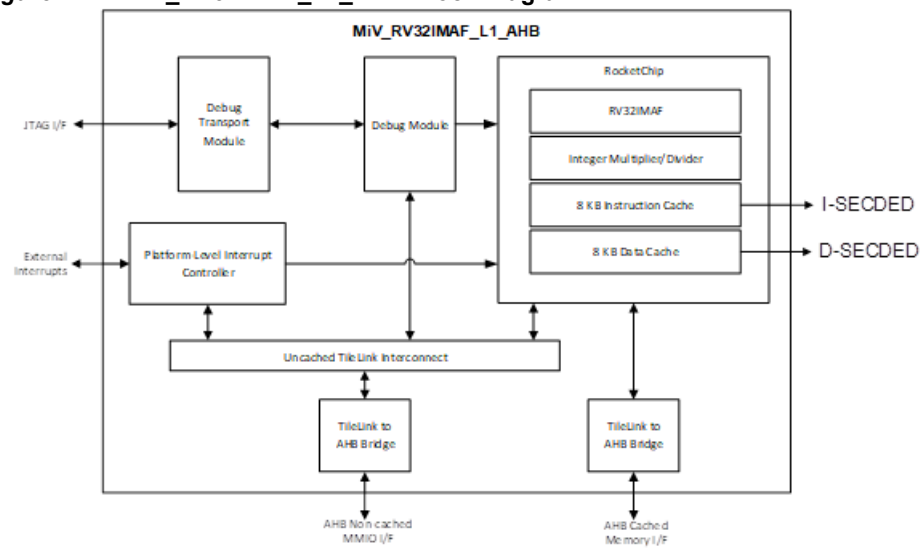


Table 2 • **MIV_RV32IMAF_L1_AHB Architecture**

Parameter	Value	Units	Notes
ISA Support	RV32IMAF		
Cores	1		
Harts/Cores	1		
Branch Prediction	None		Static Not Taken
Multiplier occupancy	16	cycles	2-bit/cycles iterative multiply
I-cache size	8	KiB	
I-cache associativity	1	way	direct-mapped
I-cache line-size	64	bytes	
D-cache size	8	KiB	
D-cache associativity	1	way	direct-mapped
D-cache line-size	64	bytes	
Reset Vector	configurable		
External interrupts	31		
PLIC Interrupt priorities	1		Fixed priorities
External memory bus	AHB		
External I/O bus	AHB		
JTAG debug transport address width	5	bits	
Hardware breakpoints	2		

3.2 MIV_RV32IMAF_L1_AHB Processor Core

MIV_RV32IMAF_L1_AHB is based on Rocket-Chip. The core provides a single hardware thread (or hart) supporting the RISC-V standard RV32IMAF ISA and machine-mode privileged architecture.

3.3 Pipelined Architecture

MIV_RV32IMAF_L1_AHB provides a high-performance single-issue in order for 32-bit execution pipeline, with a peak sustainable execution rate of one instruction per clock cycle. The RISC-V ISA standard M extensions add hardware multiply and divide instructions. MIV_RV32IMAF_L1_AHB has a range of performance options with a fully pipelined multiply unit. Following is an example of the pipeline and its timing:

Figure 3 • **Example Five Stage Pipelined Architecture**

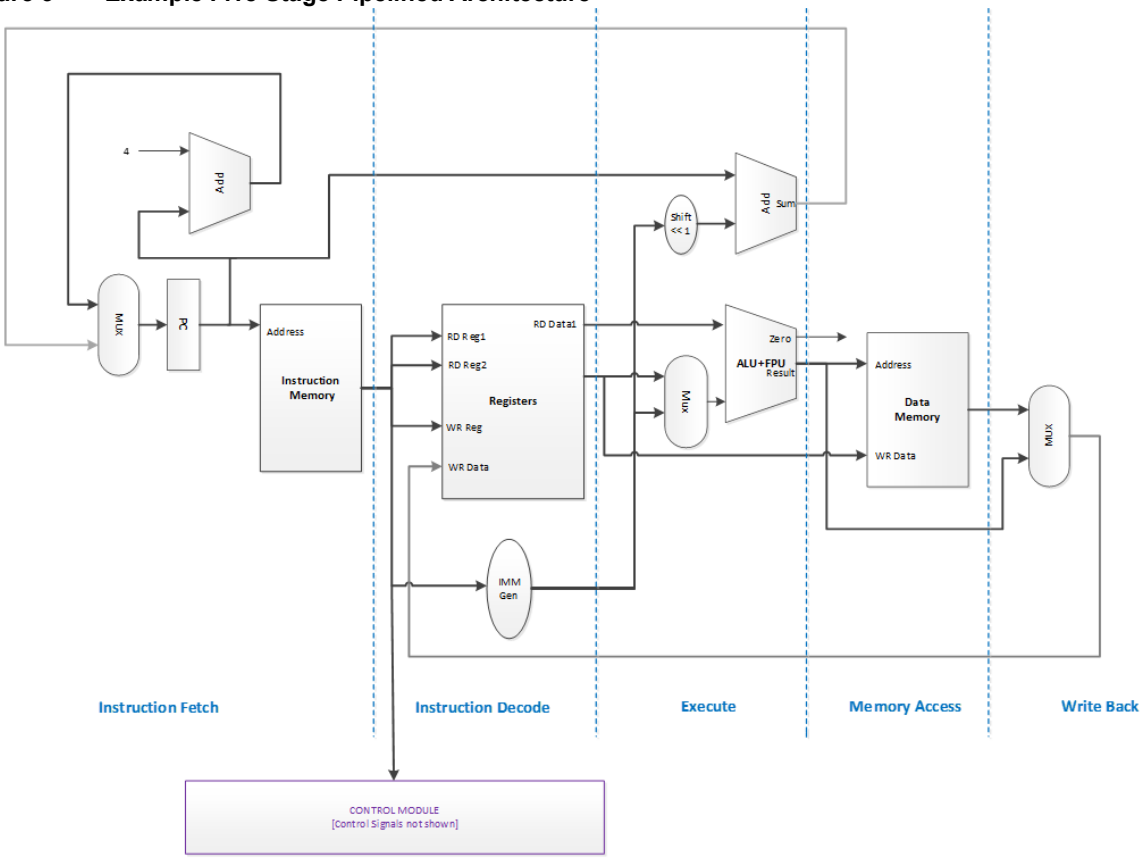


Table 3 • **Example Pipeline Timing**

Cycle Clock	1	2	3	4	5	6		n
Fetch	Instruction1	Instruction2	Instruction 3	Instruction4	Instruction5	Instruction 6	.....	Instruction n
Decode		Decode Instruction1	Decode Instruction 2	Decode Instruction3	Decode Instruction4	Decode Instruction 5	.....	Decode Instruction n-1
Execute			Execute Instruction 1	Execute Instruction 2	Execute Instruction 3	Execute Instruction 4	.....	Execute Instruction n-2
Mem access				RD\WR Memory 1	Memory Access 2	Memory Access 3	.....	Memory Access n-3
Writeback					Write Back Result 1	Write Back 2	.....	Writge Back n-4

3.4 Memory System

MIV_RV32IMAF_L1_AHB memory system supports configurable split first-level instruction and data caches, with support for hardware cache flushing, and for non-cached memory accesses. The external connections are provided for cached and non-cached TileLink fabrics.

3.5 Platform-Level Interrupt Controller

MIV_RV32IMAF_L1_AHB includes a RISC-V standard Platform-Level Interrupt Controller (PLIC) configured to support upto 31 inputs with a single priority level 0. The 31 active high inputs are serviced from 0 to 31 in ascending order.

3.6 Debug support through JTAG

MIV_RV32IMAF_L1_AHB includes full external debugger support over an industry-standard JTAG port, supporting two hardware breakpoints.

3.7 External AHB Interfaces

MIV_RV32IMAF_L1_AHB includes two external AHB interfaces, bridged from the internal TileLink interfaces. The cache controller uses the AHB memory interface to refill the instruction and data caches. The AHB I/O interface is typically used for non-cached accesses to I/O peripherals.

Note: Instructions read from memory on the I/O interface will be routed by the cache controller to the instruction cache. In this instance, burst mode can be seen on the AHB I/O interface.

3.8 ECC

MIV_RV32IMAF_L1_AHB includes optional Error-Correcting Code (ECC) functionality with Single Error Correction and Double Error Detection (SECDED) event flags. The ECC caches are initialized and these signals assert to alert the user that, an error is corrected or detected on the instruction or data cache. When ECC is enabled, the synthesis attributes are used on the FPU and ALU to infer logic instead of using the hardware Math blocks.

4 Interface

4.1 Configuration Parameters

4.1.1 MIV_RV32IMAF_L1_AHB Configurable Options

The two configurable options that apply to MIV_RV32IMAF_L1_AHB are shown in Table 4. If a configuration other than the default is required, use the configuration dialog box in SmartDesign to select the appropriate values for the configurable options. The MIV_RV32IMAF_L1_AHB core has two interfaces, cached (AHB_MST_MEM) and non-cached (AHB_MST_MMIO). The cache range is from 0x8000_0000 to 0x8FFF_FFFF. The non-cache range is from 0x6000_0000 to 0x7FFF_FFFF.

Table 4 • **MIV_RV32IMAF_L1_AHB Configuration Options**

Parameter	Valid Range	Default	Description
RESET_VECTOR_ADDR (high halfword)	0x6000 - 0x8FFF	0x6000	This is the address the processor will start executing from after a reset. This address must be word aligned.
RESET_VECTOR_ADDR (low halfword)	0x0000 – 0xFFFC	0x0	
ECC_EN	Y/N		This enables ECC and exposes the SECDED pins for both instruction and data caches. It will also instantiate a version of the FPU and ALU that uses logic instead of math blocks. A synthesis attribute is also applied to the working register (x0-x31) so a USRAM is not inferred.

4.1.2 Signal Descriptions

The signal descriptions for MIV_RV32IMAF_L1_AHB are defined in Table 5.

Table 5 • **MIV_RV32IMAF_L1_AHB I/O Signals**

Port Name	Width	Direction	Description
		Global Signals	
CLK	1	Input	System clock. All other I/Os are synchronous to this clock.
RESETN	1	Input	Synchronous reset signal. Active Low.
EXT_RESETN	1	Output	This is an optional active low reset output from the debug module, which can be used to automatically reset SoC components or external devices during a debug session.
		JTAG Interface Signals	
TDI	1	Input	Test Data In (TDI). This signal is used by the JTAG device for downloading and debugging programs. Sampled on the rising edge of TCK. If this signal is not being used it must be tied low.
TCK	1	Input	Test Clock (TCK). This signal is used by the JTAG device for downloading and debugging programs. If this signal is not being used it must be tied low.
TMS	1	Input	Test Mode Select (TMS). This signal is used by the JTAG device when downloading and debugging programs. It is sampled on the rising edge of TCK to determine the next state. If this signal is not being used it must be tied low.
TRST	1	Input	Test Reset (TRST). This is an optional signal used to reset the TAP controllers state machine. If this signal is not being used it must be tied high.
TDO	1	Output	Test Data Out (TDO). This signal is the data which is shifted out of the device during debugging. It is valid on FALLING/RISING edge of TCK. If this signal is not being used it must be marked as unused.
DRV_TDO	1	Output	Drive Test Data Out (DRV_TDO). If this signal is not being used it must be marked as unused.
		External Interrupts Signals	
IRQ	31	Input	External interrupts from off-chip or peripheral sources. These are active high level-based interrupt signals.
		Parameterized Signals	
ICACHE_SEC	1	Output	The ICACHE_SEC pin will assert if a single error correction has occurred in the instruction cache.
ICACHE_DED	1	Output	The ICACHE_DED pin will assert if a double error detection has occurred in the instruction cache.
DCACHE_SEC	1	Output	The DCACHE_SEC pin will assert if a single error correction has occurred in the data cache.
DCACHE_DED	1	Output	The DCACHE_DED pin will assert if a double error detection has occurred in the data cache.

Table 5 • **MIV_RV32IMAF_L1_AHB I/O Signals *(continued)***

Port Name	Width	Direction	Description
AHB Cached Memory Bus Master Interface			
AHB_MST_MEM_HLOCK	1	Output	AHB Master interface for cached memory accesses
AHB_MST_MEM_HTRANS	2	Output	
AHB_MST_MEM_HSEL	1	Output	
AHB_MST_MEM_HWRITE	1	Output	
AHB_MST_MEM_HADDR	32	Output	
AHB_MST_MEM_HSIZE	3	Output	
AHB_MST_MEM_HBURST	3	Output	
AHB_MST_MEM_HPROT	4	Output	
AHB_MST_MEM_HWDATA	32	Output	
AHB_MST_MEM_HREADY	1	Input	
AHB_MST_MEM_HRESP	1	Input	
AHB_MST_MEM_HRDATA	32	Input	
AHB Non-Cached Memory Mapped IO Interface			
AHB_MST_MMIO_HLOCK	1	Output	AHB Master Interface for non-cached memory accesses.
AHB_MST_MMIO_HTRANS	2	Output	
AHB_MST_MMIO_HSEL	1	Output	
AHB_MST_MMIO_HWRITE	1	Output	
AHB_MST_MMIO_HADDR	31	Output	
AHB_MST_MMIO_HSIZE	3	Output	
AHB_MST_MMIO_HBURST	3	Output	
AHB_MST_MMIO_HPROT	4	Output	
AHB_MST_MMIO_HWDATA	32	Output	
AHB_MST_MMIO_HREADY	1	Input	
AHB_MST_MMIO_HRESP	1	Input	
AHB_MST_MMIO_HRDATA	32	Input	

5 Register Map and Descriptions

This section describes the Register Map and gives a description of each of the devices within the core.

A concise list of each device is shown in Table 6.

Table 6 • **Physical Memory Map (from Rocket-Chip)**

Base	Top	Description
0x0000_0000	0x0000_1000	Debug Controller
0x0000_3000	0x0000_4000	Error Device
0x4000_0000	0x4400_0000	Platform-Level Interrupt Control (PLIC)
0x4400_0000	0x4401_0000	Core Local Interrupt (CLINT)
0x6000_0000	0x7FFF_FFFF	AHB non-cached interface
0x8000_0000	0x8FFF_FFFF	AHB cached Interface

5.1 Debug Controller

This section describes the operation of the debug hardware used in the MIV_RV32IMAF_L1_AHB. The debug hardware follows The RISC-V Debug Specification v0.13. This core supports interactive debug and two hardware breakpoints.

5.1.1 Debug Control and Status Registers

This table describes the debug registers and how they are mapped. For a comprehensive description of the Debug CSRs, see Chapter 4, in The RISC-V Debug Specification v0.13.

Table 7 • **Debug Control and Status Registers**

Debug CSRs		
CSR	**Name**	**Description**
dcsr	0x7b0	Debug control and status register provides information on the debug capabilities and the status of the debug unit.
dpc	0x7b1	Debug Program Counter. When debug mode is entered, the current PC is copied here. When leaving debug mode, execution will resume at the address stored at this Program Counter.
dscratch	0x7b2	Debug scratch register is reserved for use by the Debug ROM.
tselect	0x7a0	Trace and debug register select.
tdata1	0x7a1	First field of Trace and debug register.
tdata2	0x7a2	Second field of Trace and debug register.
tdata3	0x7a3	Third field of Trace and debug register.

5.2 Error Device

This is a TileLink slave that responds to all requests with a TileLink error. It does not have any registers. TileLink slave discards all writes to this memory range and reads returns zeros. The error device is used when an illegal off-chip requests are made.

5.3 Platform Level Interrupt Controller (PLIC)

MIV_RV32IMAF_L1_AHB includes a RISC-V standard Platform-Level Interrupt Controller (PLIC) configured to support up to 31 active high inputs with a single priority level 0. The interrupt inputs are handled from 0 – 31. The PLIC complies with The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. PLIC Register Map follows:

Table 8 • **Platform Level Interrupt Controller (PLIC)**

PLIC Register Map		
Address	**Width**	**Description**
0x4000_0000		Reserved
0x4000_0004	32	Source 1 priority
.....		
0x4000_0078	32	Source 31 priority
0x4000_1000	32	Interrupt Pending array
0x4000_2000	32	Interrupt Enable array

5.4 Core Local Interrupt (CLINT)

CLINT is a memory mapped control and status register, which controls the software and timer interrupts. CLINT complies with RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10. CLINT Register Map follows:

Table 9 • **Core Local Interrupt (CLINT)**

CLINT Register Map		
Address	**Width**	**Description**
0x4400_0000	32	MSIP Register
0x4400_4000	64	MTIMECMP Register
0x4400_bff8	64	Timer Register

6 Interrupts

This section describes the operation of the interrupts in the MIV_RV32IMAF_L1_AHB. For a comprehensive understanding of how the interrupts work, see **The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10**.

6.1 Interrupt Control and Status Registers

MIV_RV32IMAF_L1_AHB interrupt CSRs follows:

6.1.1 Machine Status Register (mstatus)

This register controls the MIV_RV32IMAF_L1_AHBs current operating state, including if interrupts are enabled or disabled. The interrupt bits from the mstatus register is as follows. For a full list of the mstatus register, see **The RISC-V Instruction Set Manual Volume II: Privileged Architecture, Version 1.10**.

Table 10 • **Machine Status Register (mstatus)**

Machine Status Register			
CSR	mstatus	**CSR address**	0x300
Bits	**Name**	**Description**	
3	MIE	Machine Interrupt Enable	
7	MPIE	Machine Previous Interrupt Enable	
12	MPP	Machine Previous Privilege Mode	

6.1.2 Machine Interrupt Enable (mie)

Each interrupt is enabled or disabled by setting the corresponding bit in the following mie register.

Table 11 • **Machine Interrupt Enable (mie)**

Machine Interrupt Enable Register			
CSR	mie	**CSR address**	0x304
Bits	**Name**	**Description**	
3	MSIP	Machine Software Interrupt Enable	
7	MTIE	Machine Timer Interrupt Enable	
11	MEIE	Machine External Interrupt Enable	

6.1.3 Machine Interrupt Pending (mip)

This register indicates which interrupts are outstanding.

Table 12 • **Machine Interrupt Pending (mip)**

Machine Interrupt Pending Register			
CSR	mip	**CSR address**	0x344
Bits	**Name**	**Description**	
3	MSIP	Machine Software Interrupt Pending	
7	MTIP	Machine Timer Interrupt Pending	
11	MEIP	Machine External Interrupt Pending	

6.1.4 Machine Cause (mcause)

This register is used when a trap is taken in machine mode and code is written to mcause, which indicates what caused the trap. The mcause register and a list of exception codes can be seen below.

Table 13 • **Machine Cause (mcause)**

Machine Cause Register			
CSR	mcause	**CSR address**	0x342
Bits	**Name**	**Description**	
[9:0]	Exception Code	Identification code of the last exception	
[30:10]	Reserved		
31	Interrupt	Asserted if trap caused by interrupt otherwise 0.	

Table 14 • **mcause Exception Codes**

mcause Exception codes		
Interrupt	Execution Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	Reserved
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	Reserved
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	Reserved
1	11	Machine external interrupt
1	≥12	Reserved

Table 14 • **mcause Exception Codes *(continued)***

mcause Exception codes		
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	Reserved
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	load page fault
0	14	Reserved
0	15	Store/AMO page fault
0	16	Reserved

7 Tool Flow

7.1 License

This core is released under a modified Apache 2.0 license and is freely available through Libero.

7.1.1 RTL

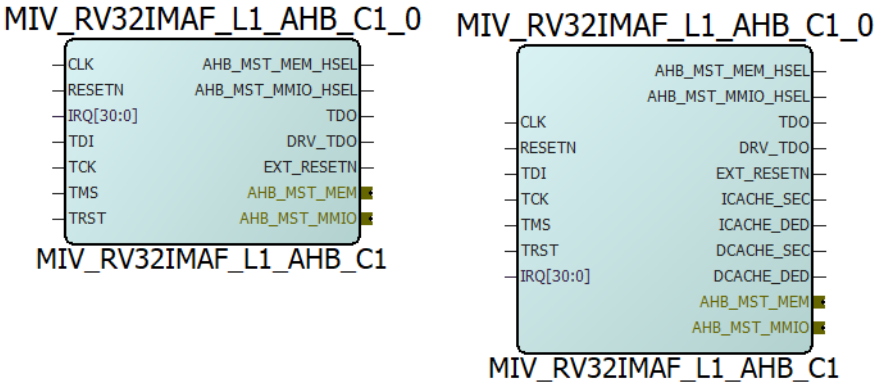
Complete Verilog source code is provided for the core, allowing the core to be instantiated with SmartDesign. Simulation, Synthesis, and Layout is performed within Libero SoC.

7.2 SmartDesign

MIV_RV32IMAF_L1_AHB is preinstalled in SmartDesign IP Deployment design environment.

For more information on using SmartDesign to instantiate and generate cores, refer to the **Using DirectCore in Libero® SoC User Guide**.

Figure 4 • **SmartDesign MIV_RV32IMAF_L1_AHB Instance View**

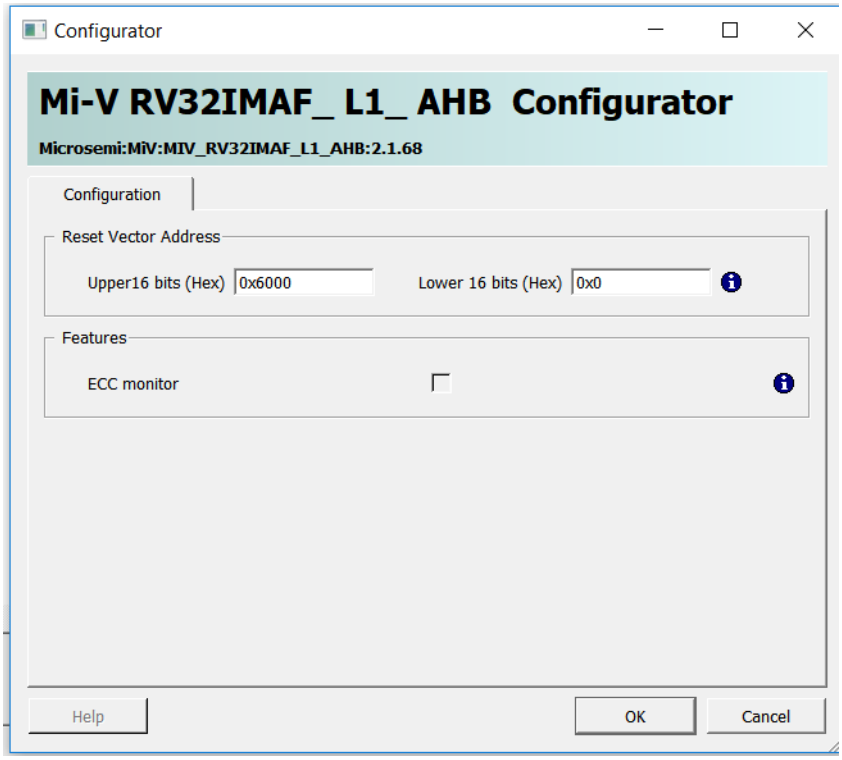


7.3 Configuring MIV_RV32IMAF_L1_AHB in SmartDesign

The core is configured using the configuration GUI within SmartDesign, as shown in Figure 5.

Note: Leading zeros are suppressed, for example, 0x6000 0000 is displayed as 0x6000 0x0. The reset vector address is word aligned. It should also be noted that the ECC monitor option is only available for families that support ECC ram blocks.

Figure 5 • **Configuring MIV_RV32IMAF_L1_AHB in SmartDesign on SmartFusion2**



7.4 Debugging

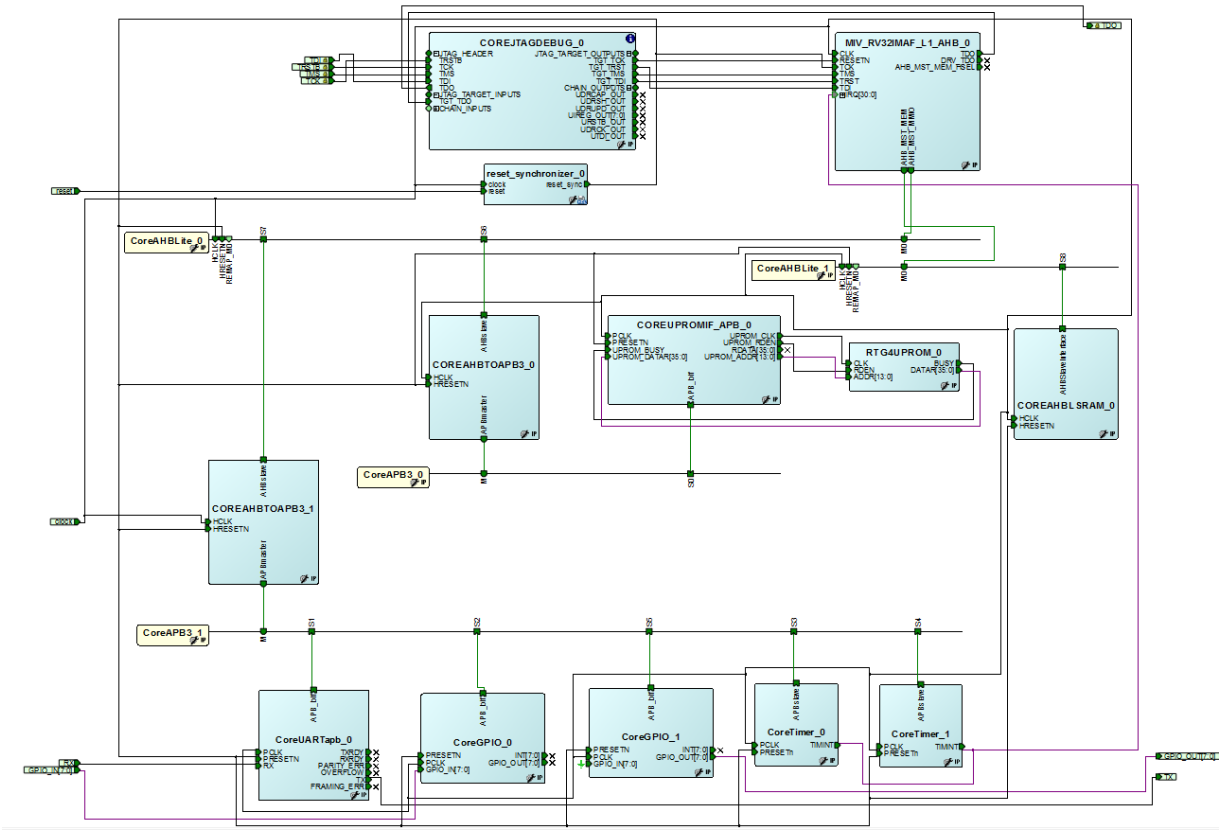
CoreJTAGDebug V3.0.100 or higher, is used to enable debugging of MIV_RV32IMAF_L1_AHB. This is available in the Libero catalog.

7.5 Simulation Flows

The user testbench for MIV_RV32IMAF_L1_AHB is omitted from this release.

MIV_RV32IMAF_L1_AHB RTL can be used to simulate the processor executing a program, using a standard Libero generated HDL testbench. An example subsystem for RTG4 is shown in Figure 6.

Figure 6 • **RTG4 Example Simulation Subsystem**



7.6 Synthesis in Libero

To run synthesis on the core, set the SmartDesign sheet as the design root, and click **Synthesis** in Libero SoC.

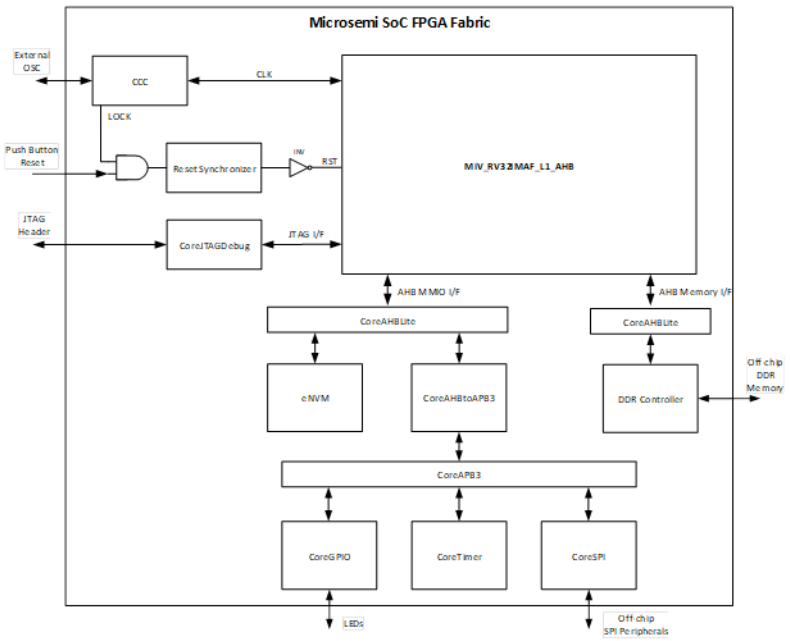
7.7 Place-and-Route in Libero

When the design is synthesized, run the compilation and the place and-route tools. Click **Layout** in Libero SoC to start Designer. MIV_RV32IMAF_L1_AHB requires the place-and-route multi-seed settings, which is set to 5.

8 System Integration

8.1 Example System

Figure 7 • **MIV_RV32IMAF_L1_AHB Example System**

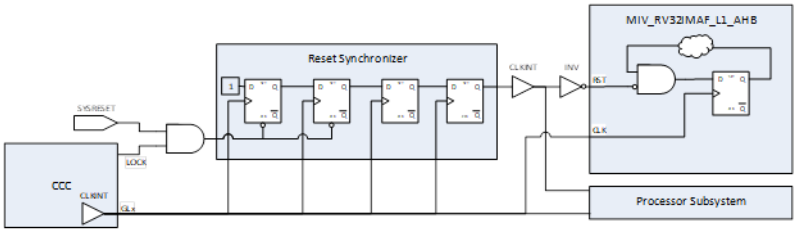


8.2 Reset Synchronization

8.2.1 RST

All the sequential elements are clocked by *CLK* within MIV_RV32IMAF_L1_AHB, which require a reset to employ a synchronous reset topology. Since, most designs source *CLK* from a CCC or PLL, it is common practice to AND the *LOCK* output of the CCC with the push button reset to generate the *RST* input for MIV_RV32IMAF_L1_AHB. However, this results in the reset being deasserted when the *CLK* comes up, hence, the reset assertion is not clocked through the sequential reset elements and goes unnoticed, most commonly leading to the processor locking-up. To guarantee that the RST assertion is seen by all sequential elements, a reset synchronizer is required on the *RST* input, shown in Figure 8.

Figure 8 • **RST Reset Synchronization**



The Verilog code snippet below implements the reset synchronizer block shown in Figure 8. The function of this block is to make the reset assertion and deassertion synchronous to CLK while guaranteeing that the reset is asserted for one or more CLK cycles within MIV_RV32IMAF_L1_AHB. This will ensure that it is registered by all sequential elements.

```
module reset_synchronizer (
    input  clock,
    input  reset,
    output reset_sync
);
reg [1:0]   sync_deasert_reg;
reg [1:0]   sync_asert_reg;

always @ (posedge clock or negedge reset)
    begin
        if (!reset)
            begin
                sync_deasert_reg[1:0] <= 2'b00;
            end
        else
            begin
                sync_deasert_reg[1:0] <= {sync_deasert_reg[0], 1'b1};
            end
    end
always @ (posedge clock)
    begin
        sync_asert_reg[1:0] <= {sync_asert_reg[0], sync_deasert_reg[1]};
    end
assign reset_sync = sync_asert_reg[1];
endmodule
```

To include this synchronizer in your Libero design, select **Create HDL** from the **Design Flow** tab in your **Libero** project. In the popup window, name the HDL file accordingly and select **Verilog** as the HDL type, while clearing the option to initialize file with the standard template. Copy and paste the **Verilog code** snippet preceding this file and save the changes. From the **Design Hierarchy** tab, drag and drop the file into the **SmartDesign** sheet, containing the MIV_RV32IMAF_L1_AHB instance and connect up the pins as shown earlier.

8.2.2 TRST

Reset synchronization is not required, on this reset input as all the sequential elements in the debug logic within MIV_RV32IMAF_L1_AHB uses an asynchronous reset topology.

9 Design Constraints

Designs containing MIV_RV32IMAF_L1_AHB require the application of the following constraints, in the design flow, to allow timing-driven placement and static timing analysis, to be performed on MIV_RV32IMAF_L1_AHB. The procedure for adding the required constraints in the Enhanced Constraints flow in Libero v11.8 or higher is as follows:

1. Double-click **Constraints > Manage Constraints** in the **Design Flow** window, and click the **Timing** tab.

For example, the system clock is used to clock MIV_RV32IMAF_L1_AHB, which is sourced from a PLL. Select **Derive** to automatically create a constraints file containing the PLL constraints. Select **Yes** when prompted to allow the constraints to be automatically included for Synthesis, Place-and-Route, and Timing Verification stages.

If changes are made to the PLL configuration in the design, update the contents of this file by clicking **Derive**. Select **Yes** when prompted to allow the constraints to be overwritten.

2. In the **Timing** tab of the **Constraint Manager** window, select **New** to create a new SDC file, and enter the file name. Design constraints other than the system clock source derived constraints are entered in this blank SDC file. Keeping derived and manually added constraints in separate SDC files allowing the **Derive** stage to be reperformed, if changes are made to the PLL configuration, without deleting all manually added constraints in the process.
3. Calculate the TCK period and half period. TCK is typically 6 MHz when debugging with FlashPro, with a maximum frequency of 30 MHz supported by FlashPro5. After completion, enter the following constraints in the blank SDC file:

```
create_clock -name { TCK } \
    -period TCK_PERIOD \
    -waveform { 0 TCK_HALF_PERIOD } \
    [ get_ports { TCK } ]
```

For example: The following constraints need to be applied for a design that uses a TCK frequency of 6 MHz:

```
create_clock -name { TCK } \
    -period 166.67 \
    -waveform { 0 83.33 } \
    [ get_ports { TCK } ]
```

4. Constraints must be applied to paths crossing the clock domain crossing between the TCK and system clock domains. MIV_RV32IMAF_L1_AHB implements two clock domain crossing FIFOs to handle the CDC and as such paths between the two clock domains may be declared as false paths to prevent minimum and maximum violations from being reported by SmartTime.

```
set_false_path -from [ get_clocks { TCK } ] \
               -to [ get_clocks { PLL_GEN_CLK } ]
set_false_path -from [ get_clocks { PLL_GEN_CLK } ] \
               -to [ get_clocks { TCK } ]
```

Where:

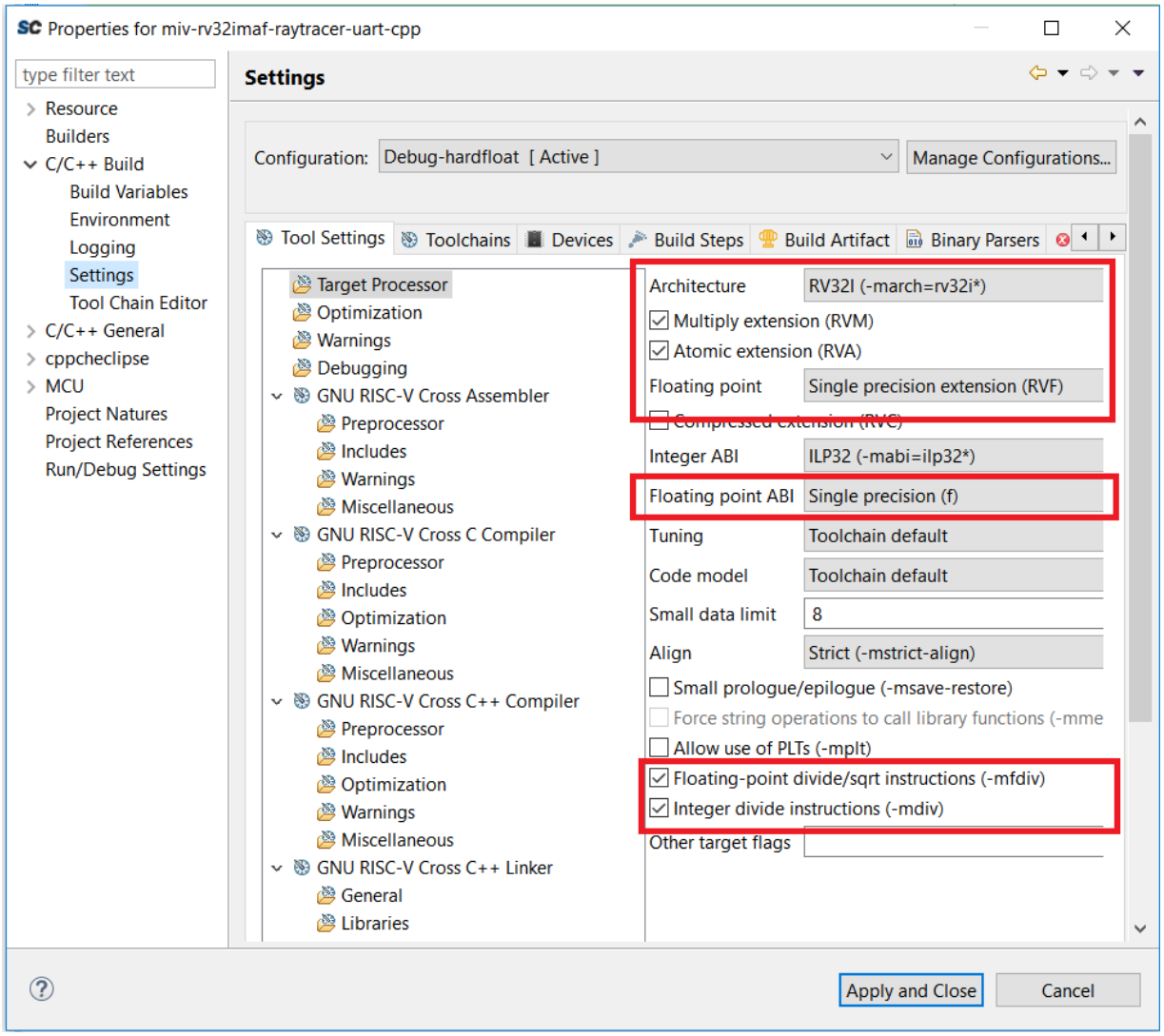
- PLL_GEN_CLK is the name applied to the create_generated_clock constraint derived in step 1 earlier.

5. Associate all constraints files with the Synthesis, Place-and-Route, and Timing Verification stages in the **Constraint Manager > Timing** tab by selecting the related check boxes for the SDC files in which the constraints were entered.

10 SoftConsole

SoftConsole version 6.0 is required to use the MIV_RV32IMAF_L1_AHB. Each SoftConsole project requires the Hardware Abstraction Layer (HAL) version 2.1 or higher. The SoftConsole Release Notes details how to set up a project for the MIV cores. Example projects exists in the example workspace, which shows how to set up a project for the MIV_RV32IMAF_L1_AHB. A list of the compiler flags that are set to use the Floating Point Unit as shown:

Figure 9 • **SoftConsole Flags**



11 Known Issues

11.1 Reset or Power Cycle the Target Hardware before each Debug Session

At the moment, the debugger cannot effect a suitable MI-V RISC-V CPU/SoC reset at the start of each debug session. So one debug session is impacted by what went before – for example, a previous debug session leaves the CPU in an ISR and a subsequent debug session does not behave as expected because of this. To mitigate this problem, it is recommended that the target hardware or board is power cycled or otherwise reset before each new debug session.