

**HB0911**  
**Handbook**  
**MIV\_RV32IMC v2.1**

March 2020



---

a  **MICROCHIP** company

# Contents

---

Revision History.....	1
1.1 Revision 1.0.....	1
2 Introduction.....	2
2.1 Overview.....	2
2.2 Features.....	2
2.3 Core Version.....	2
2.4 Supported Families.....	2
2.5 Device Resource Utilization and Performance.....	3
2.5.1 Typical Resource Utilization.....	6
2.5.2 Benchmarks.....	7
3 Functional Description.....	9
3.1 MIV_RV32IMC Architecture.....	9
3.2 Hart.....	10
3.3 Memory System.....	10
3.4 Interrupts.....	10
3.5 Debug Support via JTAG.....	11
3.6 External Interfaces.....	11
3.7 Tightly Coupled Memory.....	11
3.8 Direct Access Port.....	11
3.9 Clocks.....	11
3.10 Resets.....	11
4 Interface.....	13
4.1 Configuration Parameters.....	13
4.2 I/O Signals.....	16
5 Programmer's Model.....	20
5.1 Processor Operating States.....	20
5.2 Reset Operation.....	20
5.3 Data Types.....	20
5.4 General Purpose Registers.....	20
5.5 Machine Control and Status Registers.....	21
5.6 Debug Module.....	28
5.6.1 Debug Transport Module.....	28
5.6.2 Debug Unit.....	29
5.6.3 Hart Debug Logic.....	30
5.7 Memory Map.....	32
5.8 Subsystem Restrictions.....	34
5.9 Exceptions.....	34
5.9.1 Vectored and Non-Vectored Interrupts.....	35
5.9.2 Nested Interrupts.....	35
5.9.3 Available Interrupts.....	35
5.9.4 Interrupt Handling.....	35
5.9.5 Vectored Interrupt Offsets and Exception Priorities.....	36
5.9.6 OPSRV Register Interrupts.....	36
5.10 OPSRV Register.....	37
5.11 MTIME.....	38
5.12 ECC.....	40

<b>6 Tool Flow</b> .....	<b>42</b>
6.1 License.....	42
6.1.1 RTL.....	42
6.2 SmartDesign.....	42
6.3 Configuring MIV_RV32IMC.....	43
6.3.1 Extension Options.....	43
6.3.2 Interface Options.....	44
6.3.3 Reset Vector Address.....	44
6.3.4 Interrupt Options.....	44
6.3.5 Tightly Coupled Memory (TCM) Options.....	44
6.3.6 Other Options.....	44
6.3.7 Memory Map Tab.....	45
6.4 Debugging.....	45
6.5 Simulation Flows.....	45
6.6 Synthesis in Libero.....	47
6.7 Place-and-Route in Libero.....	47
<b>7 System Integration</b> .....	<b>48</b>
7.1 PolarFire Example System.....	48
7.2 RTG4/SF2/IG2 Example System.....	48
7.3 Reset Synchronization.....	49
7.3.1 RESETN.....	49
7.3.2 TRST.....	50
<b>8 Design Constraints</b> .....	<b>51</b>
<b>9 SoftConsole</b> .....	<b>52</b>
9.1 Setting the System Clock Frequency and Peripheral Base Addresses.....	52

# 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 1.0

Revision 1.0 was published in March 2020. This is the first publication of the MIV\_RV32IMC IP.

## 2 Introduction

---

### 2.1 Overview

The MIV\_RV32IMC is a processor core designed to implement the RISC-V instruction set for use in Microchip FPGAs.

The core includes an industry standard JTAG interface to facilitate debug access. Three optional bus interfaces are available for peripheral and memory accesses. They are AHB, APB3, and AXI which can be configured as AXI3 or AXI4.

There are three dedicated interrupts as well as six optional external interrupts.

A quick start guide is available on how to create an MIV\_RV32IMC Libero design from the help menu in the core configurator.

### 2.2 Features

- Designed for low power FPGA soft-core implementations
- Supports the RISC-V standard RV32I ISA with optional Multiply and Divide (M) and Compressed (C) extensions
- Tightly coupled memory is available and size is defined by address range
- Direct Access Port (DAP) to TCM
- External, Timer and Soft Interrupts
- Up to six optional external interrupts
- Vectored and non-vectored interrupt support
- Optional on-chip debug unit with a JTAG interface
- AHBL, APB3, and AXI3/AXI4 optional external bus interfaces

### 2.3 Core Version

This Handbook applies to MIV\_RV32IMC version 2.1.

**Note:** The five accompanying manuals for this core are as follows:

- The RISC-V Instruction Set Manual Volume I: Unprivileged ISA
- The RISC-V Instruction Set Manual Volume II: Privileged Architecture
- RISC-V External Debug Support Version 0.13.2
- MIV\_RV32IMC Quick Start Guide
- Supplementary Resource Utilization and Performance (RUP) tables

### 2.4 Supported Families

- PolarFire<sup>®</sup>
- RTG4<sup>™</sup>
- IGLOO<sup>®</sup>2
- SmartFusion<sup>®</sup>2

## 2.5 Device Resource Utilization and Performance

The device Resource Utilization and Performance (RUP) data is listed in tables 1 to 9 for the supported device families. This data is indicative only. The overall resource utilization and performance of the core is system dependent.

The entire RUP data was generated using Libero SoC v 12.3 and Synplify v2019.03M-SP1. The **P&R LEs** signify the number of logic elements used in the synthesized component for benchmarking. This value is for reference only and varies between place-and-route runs. The following tables list the device resource utilization and performance for selected configurations of the processor.

**Table 1 • RV32I APB TCM**

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	984	3899	4883	4105	109.206
RTG4	RTG4150L FCG1657	984	3829	4813	4017	89.815
SmartFusion2	M2S150T FC1152	982	3852	4834	4059	89.654
IGLOO2	M2GL150 FC1152	982	3852	4834	4059	89.654
Configuration Parameters	RISC-V Extensions: I, Multiplier: n, AHB Master: n, AHB Mirrored I/F: n, APB Master: APB3, APB Mirrored I/F: n, AXI Master: n, AXI Mirrored I/F: n, Reset Vector Address Upper 16 bits: 0x4000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 0, MTVEC Offset: 0x34, Vectored Interrupts: n, TCM: y (4 k), TCM Direct Access Port: n, Internal MTIME: n, Internal MTIME IRQ: n, Debug: n, Register Forwarding: n, ECC: n, GPR Registers: n					

**Table 2 • RV32I APB All Features**

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	2663	7901	10564	8471	67.249
RTG4	RTG4150L FCG1657	2660	7746	10406	8433	56.577
SmartFusion2	M2S150T FC1152	2660	7787	10447	8411	72.129
IGLOO2	M2GL150 FC1152	2660	7787	10447	8411	72.129
Configuration Parameters	RISC-V Extensions: I, Multiplier: n, AHB Master: AHB Lite, AHB Mirrored I/F: y, APB Master: APB3, APB Mirrored I/F: n, AXI Master: y, AXI Mirrored I/F: y, Reset Vector Address Upper 16 bits: 0x8000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 6, MTVEC Offset: 0x34, Vectored Interrupts: y, TCM: y (4 k), TCM Direct Access Port: y, Internal MTIME: y, Internal MTIME IRQ: y, Debug: y, Register Forwarding: y, ECC: y, GPR Registers: y					

**Table 3 • RV32IC APB TCM**

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	982	4165	5147	4355	98.678
RTG4	RTG4150L FCG1657	986	4230	5216	4428	83.417
SmartFusion2	M2S150T FC1152	987	4240	5227	4446	95.822
IGLOO2	M2GL150 FC1152	987	4240	5227	4446	95.822
Configuration Parameters	RISC-V Extensions: IC, Multiplier: n, AHB Master: n, AHB Mirrored I/F: n, APB Master: APB3, APB Mirrored I/F: n, AXI Master: n, AXI Mirrored I/F: n, Reset Vector Address Upper 16 bits: 0x4000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 0, MTVEC Offset: 0x34, Vectored Interrupts: n, TCM: y (4 k), TCM Direct Access Port: n, Internal MTIME: n, Internal MTIME IRQ: n, Debug: n, Register Forwarding: n, ECC: n, GPR Registers: n					

**Table 4 • RV32IC APB All Features**

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	2666	8236	10902	8800	68.923
RTG4	RTG4150L FCG1657	2670	8104	10774	8756	55.941
SmartFusion2	M2S150T FC1152	2661	8200	10861	8849	70.957
IGLOO2	M2GL150 FC1152	2661	8200	10861	8849	70.957
Configuration Parameters	RISC-V Extensions: IC, Multiplier: n, AHB Master: AHB Lite, AHB Mirrored I/F: y, APB Master: APB3, APB Mirrored I/F: n, AXI Master: y, AXI Mirrored I/F: y, Reset Vector Address Upper 16 bits: 0x8000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 6, MTVEC Offset: 0x34, Vectored Interrupts: y, TCM: y (4k), TCM Direct Access Port: y, Internal MTIME: y, Internal MTIME IRQ: y, Debug: y, Register Forwarding: y, ECC: y, GPR Registers: y					

**Table 5 • RV32IM (MACC-Pipelined) APB TCM**

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	1120	4442	5562	4802	87.093
RTG4	RTG4150L	1127	4529	5656	4879	83.696

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
	FCG1657					
SmartFusion2	M2S150T FC1152	1122	4568	5690	4918	88.708
IGLOO2	M2GL150 FC1152	1122	4568	5690	4918	88.708
Configuration Parameters	RISC-V Extensions: IM, Multiplier: MACC-Pipelined, AHB Master: n, AHB Mirrored I/F: n, APB Master: APB3, APB Mirrored I/F: n, AXI Master: n, AXI Mirrored I/F: n, Reset Vector Address Upper 16 bits: 0x4000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 0, MTVEC Offset: 0x34, Vectored Interrupts: n, TCM: y (4k), TCM Direct Access Port: n, Internal MTIME: n, Internal MTIME IRQ: n, Debug: n, Register Forwarding: n, ECC: n, GPR Registers: n					

Table 6 • RV32IM (MACC-Pipelined) All Features

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	2808	8372	11180	9061	67.522
RTG4	RTG4150L FCG1657	2798	8374	11172	9162	58.224
SmartFusion2	M2S150T FC1152	2798	8450	11248	9228	71.235
IGLOO2	M2GL150 FC1152	2798	8450	11248	9228	71.235
Configuration Parameters	RISC-V Extensions: IM, Multiplier: MACC-Pipelined, AHB Master: y, AHB Mirrored I/F: y, APB Master: APB3, APB Mirrored I/F: n, AXI Master: y, AXI Mirrored I/F: y, Reset Vector Address Upper 16 bits: 0x8000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 6, MTVEC Offset: 0x34, Vectored Interrupts: y, TCM: y (4k), TCM Direct Access Port: y, Internal MTIME: y, Internal MTIME IRQ: y, Debug: y, Register Forwarding: y, ECC: y, GPR Registers: y					

Table 7 • RV32IMC (Fabric) APB TCM

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	1157	5170	6327	5374	95.841
RTG4	RTG4150L FCG1657	1159	5197	6356	5421	81.907
SmartFusion2	M2S150T FC1152	1155	5191	6346	5403	86.498

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
IGLOO2	M2GL150 FC1152	1155	5191	6346	5403	86.498
Configuration Parameters	RISC-V Extensions: IMC, Multiplier: Fabric, AHB Master: n, AHB Mirrored I/F: n, APB Master: APB3, APB Mirrored I/F: n, AXI Master: n, AXI Mirrored I/F: n, Reset Vector Address Upper 16 bits: 0x4000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 0, MTVEC Offset: 0x34, Vectored Interrupts: n, TCM: y (4k), TCM Direct Access Port: n, Internal MTIME: n, Internal MTIME IRQ: n, Debug: n, Register Forwarding: n, ECC: n, GPR Registers: n					

Table 8 • RV32IMC (Fabric) All Features

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	2835	9265	12100	9803	64.943
RTG4	RTG4150L FCG1657	2883	9203	12036	9845	57.840
SmartFusion2	M2S150T FC1152	2883	9251	12084	9908	69.238
IGLOO2	M2GL150 FC1152	2883	9251	12084	9908	69.238
Configuration Parameters	RISC-V Extensions: IMC, Multiplier: Fabric, AHB Master: y, AHB Mirrored I/F: y, APB Master: APB3, APB Mirrored I/F: n, AXI Master: y, AXI Mirrored I/F: y, Reset Vector Address Upper 16 bits: 0x8000, Reset Vector Address Lower 16 bits: 0x0, External IRQs: 6, MTVEC Offset: 0x34, Vectored Interrupts: y, TCM: y (4k), TCM Direct Access Port: y, Internal MTIME: y, Internal MTIME IRQ: y, Debug: y, Register Forwarding: y, ECC: y, GPR Registers: y					

For more information, see the Supplementary RUP Tables manual, which is included with the core.

## 2.5.1 Typical Resource Utilization

The following table lists a breakdown of average resource usage for core options across the supported families.

Table 9 • Option Resources

Feature	Parts	Synthesis		
		Avg. DFF	Avg. 4LUT	Avg. Total
AHBL	MPF500T-1FCG1152E	108	102	210
APB	RTG4150L FCG1657 M2S150T FC1152	115	144	259
AXI	M2GL150 FC1152	514	492	1006
Ext_sys_interrupts (6)		6	61	67

Feature	Parts	Synthesis		
		Avg. DFF	Avg. 4LUT	Avg. Total
Vectored interrupts		35	154	189
TCM (4 k)		62	372	434
DAP		0	84	84
Mtime and Mtime irq		160	425	585
Debug		564	1756	2320
ECC		12	407	429
GPR registers		989	1544	2533
Register forwarding		5	289	294

## 2.5.2 Benchmarks

Table 10 • Coremark Results

Benchmarks	Memory Location	RV32	Multiplier	Reg Fwd	Reg GPRs	ECC	Coremark/MHz
Extension bench- marks	TCM	IMC	MACC	1	1	0	2.767
	TCM	IMC	MACC	0	1	0	<b>2.533</b>
	TCM	IM	MACC	0	0	0	<b>1.567</b>
	TCM	IM	MACC pipe	0	0	0	1.567
	TCM	IMC	MACC	0	0	0	<b>1.567</b>
	TCM	IMC	MACC	1	0	0	1.567
	TCM	IMC	MACC pipe	0	0	0	1.5
	TCM	IM	Fabric	0	0	0	<b>1.067</b>
	TCM	I	n/a	1	1	0	<b>1.067</b>
	TCM	IMC	Fabric	0	0	0	<b>1.033</b>
	TCM	I	n/a	0	1	0	0.967
	TCM	I	n/a	0	0	0	0.533
	TCM	IC	n/a	0	0	0	0.533
	TCM	I	n/a	0	0	1	0.533
	TCM	I	n/a	1	0	0	<b>0.533</b>

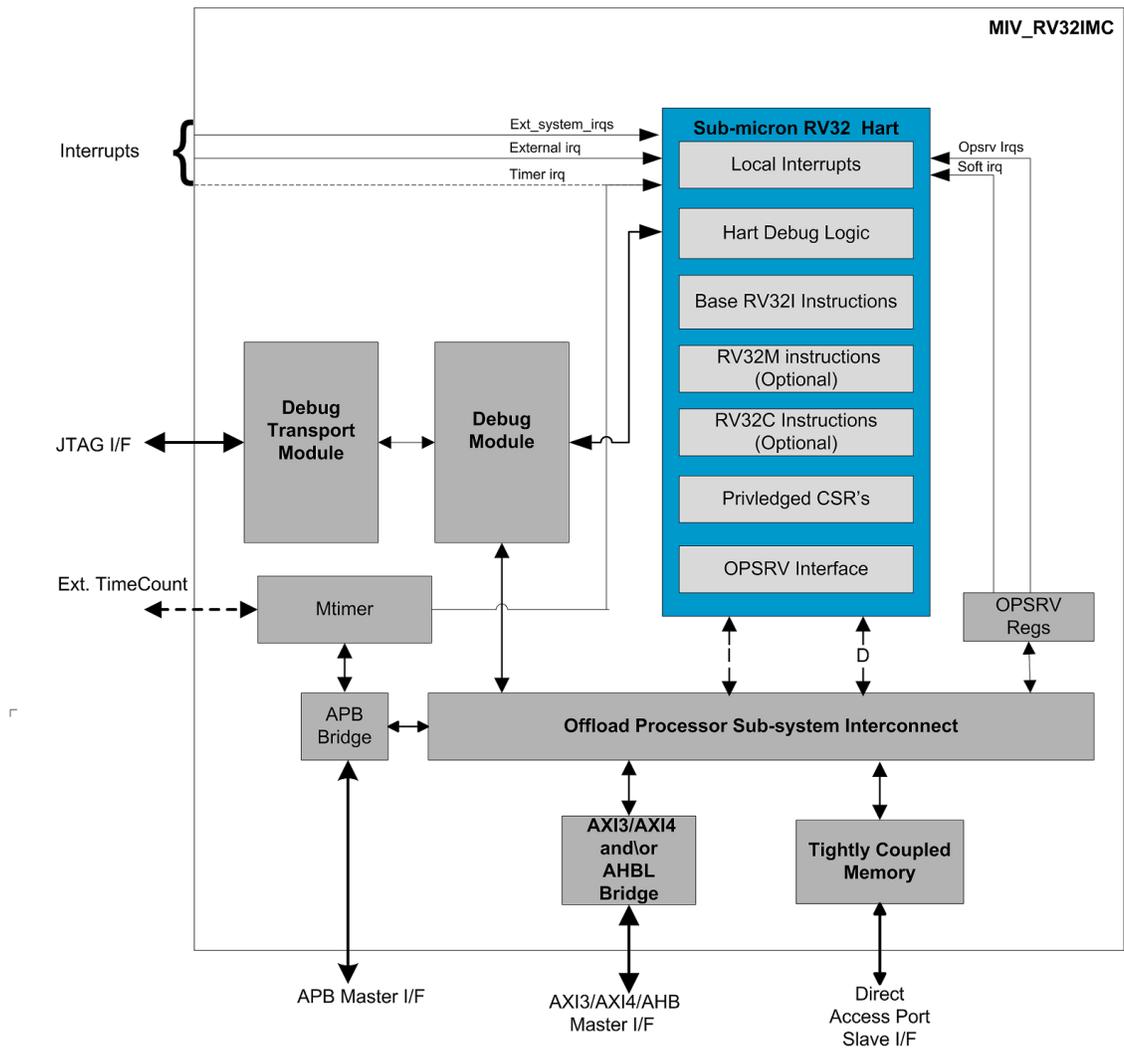
Benchmarks	Memory Location	RV32	Multiplier	Reg Fwd	Reg GPRs	ECC	Coremark/MHz
Interface bench-marks	AHB	IMC	MACC	0	0	0	0.467
	AHB	IM	MACC	0	0	0	0.433
	AHB	IC	n/a	0	0	0	0.2
	AHB	I	n/a	0	1	0	0.2
	AHB	I	n/a	1	1	0	0.2
	AHB	I	n/a	0	0	0	0.167
	AHB	I	n/a	1	0	0	0.167
	AXI	IM	MACC	0	0	0	0.4
	AXI	IMC	MACC	0	0	0	0.4
	AXI	IC	n/a	0	0	0	0.167
	AXI	I	n/a	0	1	0	0.167
	AXI	I	n/a	0	0	0	0.133
	AXI	I	n/a	1	0	0	0.133

### 3 Functional Description

#### 3.1 MIV\_RV32IMC Architecture

The core architecture comprises of a RV32IMC four stage pipelined processor unit integrated with an Offload Processor Subsystem for RISC-V (OPSRV). The OPSRV consists of a system interconnect with a JTAG Debug Module, System MTimer, Tightly Coupled Memory (TCM) with a Direct Access Port (DAP- Slave APB), AHB\AXI\APB master interfaces, and OPSRV registers. The following figure shows the block level architecture of MIV\_RV32IMC device.

Figure 1 • MIV\_RV32IMC Block Diagram



The following table lists the key features of the core.

**Table 11 • MIV\_RV32IMC Architecture features**

Feature	Value	Units	Notes
ISA support	RV32IMC		Base RV32I, optional multiply and divide and optional compressed extensions
Harts	1		Submicron RV32IMC
Reset vector	Configurable		Word aligned address 0x1000_0000 and above available
Interrupts	13		External, Software and Timer interrupts, six optional external interrupts, and ECC interrupts. Vectored interrupts supported.
Timers/Counters	1		An MTIME block is available to generate a time value and periodic interrupts
Bus interfaces	AHB/AXI3/AXI4/APB		Optional AHB, AXI3/AXI4, and APB
JTAG debug transport address width	7	Bits	
Local memories	1		Width of TCM start and end address determines the size of the local memory
Local memory access	1		Optional Direct Access Port (DAP) provides slave APB access to TCM

## 3.2 Hart

The MIV\_RV32IMC hart is based on the RISC-V Instruction Set Architecture (ISA). The hart supports the RISC-V standard RV32 Integer (I), Multiply (M), and Compressed (C) ISA. It also supports the machine-mode privileged architecture and debug mode.

The hart is a four stage pipelined submicron processor, which has been designed to be highly configurable for use in Microchip FPGAs. It is designed to be used as a standalone or auxiliary processor within FPGA designs. The hart contains the base RISC-V Integer ISA extension. Optionally, the RISC-V M ISA extension adds hardware multiply and divide instructions. Optionally, the RISC-V C ISA extension adds the compressed instruction set.

## 3.3 Memory System

The core is non-cached. The Tightly Coupled Memory (TCM) is available as for instruction and data storage. A range of system peripherals are accessed across AXI (AXI4/AXI3), AHB, and APB bus interfaces.

## 3.4 Interrupts

The RISC-V external interrupt is available for use as a top-level input to the core. Six optional external interrupts can also be enabled at the top level for use as external interrupts. The RISC-V software interrupt is available and can be accessed through the OPSRV register. The timer interrupt can be exposed to the top level or connected internally to a compare register that can be accessed through software, and generates interrupts at a fixed interval. There is an OPSRV register interrupt available that signals TCM, ECC, or AXI write errors. Interrupts can be configured in vectored or non-vectored mode when the core is being configured to allow for a defined vector for each interrupt, if required. Interrupts are positive edge triggered.

### 3.5 Debug Support via JTAG

The core includes support for an external debugger using a JTAG port. This v0.13.2 debug implementation is abstract command based and uses system bus access to write to memory. The core does not support the use of a program buffer. The following debug features are provided:

- Hart can be halted and resumed
- All hart registers (including CSRs) can be read/written
- Memory can be accessed
- Binary files can be downloaded to memory
- Hart can be debugged from the very first instruction executed
- Debug can perform single-step operations and can execute one instruction at a time
- A RISC-V hart can be halted when a software breakpoint instruction is executed

### 3.6 External Interfaces

The core supports three optional external interfaces: AHB, APB, and AXI (AXI3/AXI4). Each interface has its own address space mapped at compile time. The address spaces may not overlap.

The core can boot from a word aligned address range, within the configurator specified address space, by setting the `RESET_VECTOR` and modifying the linker scripts for the firmware project. This address can be half word aligned, if C extension is used.

### 3.7 Tightly Coupled Memory

The core supports Tightly Coupled Memory (TCM). The size of the memory is defined by the start and end address of the TCM. This memory can be booted by setting the `RESET_VECTOR` to the address of the TCM. The TCM can also be initialized at power on or programmed through the DAP interface.

### 3.8 Direct Access Port

A Direct Access Port is available over an APB slave interface. This allows reading and writing to the TCM from an external source before the core is brought out of reset. It is recommended that the address widths for the TCM and the DAP are of the same size to avoid memory read/write violations.

### 3.9 Clocks

The system clock frequency should be chosen to meet design timing requirements with clock constraints applied. The tables in [Device Resource Utilization and Performance](#) section list the upper clock frequency obtained for a specified device from each supported FPGA family, whilst meeting timing requirements for the configurations defined. Sequential logic within the core is driven on the positive clock edge.

When the debug option is enabled, the JTAG debug signals are made available at the top level. The JTAG has a clock signal TCK whose characteristics are determined by the connected JTAG debugger. It is advised that the applied TCK frequency should not be greater than one-seventh of the system clock frequency and remain within the maximum frequency permitted for the JTAG probe in use. The TCK should have clock constraints applied. For more information, see the [Design Constraints](#) section.

### 3.10 Resets

The `RESETN` is an active low hard reset, which resets everything within the core. An external reset synchronizer is required (for more information, see the [Reset Synchronization](#) section). In many cases, the synchronizer is integrated within a family specific reset core, for example, `Core_Reset_PF`.

The EXT\_RESETN is an active low reset output. This is fed through from the RESETN and also driven from the debug module during a debug session to allow a system reset through the debugger.

There is an internal CPU Soft Reset feature accessible through software. For more information, see the [Table 46 • opsrv\\_soft\\_reg \(0x6020\)](#) table in the OPSRV Register section.

The TRST is an active high reset signal for the JTAG Test Access Port (TAP).

## 4 Interface

### 4.1 Configuration Parameters

The following table lists the parameters (Verilog) or generics (VHDL) for configuring the RTL code of the core.

**Table 12 • MIV\_RV32IMC Parameters and Generics Descriptions**

Name	Range	Default Value	Description
RESET_VECTOR_ADDR_1	0x1000-0xFFFF	0x8000	This is the address the processor will start executing from after a reset. This address is byte aligned.
RESET_VECTOR_ADDR_0	0x0000-0xFFFC	0x0	
DEBUGGER	0 or 1	1	JTAG Debugger 0: Disable 1: Enabled
AXI_MASTER_TYPE	0 to 2	0	AXI Master Type 0: None 1: AXI3 2: AXI4
AXI_SLAVE_MIRROR	0 or 1	0	AXI Slave Mirror 0: None 1: AXI Slave Mirror <b>Note:</b> This parameter is only used when an AXI Master is selected
AXI_START_ADDR_1	0x1000-0xFFFF	0x6000	This is the AXI start address. <code>AXI_START_ADDR_1</code> and <code>AXI_START_ADDR_0</code> represent the upper and lower 16 bits of the address respectively.
AXI_START_ADDR_0	0x0000-0xFFFF	0x0	
AXI_END_ADDR_1	0x1000-0xFFFF	0x6FFF	This is the AXI end address. <code>AXI_END_ADDR_1</code> and <code>AXI_END_ADDR_0</code> represent the upper and lower 16 bits of the address respectively.
AXI_END_ADDR_0	0xFFFF-0xFFFC	0xFFFF	
AHB_MASTER_TYPE	0 or 1	1	AHB Master Type 0: None 1: AHB-Lite
AHB_SLAVE_MIRROR	0 or 1	0	AHB Slave Mirror 0: None 1: AXI Slave Mirror <b>Note:</b> This parameter is only used when an AHB Master is selected
AHB_START_ADDR_1	0x1000-0xFFFF	0x8000	This is the AHB start address. <code>AHB_START_ADDR_1</code> and <code>AHB_START_ADDR_0</code> represent the upper and lower 16 bits of the address respectively.
AHB_START_ADDR_0	0x0000-0xFFFF	0x0	
AHB_END_ADDR_1	0x1000-0xFFFF	0x8FFF	This is the AHB end address. <code>AHB_END_ADDR_1</code> and <code>AHB_END_ADDR_0</code> represent the upper and lower 16 bits of the address respectively.
AHB_END_ADDR_0	0xFFFF-0xFFFC	0xFFFF	

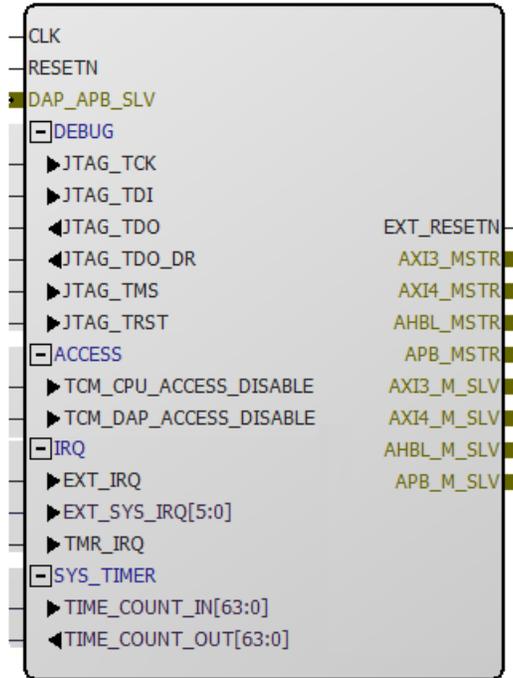
Name	Range	Default Value	Description
APB_MASTER_TYPE	0 or 1	1	APB Master Type 0: None 1: APB3
APB_SLAVE_MIRROR	0 or 1	0	AHB Slave Mirror 0: None 1: AXI Slave Mirror <b>Note:</b> This parameter is only used when an AHB Master is selected
APB_START_ADDR_1	0x1000-0xFFFF	0x7000	This is the APB start address. APB_START_ADDR_1 and APB_START_ADDR_0 represent the upper and lower 16 bits of the address respectfully.
APB_START_ADDR_0	0x0000-0xFFFF	0x0	
APB_END_ADDR_1	0x1000-0xFFFF	0x7FFF	This is the APB end address. APB_END_ADDR_1 and APB_END_ADDR_0 represent the upper and lower 16 bits of the address respectfully.
APB_END_ADDR_0	0xFFFF-0xFFFC	0xFFFF	
TCM_PRESENT	0 or 1	0	TCM Present 0: Disabled 1: Enabled
TCM_START_ADDR_1	0x1000-0xFFFF	0x4000	This is the TCM start address. TCM_START_ADDR_1 and TCM_START_ADDR_0 represent the upper and lower 16 bits of the address respectfully.
TCM_START_ADDR_0	0x0000-0xFFFF	0x0	
TCM_END_ADDR_1	0x1000-0xFFFF	0x4000	This is the TCM end address. TCM_END_ADDR_1 and TCM_END_ADDR_0 represent the upper and lower 16 bits of the address respectfully.
TCM_END_ADDR_0	0xFFFF-0xFFFC	0x4000	
TCM_DAP_PRESENT	0 or 1	0	TCM DAP Present 0: Disabled 1: Enabled
DAP_START_ADDR_1	0x1000-0xFFFF	0x4000	This is the TCM DAP start address. DAP_START_ADDR_1 and DAP_START_ADDR_0 represent the upper and lower 16 bits of the address respectfully.
DAP_START_ADDR_0	0x0000-0xFFFF	0x0	
DAP_END_ADDR_1	0x1000-0xFFFF	0x4000	This is the TCM DAP end address. DAP_END_ADDR_1 and DAP_END_ADDR_0 represent the upper and lower 16 bits of the address respectfully.
DAP_END_ADDR_0	0xFFFF-0xFFFC	0x4000	
GEN_DECODE_RV32	0 to 3	3	RISCV ISA Extension Select 0: I 1: IM 2: IC 3: IMC
GEN_MUL_TYPE	0 to 2	0	Multiplier Type 0: Fabric 1: MACC (Non-Pipelined) 2: MACC (Pipelined)
VECTORED_INTERRUPTS	0 or 1	1	Vectored Interrupts 0: Disabled

Name	Range	Default Value	Description
			1: Enabled
MTVEC_OFFSET	0x0 to 0xFFFF	0x34	MTVEC Offset as defined in the RISC-V HAL (currently 0x100).
NUM_EXT_IRQS	0 to 6	6	Number of external interrupts
FWD_REGS	0 or 1	0	Forwarding Registers 0: Disabled 1: Enabled
ECC_ENABLE	0 or 1	0	ECC 0: Disabled 1: Enabled
INTERNAL_MTIME	0 or 1	1	Internal MTIME 0: Disabled 1: Enabled
MTIME_PRESCALER	0 to 65535	100	The MTIME_PRESCALER integer value divided by the CLK frequency derives an MTIME time base given by the equation:  $\text{MTIME timebase} = \frac{\text{MTIME\_PRESCALER}}{\text{CLK (Hz)}}$
INTERNAL_MTIME_IRQ	0 or 1	1	Internal MTIME 0: Disabled 1: Enabled
GPR_REGS	0 or 1	0	GPR Registers 0: Disabled 1: Enabled

## 4.2 I/O Signals

The following figure shows all the I/O signals for the core.

**Figure 2 • MIV\_RV32IMC Full I/O View**



The following table lists the MIV\_RV32IMC I/O signal description.

**Table 13 • MIV\_RV32IMC I/O Signal Description**

Port Name	Width	Direction	Description
<b>Global Signals</b>			
CLK	1	In	System clock. All other I/Os are synchronous to this clock.
RESETN	1	In	Synchronized reset signal. This signal is active low.
EXT_RESETN	1	Out	External system reset, active low. Driven by RESETN and Debugger system reset (debug mode).
<b>JTAG Interface Signals</b>			
JTAG_TDI	1	In	Test Data In (TDI). This signal is used by the JTAG device for downloading and debugging programs. Sampled on the rising edge of TCK.
JTAG_TCK	1	In	Test Clock (TCK). This signal is used by the JTAG device for downloading and debugging programs.
JTAG_TMS	1	In	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.

Port Name	Width	Direction	Description
JTAG_TRST	1	In	Test Reset (TRST). This is an optional signal used to reset the TAP controllers state machine. This signal is active high.
JTAG_TDO	1	Out	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.
JTAG_TDO_DR	1	Out	Drive Test Data Out (DRV_TDO). This signal is used to drive a tri-state buffer.
<b>Interrupt Signals</b>			
EXT_IRQ	1	In	External interrupt from peripheral source. An active high level based interrupt signal. Tie this input low if unused.
EXT_SYS_IRQ	6	In	Optional External System Interrupts. This signal is active high. Tie any unused inputs low.
TMR_IRQ	1	In	A Timer interrupt input is exposed when the internal MTIME IRQ parameter is not selected in the GUI. This in an active high level based interrupt. Tie this input low if unused.
<b>System Time Signals</b>			
TIME_COUNT_IN	64	In	External system timer count
TIME_COUNT_OUT	64	Out	Internal system timer count
<b>TCM Access Signals</b>			
TCM_CPU_ACCESS_DISABLE	1	In	When asserted, CPU access to the TCM is disabled.
TCM_DAP_ACCESS_DISABLE	1	In	When asserted, DAP access to the TCM is disabled.
<b>APB Master Interface</b>			
APB_MSTR_PADDR	32	Out	APB Master Interface. The address range is 0x1000_0000 to 0xFFFF_FFFF. This interface can also be configured as a mirrored slave through the GUI.
APB_MSTR_PSEL	1	Out	
APB_MSTR_PENABLE	1	Out	
APB_MSTR_PWRITE	1	Out	
APB_MSTR_PRDATA	32	In	
APB_MSTR_PWDATA	32	Out	
APB_MSTR_PREADY	1	In	
APB_MSTR_PSLVERR	1	In	
<b>AHB Master Interface</b>			
AHB_MSTR_HMASTLOCK	1	Out	AHB Master Interface. The address range is 0x1000_0000 to 0xFFFF_FFFF. This interface can also be configured as a mirrored slave through the GUI.
AHB_MSTR_HTRANS	2	Out	
AHB_MSTR_HWRITE	1	Out	

Port Name	Width	Direction	Description
AHB_MSTR_HADDR	32	Out	
AHB_MSTR_HSIZE	3	Out	
AHB_MSTR_HBURST	3	Out	
AHB_MSTR_HPROT	4	Out	
AHB_MSTR_HWDATA	32	Out	
AHB_MSTR_HREADY	1	In	
AHB_MSTR_HRESP	1	In	
AHB_MSTR_HRDATA	32	In	
AHB_MSTR_HSEL	1	Out	HSEL only used when AHB_SLAVE_MIRROR is set to 1.
<b>AXI Master Interface</b>			
AXI_MSTR_AWREADY	1	Out	AXI (AXI3/AXI4) Master Interface. The address range is 0x1000_0000 to 0xFFFF_FFFF. This interface can also be configured as a mirrored slave through the GUI.
AXI_MSTR_AWVALID	1	Out	
AXI_MSTR_AWID	1	Out	
AXI_MSTR_AWADDR	32	Out	
AXI_MSTR_AWLEN	4	Out	
AXI_MSTR_AWSIZE	3	Out	
AXI_MSTR_AWBURST	2	Out	
AXI_MSTR_AWLOCK	1	Out	
AXI_MSTR_AWCACHE	4	Out	
AXI_MSTR_AWPROT	3	Out	
AXI_MSTR_WREADY	1	in	
AXI_MSTR_WVALID	1	Out	
AXI_MSTR_WID	1	Out	
AXI_MSTR_WDATA	32	Out	
AXI_MSTR_WSTRB	4	Out	
AXI_MSTR_WLAST	1	Out	
AXI_MSTR_BREADY	1	Out	
AXI_MSTR_BVALID	1	in	
AXI_MSTR_BID	1	in	

Port Name	Width	Direction	Description
AXI_MSTR_BRESP	2	in	
AXI_MSTR_BUSER	1	in	
AXI_MSTR_ARREADY	1	in	
AXI_MSTR_ARVALID	1	Out	
AXI_MSTR_ARID	1	Out	
AXI_MSTR_ARADDR	32	Out	
AXI_MSTR_ARLEN	4	Out	
AXI_MSTR_ARSIZE	3	Out	
AXI_MSTR_ARBURST	2	Out	
AXI_MSTR_ARLOCK	1	Out	
AXI_MSTR_ARCACHE	4	Out	
AXI_MSTR_ARPROT	3	Out	
AXI_MSTR_RREADY	1	Out	
AXI_MSTR_RVALID	1	Out	
AXI_MSTR_RID	1	Out	
AXI_MSTR_RDATA	32	Out	
AXI_MSTR_RRESP	2	in	
AXI_MSTR_RLAST	1	in	
<b>APB Slave Interface (DAP)</b>			
DAP_APB_SLV_PADDR	32	In	APB Slave Interface (DAP). The address range is 0x1000_0000 to 0xFFFF_FFFF.
DAP_APB_SLV_PSEL	1	In	
DAP_APB_SLV_PENABLE	1	In	
DAP_APB_SLV_PWRITE	1	In	
DAP_APB_SLV_PRDATA	32	Out	
DAP_APB_SLV_PWDATA	32	In	
DAP_APB_SLV_PREADY	1	Out	
DAP_APB_SLV_PSLVERR	1	Out	

## 5 Programmer's Model

This core implements the RISC-V Integer extension with optional support for the Multiplication and Division extension (M) and the Compressed extension (C). Multiplication and Division provides hardware support for these operations and Compressed allows for a subset of the Integer instructions to be encoded as 16-bit instructions as opposed to 32-bit instructions.

The M improves operating performance of the processor at the expense of area and speed, while C allows for reduced code size with additional area.

### 5.1 Processor Operating States

**Machine Mode:** This core can be run in RISC-V machine mode and is the standard operating state for the core. In this mode, the 32-bit I and M instructions can be executed along with the 16-bit C instructions.

**Debug Mode:** The core enters the debug mode when debugging using the JTAG interface.

### 5.2 Reset Operation

Out of reset or as a result of a CPU soft reset, PC takes on the value of the Reset Vector Address (RVA) and begins executing code from this address.

### 5.3 Data Types

This core supports the following data types:

- 32-bit words
- 16-bit halfwords
- 8-bit bytes

Instructions can be encoded as 32-bit words for all extensions and a subset of the Integer instructions can be encoded as 16-bit words when the C extension is included.

### 5.4 General Purpose Registers

The following table lists the 32-bit RISC-V General Purpose Registers (GPRs) available in the core.

**Table 14 • General Purpose Registers**

Register	ABI Name	Description
x0	zero	Hardwired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5–x7	t0–t2	Temporary registers
x8	s0/fp	Saved register/Frame pointer

Register	ABI Name	Description
x9	s1	Saved Register
x10–x11	a0–a1	Function arguments/Return values
x12–x17	a2–a7	Function arguments
x18–x27	s2–s11	Saved registers
x28–x31	t3–t6	Temporary registers

## 5.5 Machine Control and Status Registers

As the core only supports machine mode, it only needs to implement a small subset of the machine mode registers defined in the RISC-V privileged architecture. The remaining registers and bits of registers are addressable, and are hard coded as defined by the privileged architecture specification.

The following table lists the implemented CSRs.

**Table 15 • mvendorid CSR (0xF11)**

<b>Bits</b>	31:0
<b>Field</b>	Vendor ID
<b>R/W</b>	RO
<b>Reset</b>	Preset value = l_submicron_vendorid

The vendor ID CSR reads the value defined by the l\_submicron\_vendorid constant configured at build time.

**Table 16 • marchid CSR (0xF12)**

<b>Bits</b>	31:0
<b>Field</b>	Architecture ID
<b>R/W</b>	RO
<b>Reset</b>	Preset value = l_submicron_marchid

**Table 17 • mimpid CSR (0xF13)**

<b>Bits</b>	31:0
<b>Field</b>	Implementation ID
<b>R/W</b>	RO
<b>Reset</b>	Preset value = l_submicron_mimpid

**Table 18 • mhartid CSR (0xF14)**

<b>Bits</b>	31:0
<b>Field</b>	Hart ID
<b>R/W</b>	RO
<b>Reset</b>	Preset value = I_hart_id

**Table 19 • mstatus CSR (0x300)**

<b>Bits</b>	31	30:23	22	21	20	19	18	17
<b>Field</b>	SD	-	TSR	TW	TVM	MXR	SUM	MPRV
<b>R/W</b>	RO	WPRI	RO	RO	RO	R RO	RO	RO
<b>Reset</b>	-	-	-	-	-	-	-	-

<b>Bits</b>	16:15	14:13	12:11	10:9	8	7	6	5	4	3	2	1	0
<b>Field</b>	XS	FS	MPP	-	SPP	MPIE	-	SPIE	UPIE	MIE	-	SIE	UIE
<b>R/W</b>	RO	RO	RO	WPRI	RO	RW	WPRI	RW	RO	RO	WPRI	RO	RO
<b>Reset</b>	-	-	3	-	-	-	-	-	-	0	-	-	-

This core only supports the machine mode. It only implements the mie and mpie bits as actual read-write register bits. The MPP is always hardwired to 3 as it can only ever be machine mode. The remaining bits are tied off to 0.

When the status register is read using the supervisor, or user mode alias (sstatus(0x100) and ustatus(0x000) respectively), they can still be accessed without an illegal instruction exception, as these registers are accessible from machine mode. However, only the supervisor and user mode bits are available in sstatus, and only the user bits are available in ustatus. Therefore, in both cases all bits read as 0 and not be writable.

The MPP is always in the machine mode (that is, 3) as the core only supports machine mode. The MIE is architecturally defined to reset to 0. All other bits do not have a defined reset value.

The XS is currently not implemented. The SD bit reflects the state of the XS filed (the core does not support floating point instructions. Therefore, FS is always == 0).

**Table 20 • misa CSR (0x301)**

<b>Bits</b>	31:30	29:26	25:0
<b>Field</b>	Base ISA	-	Extension
<b>R/W</b>	RO	WPRI	RO

Base ISA is RV32 = 2'b01.

According to the RISC-V architecture, misa may optionally be implemented as a RW register to allow the supported base ISA and extensions to change. However, in this core it is implemented as hardwired read-only, as these decisions are build time configuration options.

**Table 21 • misa CSR Extension Bit Description**

Bit	Character	Description	Submicron implementation
0	A	Atomic extension	0
1	B	Bit manipulation extension	0
2	C	Compressed extension	Configuration option
3	D	Double-precision floating-point extension	0
4	E	RV32E base ISA	0
5	F	Single-precision floating-point extension	0
6	G	Additional standard extensions present	0
7	H	Hypervisor mode implemented	0
8	I	RV32I/64I/128I base ISA	1
9	J	Dynamically translated languages extension	0
10	K	Reserved	0
11	L	Tentatively reserved for Decimal Floating-Point extension	0
12	M	Integer Multiply/Divide extension	Configuration option
13	N	User-level interrupts supported	0
14	O	Reserved	0
15	P	Packed-SIMD extension	0
16	Q	Quad-precision floating-point extension	0
17	R	Reserved	0
18	S	Supervisor mode implemented	0
19	T	Tentatively reserved for Transactional Memory extension	0
20	U	User mode implemented	0
21	V	Vector extension	0
22	W	Reserved	0
23	X	Non-standard extensions present	0
24	Y	Reserved	0
25	Z	Reserved	1 (for support of Zicsr and ZFencei)

**Table 22 • mie CSR (0x304) Machine Interrupt Enable Register**

Bits	31	30	29:24	23:18	17	16
Field	-	OPSRV_IRQ_IE	MSYS_EIE	-	MGECIE	MGEUIE
R/W	RO	RW	RW	WPRI	RW	RW
Reset	-	-	-	-	-	-

Bits	15:12	11	10	9	8	7	6	5	4	3	2	1	0
Field	-	MEIE	-	-	-	MTIE	-	-	-	MSIE	-	-	-
R/W	WPRI	RW	WPRI	RO	RO	RW	WPRI	RO	RO	RW	WPRI	RO	RO
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-

**Table 23 • mip CSR (0x344) Machine Interrupt Pending Register**

Bits	31	30	29:24	23:18	17	16
Field	-	OPSRV_IRQ_IE	MSYS_EIP	-	MGECIP	MGEUIP
R/W	RO	RW	RW	WPRI	RO	RO
Reset	-	-	-	-	-	-

Bits	15:12	11	10	9	8	7	6	5	4	3	2	1	0
Field	-	MEIP	-	-	-	MTIP	-	-	-	MSIP	-	-	-
R/W	WPRI	RO	WPRI	RO	RO	RO	WPRI	RO	RO	RO	WPRI	RO	RO
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-

**Table 24 • Interrupt Bit Description**

Name	Description
MSI	Software interrupt
MTI	Timer interrupt
MEI	External interrupt
MGEUI	GPR ECC uncorrectable error. Exists if ECC is enabled, otherwise 0.
MGECI	GPR ECC correctable error. Exists if ECC is enabled, otherwise 0.
MSYS_EI[5:0]	System external interrupts. Exists if external interrupts are enabled.

When the **ie** register is read using the supervisor, or user mode alias (**sie**(0x104) and **uie**(0x004) respectively, and similarly for **ip** (**sip**(0x144) and **uip** (0x044)), they can still be accessed without an illegal instruction exception, as these registers are accessible from machine mode. However, only the supervisor and user

mode bits are available in sie/sip, and only the user bits in uie/uips. Therefore, in both cases all bits read as 0 and are not writable.

**Table 25 • mtvec CSR (0x305)**

Bits	31:2	1:0
Field	Exception vector base	mode
R/W	If vectored interrupts are disabled (Direct Mode), the mtvec exception base vector is a read/write RW register. If vectored interrupts are enabled (Vector Mode), the mtvec exception base vector is a read only register, where the value is defined by the reset vector address + the MTVEC offset value provided in the configurator.	RW RO
Reset	In Direct Mode, the reset value is not defined. In Vector Mode, the reset value is the reset vector address + the MTVEC offset value provided in the configurator.	2'b00 2'b01

**Table 26 • mepc CSR (0x341)**

Bits	31:0
Field	Machine exception program counter
R/W	RW
Reset	-

mepc[0] is always 0. Therefore, it is hardwired 0.

**Table 27 • mcause CSR (0x342)**

Bits	31	30:0
Field	Interrupt	Exception code
R/W	RW	RW
Reset	0	0

The exception codes are as defined in the RISC-V privileged architecture. The core implements the architecturally defined trap codes and additional custom trap codes as described in the following table. Therefore, only code[4:0] are implemented as register bits, the remaining bits are hardwired to 0. The italicized interrupts and exceptions in the following table are non-standard custom traps defined.

**Table 28 • mcause Exception Codes**

Interrupt	Exception code	Description
1	3	MSI [Highest priority]
1	7	MTI
1	11	MEI
1	16	<i>MGEUI</i>
1	17	<i>MGECI</i>
1	24–29	<i>MSYS_EI0-MSYS_EI5</i>

Interrupt	Exception code	Description
1	30	<i>OPSRV IRQ</i>
0	3	Trigger breakpoint
0	2	Illegal instruction
0	0	Instruction address misalign
0	11	M env-call
0	3	Breakpoint
0	6	Store address misaligned
0	4	Load address misaligned
0	24	<i>Instruction fetch read bus error</i>
0	25	<i>Instruction fetch read parity error</i>
0	26	<i>Data load bus error</i>
0	27	<i>Data load parity error [Lowest priority]</i>

The **mcause** register is reset to 0 following hard or soft reset.

**Table 29 • mtval CSR (0x343)**

<b>Bits</b>	<b>31:0</b>
<b>Field</b>	mtval
<b>R/W</b>	RW
<b>Reset</b>	x

**Table 30 • MTVAL Value Following Trap Taken**

Interrupt/exception cause	MTVAL
MSI [Highest priority]	0
MTI	0
MEI	0
<i>MGEUI</i>	<i>0</i>
<i>MGECI</i>	<i>0</i>
<i>MICNT</i>	<i>0</i>
<i>MCCNT</i>	<i>0</i>
<i>MHCNT</i>	<i>0</i>

Interrupt/exception cause	MTVAL
<i>MSYS_EI0-MSYS_EI7</i>	0
Trigger breakpoint	Program counter of the instruction retiring
Illegal instruction	Encoding of instruction retiring
Instruction address misalign	Program counter of the instruction retiring
M env-call	0
Breakpoint	Program counter of the instruction retiring
Store address misaligned	Store address of faulting access
Load address misaligned	Load address of faulting access
<i>Instruction fetch read bus error</i>	<i>Program counter of the instruction retiring</i>
<i>Instruction fetch read parity error</i>	<i>Program counter of the instruction retiring</i>
<i>Data load bus error</i>	<i>Load address of faulting access</i>
<i>Data load parity error [Lowest priority]</i>	<i>Load address of faulting access</i>

**Table 31 • mscratch CSR (0x340)**

<b>Bits</b>	<b>31:0</b>
<b>Field</b>	mscratch
<b>R/W</b>	RW
<b>Reset</b>	-

**Table 32 • time CSR (0xC01)**

<b>Bits</b>	<b>31:0</b>
<b>Field</b>	time[31:0]
<b>R/W</b>	RO
<b>Reset</b>	x

**Table 33 • timeh CSR (0xC81)**

<b>Bits</b>	<b>31:0</b>
<b>Field</b>	timeh[63:32]
<b>R/W</b>	RO
<b>Reset</b>	x

Time is a read-only user CSR.

## 5.6 Debug Module

The MIV\_RV32IMC debug module is implemented in compliance with the RISC-V External Debug Support specification v0.13.2. The debug module consists of the following three main blocks:

1. Debug Transfer Module with an interface module DMI
2. Debug Unit—abstract command based with System Bus Access
3. Hart Debug Logic—debug CSR's and halt/run logic

### 5.6.1 Debug Transport Module

An external debugger communicates with the core's debug sub-system through a JTAG interface with a Test Access Port (TAP) controller. The TAP Controller Instruction Register has a length of five bits. Upon reset, its value is 0x01, selecting the IDCODE instruction. The following table lists the full instruction set.

**Table 34 • TAP Controller Instructions**

IR code	Mnemonic	Full name
0x00	-	Reserved
0x01	IDCODE	IDCODE
0x02–0x03	-	Reserved
0x04	-	Reserved
0x05–0x0F	-	Reserved
0x10	DTMCS	DTM Control and Status
0x11	DMI_ACCESS (DMI)	Debug Module Interface Access
0x12–0x1E	-	Reserved
0x1F	BYPASS	BYPASS instruction

Internal connection between TAP and DM is a form of serial scan interface. Source and destination of the TAP controller scan interface are in different clock domains. The TAP runs in the JTAG's TCK clock domain, whereas, the DM are in the system clock domain. Therefore, the TAP controller scan interface must pass through a clock synchronization process.

Using the Debug Module Interface (DMI), the debug module (DM) exposes a standard register interface to the core's debug features:

- Run control of the core's single hart
- Access to its internal registers (GPRs and CSRs)
- Access to its memory space

## 5.6.2 Debug Unit

The debug module implementation is abstract command based for GPR\CSR access and uses system bus access to perform read/write operations to memory locations. The following table lists the registers implemented in Debug Module.

**Table 35 • Debug Module Registers**

Address	Mnemonic	Full name
0x00–0x03	-	Reserved
0x04	DATA0	Abstract Data 0
0x05	-	Abstract Data 1, not implemented
0x06–0x0F	-	Reserved
0x10	DMCONTROL	Debug Module Control
0x11	DMSTATUS	Debug Module Status
0x12	-	Reserved
0x13	HALTSUM1	Halt Summary 1 (single bit)
0x14–0x15	-	Reserved
0x16	ABSTRACTCS	Abstract Control and Status
0x17	COMMAND	Abstract Command
0x18	-	Abstract Command Autoexec, not implemented
0x19	-	Configuration String Pointer, not implemented
0x1A–0x37	-	Reserved
0x38	SBCS	System Bus Access Control and Status
0x39	SBADDRESS	System Bus Address [31:0]
0x3A–0x3B	-	Reserved
0x3C	SBDATA	System Bus Data [31:0]
0x3D–0x3F	-	Reserved
0x40	HALTSUM0	Halt Summary 0 (single bit)
0x41–0x7F	-	Reserved

### 5.6.3 Hart Debug Logic

The Submicron hart implements the halt\run logic and required debug CSR registers.

Two debug CSRs: DCSR, and DPC are accessible using abstract debug commands when in Debug Mode.

**Table 36 • Debug Control and Status Registers**

Address	Mnemonic	Full name
0x7B0	DCSR	Debug Control and Status
0x7B1	DPC	Debug PC
0x7B2 – 0x7BF	-	Reserved

#### 5.6.3.1 Debug Control and Status CSR (DCSR–0x7B0)

**Table 37 • Debug Control and Status Registers**

Bits	Name	Access	Reset Value	Description
31:28	xdebugver	RO	4	The field's value (4) indicates that debug support exists as described in the RISC-V Debug Spec version 0.13.
27:16	rsrv3	RO	0	Reserved for future use.
15	ebreakm	RW	0	Encoding: 0 - ebreak instructions in M-mode behave as described in the Privileged Spec. 1 - ebreak instructions in M-mode enter Debug Mode (Soft Breakpoint).
14:12	rsrv2	RO	0	Reserved for future use. (Supervisor/User modes not supported)
11	stepie	RW	0	Encoding: 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.
10:9	rsrv1	RO	0	Reserved for future use. (System Counter/Timer halting not supported)
8:6	cause	RO	0	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. Encoding: 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); Other values are reserved for future use.
5:3	rsrv0	RO	0	Reserved for future use (mprven\nmp not supported).
2	step	RW	0	When set and not in debug mode, the hart only executes a single instruction and then enters debug mode. If the instruction does not complete due to an exception, the hart immediately enters the debug mode before

Bits	Name	Access	Reset Value	Description
				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	RW	3	Contains the privilege level the hart was operating in when debug mode was entered. Mi-V debug has the field hardwired to 3. Only machine mode is supported.

### 5.6.3.2 Debug Program Counter CSR (DPC–0x7B1)

Upon entry to debug mode, DPC is updated with the virtual address of the next instruction to be executed. The following table lists the behavior.

**Table 38 • DPC CSR Address Behavior**

Cause	Virtual address in DPC
ebreak	Address of the ebreak instruction.
Single Step	Address of the instruction that would be executed next, that is, PC + 4 or the destination PC, if jumps/branches taken.
Halt Request	Address of the next instruction to be executed at the time that debug mode was entered.

When resumed, the hart's PC is updated to the virtual address stored in DPC. A debugger may write DPC to change where the hart resumes.

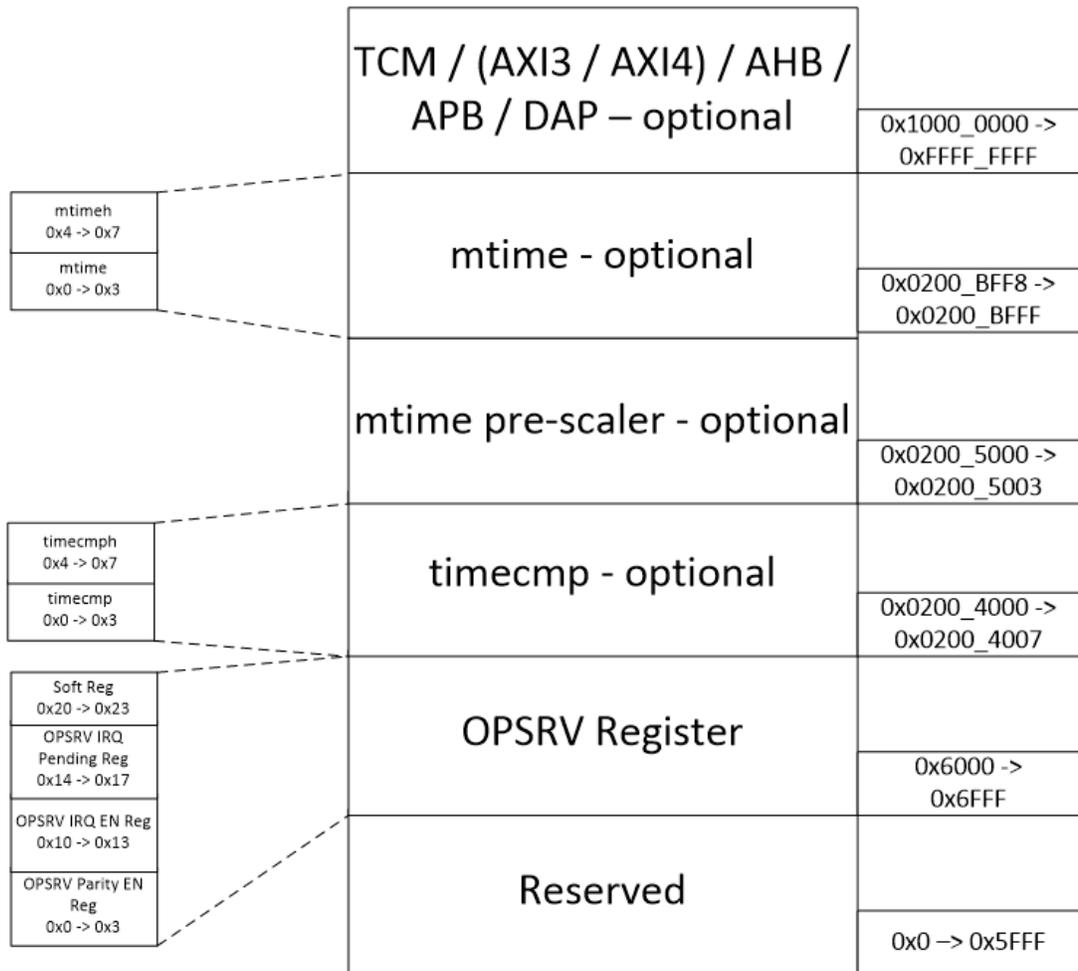
**Table 39 • DPC CSR Register**

Bits	Name	Access	Reset Value	Description
31:0	dpc	RW	-	Field contains the debug PC value.

## 5.7 Memory Map

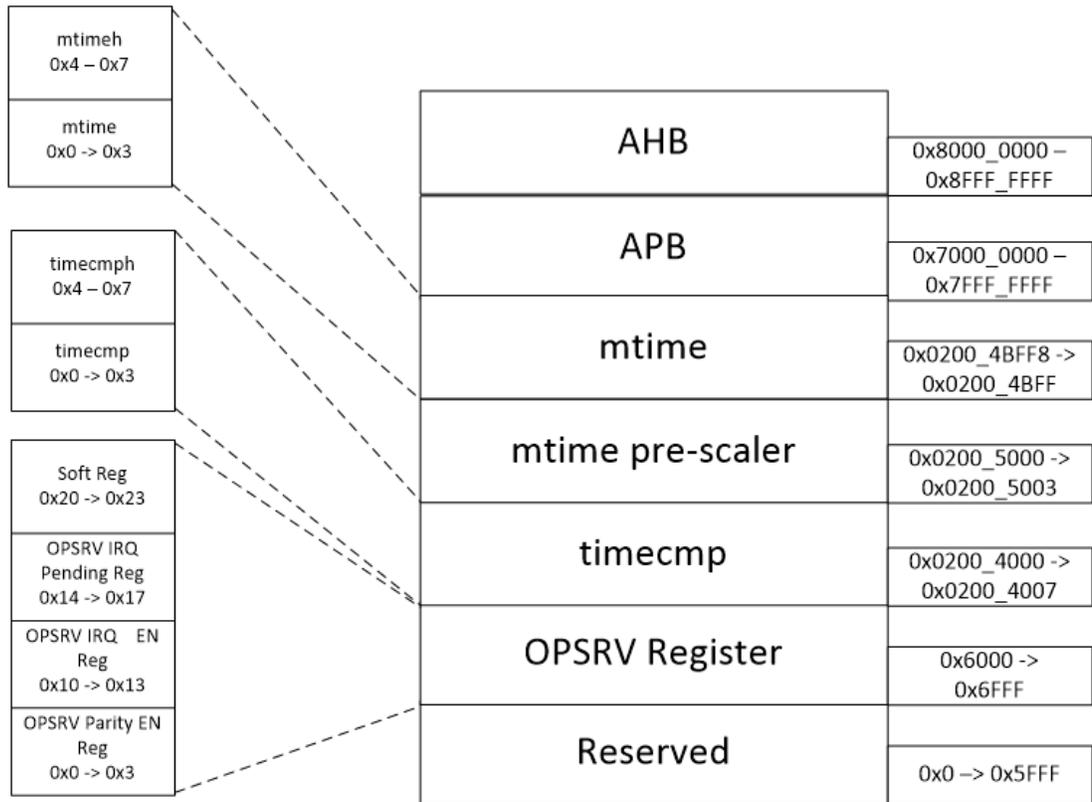
The memory map of the core is highly configurable. The GPRs, CSRs, and debug registers are contained in the reserved range. The OPSRV Register is memory mapped to 0x6000. The optional 64-bit timecmp register is mapped to 0x0200\_4000. The mtime pre-scaler is mapped to 0x0200\_5000. The 64-bit mtime register is mapped to 0x0200\_BFF8. The following figure shows the memory map.

**Figure 3 • Memory Map**



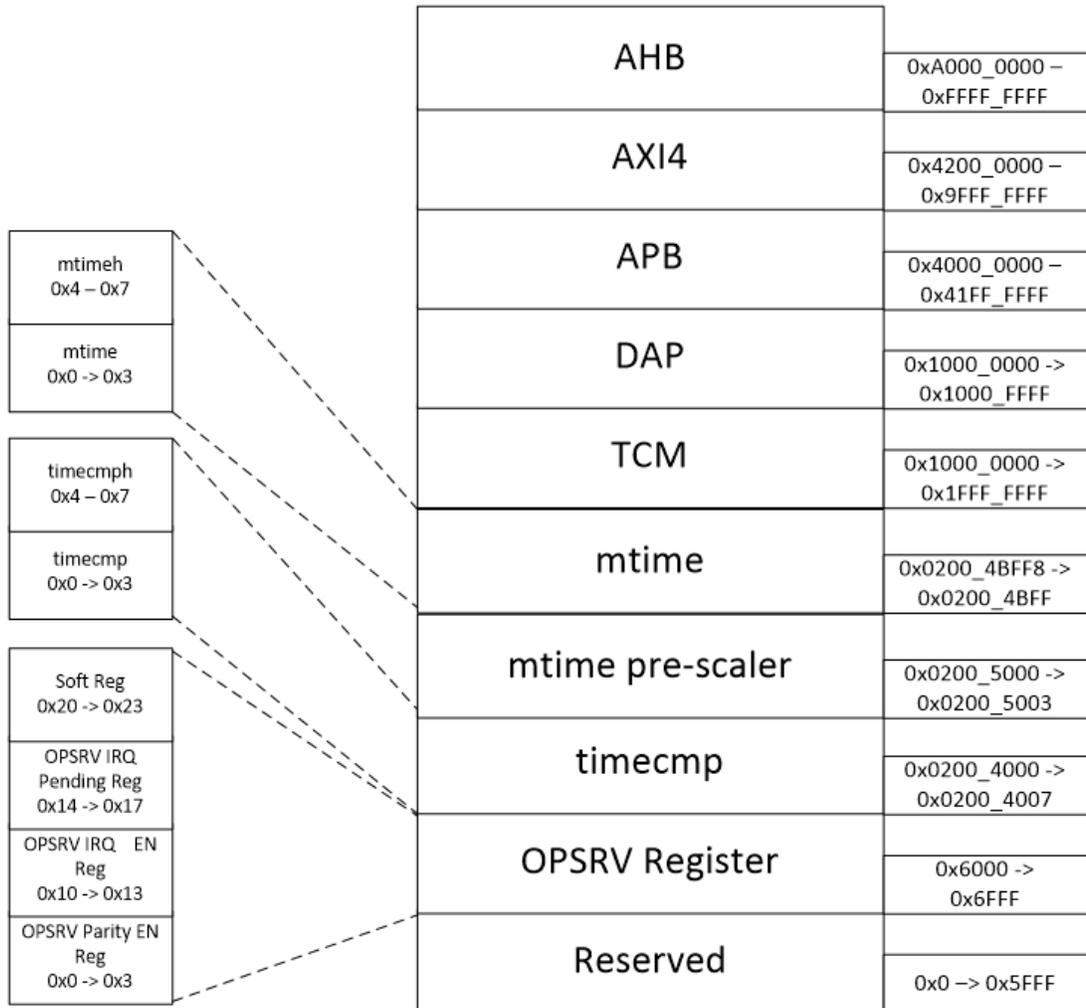
The following figure shows the default memory map for the core.

**Figure 4 • Default Memory Map**



The following figure shows a memory map using the full accessible range.

**Figure 5 • Example System Memory Map**



## 5.8 Subsystem Restrictions

Very few restrictions are placed on the configuration of this core. Interface slots and TCM must have a start address  $\geq 0x1000_0000$  and cannot have overlapping start and end addresses. These rules are enforced by the core configurator. The DAP can have an address anywhere in this accessible range, but the TCM must be within this address space to be accessed.

In vectored mode, the mtvec is restricted to being a read only register. This is different to the normal mtvec operation expected by the Microchip RISC-V HAL, and its user guide should be consulted on how to properly deal with this configuration.

## 5.9 Exceptions

The core handles all exceptions. The core can be configured to handle interrupts in a vectored or non-vectored mode, with faults causing the PC to jump to mtvec regardless of vectored or non-vectored mode.

### 5.9.1 Vectored and Non-Vectored Interrupts

In vectored mode, each interrupt causes the PC to jump to a defined address relative to the `mtvec` value defined at build time. In vectored mode, the `mtvec` is defined as a read only register. It is populated with the value of the reset vector + the `MTVEC OFFSET` value set in the core configurator. This must be accommodated for when writing and linking software. In non-vectored mode, all exceptions cause the PC to jump to the `mtvec` address (a R/W register in non-vectored mode so that the `mtvec` can be dynamically changed using a `csrw` or `csrwi` instruction) and the exception cause can be determined by checking the value in the `mcause` register.

### 5.9.2 Nested Interrupts

Nested interrupts are supported and interrupts can be re-enabled during an ISR by setting the `mie` bit in the `mstatus` register. This bit is automatically un-set when taking an interrupt, and is re-enabled when an `mret` instruction is executed.

### 5.9.3 Available Interrupts

The following interrupts are available to generate exceptions:

- External
- Software
- Timer
- GPR ECC Uncorrectable
- GPR ECC Correctable
- Custom External x 6
- OPSRV Register

The OPSRV register interrupt is triggered by any of the following interrupts present in the OPSRV register:

- TCM ECC Correctable Error
- TCM ECC Uncorrectable Error
- AXI Write Error

The external interrupt is available to be used as an input to the core. The software interrupt is internally connected to bit[1] of the OPSRV soft register and writing a 1 to this bit causes a soft interrupt. The timer interrupt can be internally connected to a counter and time compare register, which can be used to generate periodic interrupts, or a counter input can be made available as a top-level input to the core and the compare register. The six custom external interrupts are available as inputs to the core. The OPSRV register interrupt is internally connected to the OR'd outputs of the interrupts available in the OPSRV pending register.

### 5.9.4 Interrupt Handling

When an exception is generated in non-vectored mode, the PC jumps to `mtvec`. Once there, the register states can be pushed to the stack. The cause of the exception determined, if it is not a fault, can be handled and register states restored. An `mret` instruction sets the PC to the value of the next instruction following when the exception was taken and re-enables interrupts. This ISR is used in the Microchip RISC-V HAL.

### 5.9.5 Vectored Interrupt Offsets and Exception Priorities

When an exception is generated in vectored mode, the PC jumps to the defined address for each interrupt source with a defined priority. The following table lists the defined priorities.

**Table 40 • Vectored Interrupt Offsets and Exception Priorities**

Priority	Exception Source	Start Address from MTVEC	End Address from MTVEC
Highest	Software interrupt	0xC	0x1B
	Timer interrupt	0x1C	0x2B
	External interrupt	0x2C	0x3B
	GPR ECC uncorrectable error interrupt	0x3C	0x4B
	GPR ECC correctable error interrupt	0x4C	0x5B
	Custom external interrupt 0	0x60	0x63
	Custom external interrupt 1	0x64	0x67
	Custom external interrupt 2	0x68	0x6B
	Custom external interrupt 3	0x6C	0x6F
	Custom external interrupt 4	0x70	0x73
	Custom external interrupt 5	0x74	0x77
	OPSRV interrupt	0x78	0x8B
Lowest	Fault	0x0	0xB

### 5.9.6 OPSRV Register Interrupts

The OPSRV register interrupt handler manages the following interrupts.

- TCM ECC correctable error
- TCM ECC uncorrectable error
- AXI Write response error

When any of the OPSRV register interrupts are triggered, the OPSRV interrupt to the core asserts and remains asserted until the interrupt is handled.

Each interrupt has an enable bit in the OPSRV Register Interrupt Enable register addressed at 0x6010:

**Table 41 • OPSRV Interrupt Enable Register**

Bit	Interrupt Enabled
0	TCM ECC correctable error
1	TCM ECC uncorrectable error
4	AXI write response error

Each interrupt has a pending bit in the OPSRV Register Interrupt Pending Register addressed at 0x6014:

**Table 42 • OPSRV Interrupt Pending Register**

Bit	Interrupt Pending
0	TCM ECC correctable error
1	TCM ECC uncorrectable error
4	AXI write response error

The interrupt pending register of the OPSRV register should be read to determine which interrupt occurred causing the OPSRV register interrupt to assert. Interrupts from the OPSRV register can be cleared by writing to the corresponding interrupt pending bit in the interrupt pending register. The priority with which OPSRV register interrupts are serviced is defined by the software.

The soft interrupt is also contained within the OPSRV register, but not managed in the same way as the TCM, ECC, or AXI interrupts. It is bit[1] in the OPSRV soft register addressed at 0x6020. Writing a 1 to this bit causes a soft interrupt to occur. It can be cleared by writing a 0 to the bit.

## 5.10 OPSRV Register

The offload processor subsystem interfaces with the hart and provides an interconnect for the interfaces. The DAP, TCM, and OPSRV registers are accessed by the hart. The following table lists the several additional configuration registers of the OPSRV register for features of the core.

**Table 43 • opsrv\_cfg (0x6000)**

Bits	31:1	0
Field		opsrv_parity_en
R/W	RO	RW
Reset		0

Setting the `opsrv_parity_en` bit enables parity checking on TCM and interface transactions. Data in the TCM must be written with parity when this feature is enabled, and bus parity must be generated by peripherals connected to the core. In this release of the core, the parity enable register has been tied to 0, as parity is not being supported.

**Table 44 • opsrv\_irq\_en (0x6010)**

Bits	31:5	4	3:2	1	0
Field		AXI write response err irq en		TCM ECC uncorrectable err irq en	TCM ECC correctable err irq en
R/W	RO	RW	RO	RW	RW
Reset		0	0		0

This register contains the enable bits for each of the OPSRV interrupts. Setting any of the available interrupt bits allows that interrupt to assert `opsv_irq`. Machine interrupts will still need to be enabled.

**Table 45 • opsv\_irq\_pend (0x6014)**

Bits	31:5	4	3:2	1	0
Field		AXI write response err irq pend		TCM ECC uncorrectable err irq pend	TCM ECC correctable err irq pend
R/W	RO	RW	RO	RW	RW
Reset		0	0		0

This register contains the pending bits for each of the OPSRV interrupts. When any of these bits assert, the `opsv_irq` is triggered, if the corresponding enable bit is set. Writing a 1 to any set bit clears it.

**Table 46 • opsv\_soft\_reg (0x6020)**

Bits	31:3	2	1	0
Field		core_gpr_ded_reset	soft_irq	soft_rst
R/W	RO	RW	RW	RW
Reset	0	0	0	0

Setting the `soft_irq` bit in this register causes a soft interrupt to be triggered in the core. The machine interrupts and the software interrupt should be enabled for this interrupt to be taken. Writing 0 to this bit clears it. Setting the `soft_rst` bit causes a CPU soft reset. This bit unsets after 1 clock cycle to prevent lockup.

The `core_gpr_ded_reset_reg` is set, when the core has reset due to GPR DED error when ECC is enabled. It can be cleared by writing 0 to the bit.

## 5.11 MTIME

The MTIME is an optional block for this core. It contains a compare register (`mtimecmp`), which sources a time count from an external source (`TIME_COUNT`) or an internal counter (`mtime`).

**Table 47 • mtimecmp (0x200\_4000)**

Bits	31:0
Field	mtimecmp
R/W	RW
Reset	FFFF_FFFF

**Table 48 • mtimecmph (0x200\_4004)**

Bits	31:0
Field	mtimecmph
R/W	RW
Reset	FFFF_FFFF

The `mtimecmp` and `mtimecmph` registers contain the time value that triggers a timer interrupt to the core. When `mtime` is greater than or equal to this value, an interrupt is generated to the core. The machine interrupts and timer interrupts should be enabled for this interrupt to be taken.

**Table 49 • `mtime_prescaler (0x200_5000)`**

Bits	31:0
Field	<code>mtime_prescaler</code>
R/W	RO
Reset	Value set from core configurator

The `mtime` prescaler register is read only and populated with the `mtime` prescaler value set in the core configurator. This register exists to indicate the configured prescaler value for `mtime` relative to the system clock.

**Table 50 • `mtime (0x200_BFF8)`**

Bits	31:0
Field	<code>mtime</code>
R/W	RW
Reset	0

**Table 51 • `mtimeh (0x200_BFFC)`**

Bits	31:0
Field	<code>mtimeh</code>
R/W	RW
Reset	0

The `mtime` and `mtimeh` registers contain the time value.

The `mtimecmp` register is 64-bits wide and initialized to a value of `0xFFFF_FFFF_FFFF_FFFF`. The `mtime` register is also a 64-bit value and is initialized to `0x0`. It is incremented by an internal counter, if selected from the configurator or from a 64-bit `TIME_COUNT` input to the core. Once `mtime`  $\geq$  `mtimecmp`, a timer interrupt is generated to the core. This interrupt is serviced only if machine interrupts and the timer interrupt are enabled.

The lower 32 bits of the `mtimecmp` register can be written as follows:

C:

```
int *time_cmp_addr;
time_cmp_addr = 0x02004000;
*time_cmp_addr = time_count; // time_count is the value written to the timecmp register
```

The upper 32 bits of the `mtimecmp` (`mtimecmph`) register can be written as follows:

C:

```
int *time_cmph_addr;
time_cmph_addr = 0x02004004;
*time_cmph_addr = time_count; // time_count is the value written to the timecmp register
```

**Note:**

On first use of the system timer interrupt, the `timecmp` and `timecmph` reset values are `0xFFFF_FFFF`. These values generate an interrupt and should be set by the user as well.

The `mtime` register can also be written by using the following code.

The lower 32 bits of the `mtime` register can be written as follows:

C:

```
int *time_addr;
time_addr = 0x02004BFF8;
*time_addr = time_count; // time_count is the value written to the time register
```

The upper 32 bits of the `mtime` (`mtimeh`) register can be written as follows:

C:

```
int *time_addr;
timeh_addr = 0x02004BFFC;
*timeh_addr = time_count; // time_count is the value written to the time register
```

**Note:**

Ensure that an overflow situation should not occur when using the `mtime` compare register. For example, with a prescaler of 50 and a 50 MHz system clock, it will take approximately 500,000 years to reach overflow. However, if a lower prescaler value is set, this will occur sooner, or, software should have appropriate handling in place to prevent an overflow situation.

For example:

`mtime` value == `0xFFFF_FFFF_FFFF_FFF0`

An interrupt is required in `0xFFFF` cycles of `mtime`. This occurs when:

`mtime` value == `0x0000_0000_0000_FFF0`

Setting an `mtimecmp` value == `0x0000_0000_0000_FFF0` causes an immediate interrupt, as `mtime` is greater than `mtimecmp`. To resolve this, `mtime` and `mtimeh` should first be written with `0x0` and an `mtimecmp` value of `0xFFFF` should be set.

## 5.12 ECC

Error Correcting Codes (ECC) can be enabled for the core through the configurator. They encode parity with data in RAM and can correct single bit errors and detect double bit errors, Single Error Correct Double Error Detect (SECEDED).

If RAM based GPRs are used (default) and ECC is enabled, a fabric encoder and decoder will be instantiated for the RAM. If the TCM is used, an encoder and decoder will also be instantiated for the memory when ECC is enabled.

When ECC is enabled for the GPRs, the core are held in soft reset for several cycles on start-up while the GPRs are initialized to 0. This prevents erroneous SECEDED errors on uninitialized RAM.

If a double bit error is detected from a GPR, the core automatically initiates a soft reset and the GPRs are reinitialized before start-up. This prevents bad data from the GPR being used. In the event of a DED, soft reset bit[2] of the OPSRV Soft register sets to indicate that the core has recovered from a double bit error and this bit can be cleared by writing to it. The individual machine ECC interrupts have to be enabled for an ECC error to be handled, but the soft reset will occur on a DED regardless of the interrupts being enabled.

When ECC is enabled for the TCM, it is not initialized. This process should be undertaken in software before enabling the TCM ECC interrupts in the OPSRV Register.

Sample code to initialize the TCM to 0:

C:

```
volatile uint32_t *tcm_addr;
tcm_addr = 0x40000000; // Memory mapped start address of TCM
uint32_t tcm_end_addr = 0x40002000; // Memory mapped end address of TCM

while (tcm_addr != tcm_end_addr){ // loop until end is reached
    *tcm_addr = 0x0; // write 0 to memory location
    tcm_addr = tcm_addr + 0x1; // increment pointer
}
```

Sample code to scrub the TCM and generate parity for initialized memory:

C:

```
volatile uint32_t *tcm_addr;
tcm_addr = 0x40000000; // Memory mapped start address of TCM
uint32_t tcm_end_addr = 0x40002000; // Memory mapped end address of TCM

while (tcm_addr != tcm_end_addr){
    *tcm_addr = *tcm_addr;
    tcm_addr = tcm_addr + 0x1;
}
```

Programming the TCM through the DAP also generates parity if ECC is enabled.

## 6 Tool Flow

### 6.1 License

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

#### 6.1.1 RTL

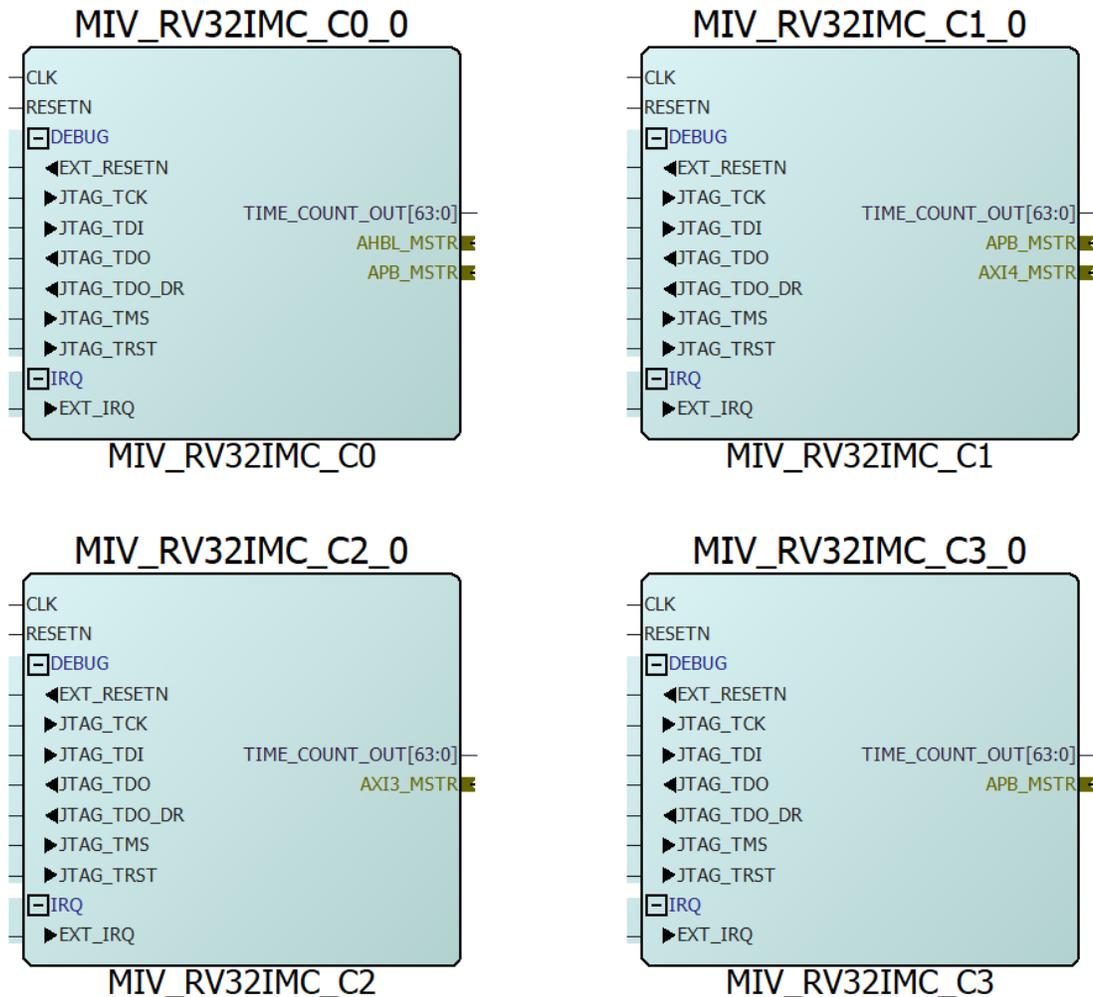
Complete Verilog source code is provided for the core. The core can be readily instantiated within SmartDesign. Simulation, Synthesis, and Layout can be performed within Libero SoC.

### 6.2 SmartDesign

The MIV\_RV32IMC is pre installed in the SmartDesign IP deployment design environment. The core is available from the Libero catalog.

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

**Figure 6 • SmartDesign MIV\_RV32IMC Instance Views**

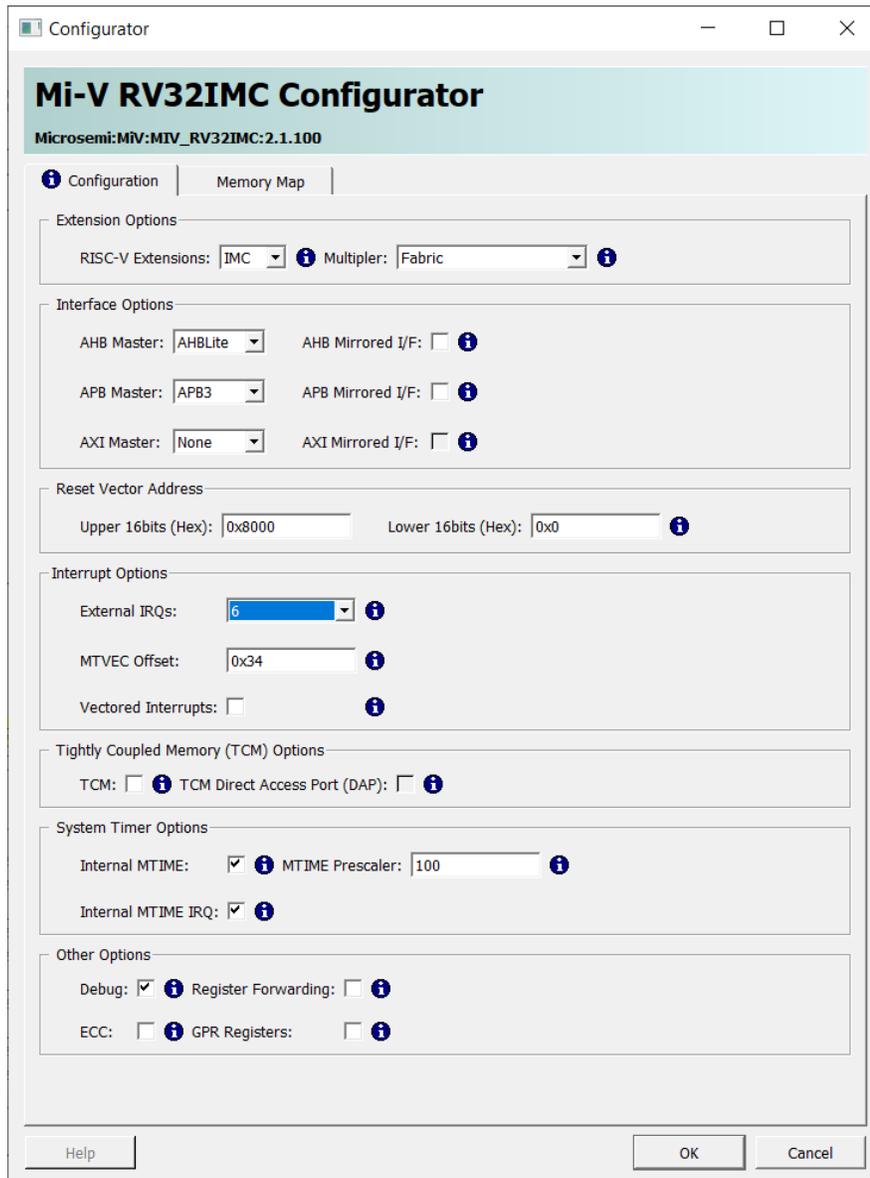


## 6.3 Configuring MIV\_RV32IMC

The core is configured using the configuration GUI within SmartDesign, as shown in the following figure.

**Note:** Leading zeros are suppressed. For example, 0x8000 0000 is displayed as 0x8000 0x0. The reset vector is word-aligned.

**Figure 7 • Configuration GUI for MIV\_RV32IMC in SmartDesign**



### 6.3.1 Extension Options

Under the Extension Options heading, the core can be configured to use a combination of the following RISC-V standard extensions:

- **I**—Base Integer instruction set
- **M**—Multiply and Divide instruction set (optional)
- **C**—Compressed instruction set (optional)

The core Multiplier can be set to either use the Fabric Multiplier or the MACC multiplier:

- **Fabric**—multiply operations complete in 32 clock cycles.
- **MACC (Non-Pipelined)** —multiply operations complete in one clock cycle. However, the maximum operating frequency of the processor decreases in comparison to the fabric multiplier.
- **MACC (Pipelined)**—multiply operations complete in typically one extra clock cycle. The maximum operating frequency of the processor increases in comparison to the non-pipelined MACC multiplier.

### 6.3.2 Interface Options

Under the Interface Options heading, the core can be configured to use the following bus interfaces:

- AHB Master: AHBLite
- APB Master: APB 3.0
- AXI Master: AXI3 or AXI4

The option to configure these interfaces as mirrored slaves is available. This option allows a single slave component (for example, RAM or UART) to be connected directly to the interface at the start address, without the need for a bus master.

### 6.3.3 Reset Vector Address

Under the Reset Vector Address heading, the reset vector address of the core can be configured. The default boot address is 0x8000\_0000. The code that runs from initialized memory must be built using the correct linker script. For example, in the RISC-V HAL, which can be generated from the Firmware catalog, there are two example linker scripts: `Microsemi-riscv-ram.ld` and `Microsemi-riscv-igloo2.ld`. `Microsemi-riscv-ram.ld` is configured for a single memory at address 0x8000\_0000 in Random Access Memory (RAM), such as DDR or LSRAM. The `Microsemi-riscv-igloo2.ld` is configured to use NVM (ROM) at address 0x6000\_0000 and RAM at 0x8000\_0000. To boot from initialized memory on power-up, the reset vector and RAM start address in the linker script must match, otherwise, the core will not boot as expected.

### 6.3.4 Interrupt Options

Under the Interrupt Options heading, up to six optional external interrupts are available alongside the standard external interrupt.

### 6.3.5 Tightly Coupled Memory (TCM) Options

Under the Tightly Coupled Memory (TCM) Options heading, the option to enable TCM along with the additional option to enable a TCM Direct Access Point (DAP) is available. This option allows reading and writing to the TCM by an external core over the APB Interface. The TCM depth is determined by the address space allocated in the Interface Memory Mapping configuration.

### 6.3.6 Other Options

Under the Other Options heading, the option to enable register forwarding is available. This option increases the area and decreases the maximum operating frequency of the system, while increasing the core performance.

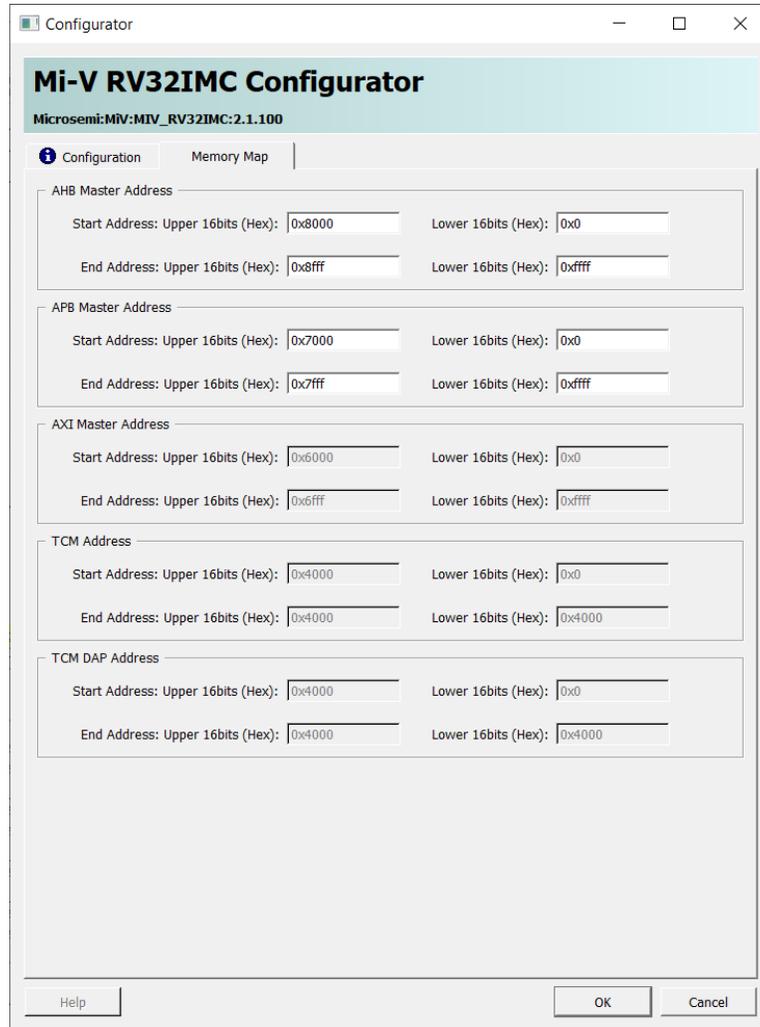
Option to enable ECC (Error Correcting Code) is available, which instantiates fabric ECC encoding and decoding logic for the TCM and GPRs, if they are RAM based.

Debug option to enable or disable the JTAG debug feature is also available.

### 6.3.7 Memory Map Tab

The address range for each interface can be configured through the core configurator's memory map tab. The following figure shows the default address ranges used for each interface.

**Figure 8 • Memory Map GUI for MIV\_RV32IMC in SmartDesign**



In the event of an overlap conflict between address ranges, the configurator displays a warning and does not generate the core until the conflict is resolved.

## 6.4 Debugging

The CoreJTAGDebug v3.1.100 or later, is used to enable debugging of MIV\_RV32IMC. This is available in the Libero Catalog.

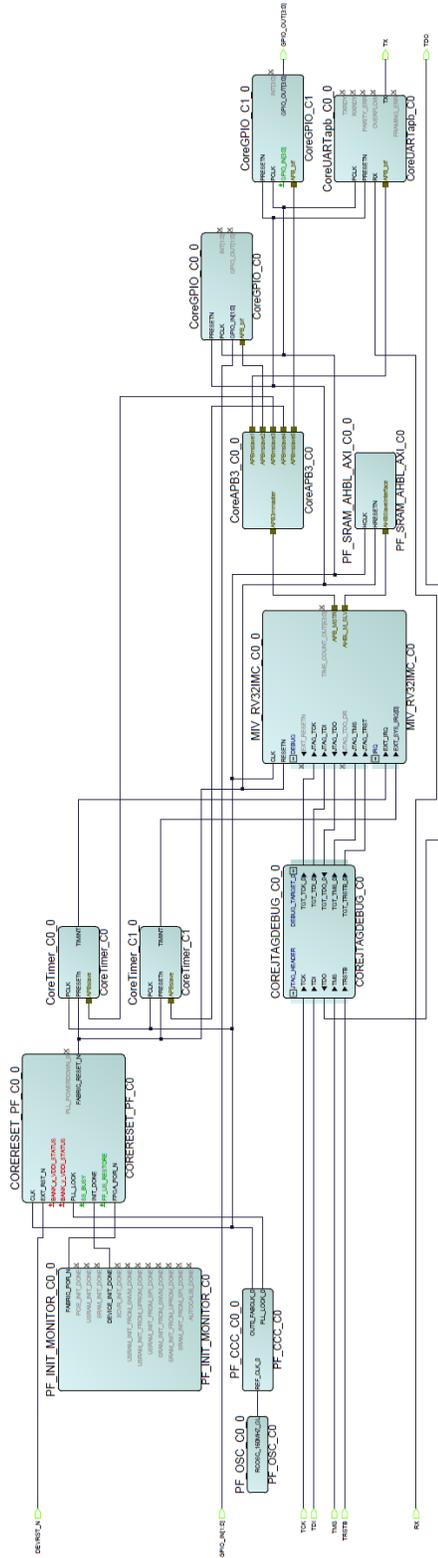
## 6.5 Simulation Flows

The user testbench for MIV\_RV32IMC is not included in this release.

The MIV\_RV32IMC RTL can be simulated using a standard Libero generated HDL testbench. An example subsystem is shown in the following figure. A hex file found in the Debug or Release folders generated by SoftConsole is needed for this method. When the hex file is generated, remove the first line before importing

into memory. The RESET\_VECTOR of the MIV core is set to 0x8000\_0000. Therefore, it boots from the LSRAM\_0. Using this design, the MIV core can be simulated.

Figure 9 • Example Subsystem



## 6.6 Synthesis in Libero

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

## 6.7 Place-and-Route in Libero

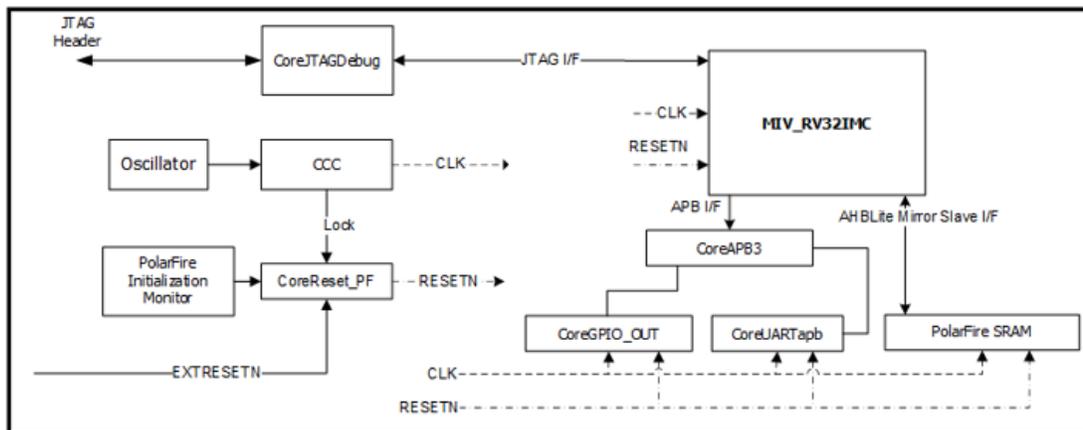
After the design is synthesized, run the compilation and the place-and-route tools. Click the **Layout** icon in Libero SoC to invoke designer.

## 7 System Integration

### 7.1 PolarFire Example System

The following figure shows the block diagram containing the basic building blocks to start using the core in a simple system.

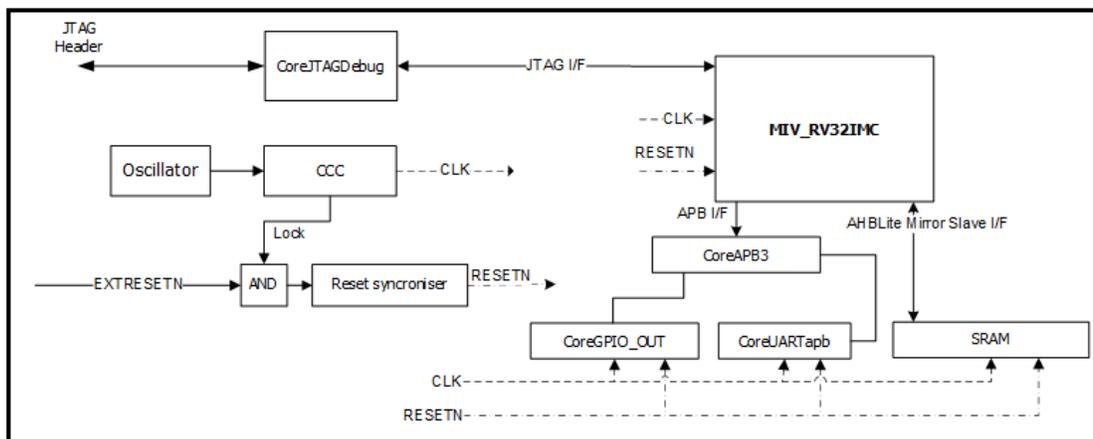
Figure 10 • MIV\_RV32IMC: An Example System



### 7.2 RTG4/SF2/IG2 Example System

The following figure shows the block diagram containing the basic building blocks to start using the core in a simple system for the RTG4/SF2/IG2.

Figure 11 • MIV\_RV32IMC: An Example System for RTG4/SF2/IGL2



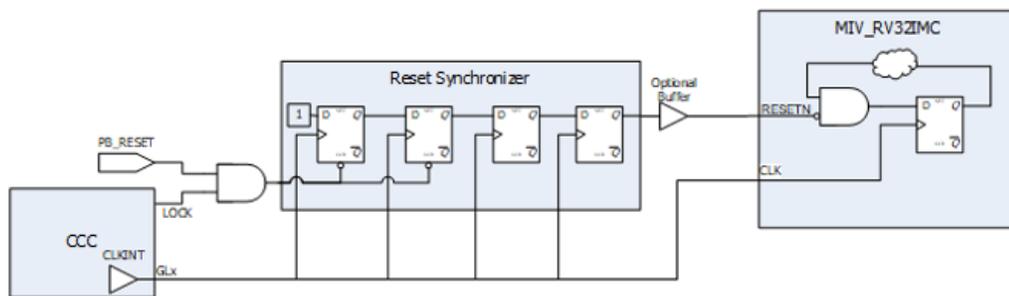
**Note:** The difference between the PolarFire and the other design is the use of the PolarFire Initialization Monitor and CoreReset\_PF. The HDL reset synchroniser acts as the CoreReset\_PF, as this IP is only available on PolarFire. For other families, the reset synchroniser is used. The HDL code for the reset synchroniser is available in the following section.

## 7.3 Reset Synchronization

### 7.3.1 RESETN

All sequential elements are clocked by the *CLK* signal, which requires that resets to employ a synchronous reset topology. As most designs source *CLK* from a CCC/PLL, it is a common practice to AND the *LOCK* output of the CCC with the push button reset (*PB\_RESET*), to generate the *RESETN* input for core. However, this results in the reset being de-asserted when the *CLK* comes up. Therefore, the reset de-assertion is not clocked through the sequential reset elements and goes unnoticed most commonly leading to the processor locking-up. To guarantee that the *RESETN* de-assertion is seen by all sequential elements, a reset synchronizer is required on the *RESETN* input, as shown in the *MIV\_RV32IMC: An Example System for RTG4/SF2/IGL2* figure.

Figure 12 • *RESETN* Reset Synchronization



The following Verilog code snippet implements the reset synchronizer block shown in the *SmartDesign MIV\_RV32IMC Instance Views* figure. The function of this block is to make the reset assertion and de-assertion synchronous to *CLK* while guaranteeing that the reset is asserted for one or more *CLK* cycles to the core, to ensure that it is registered by all sequential elements.

```

module reset_synchronizer (
    input clock,
    input reset,
    output reset_sync
);
    reg [1:0] sync_deassert_reg;
    reg [1:0] sync_assert_reg;

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

```

Perform the following steps to include this synchronizer in your Libero design.

1. Select **Create HDL** from the **Design Flow** tab in your Libero project.
2. In the pop-up window, name the HDL file accordingly and select **Verilog** as the HDL type.
3. Uncheck the option to initialize file with standard template.

4. Copy and paste the Verilog code above into this file and save the changes.
5. Build the Design Hierarchy, and then from the **Design Hierarchy** tab, drag and drop the file into the SmartDesign containing the core instance and connect the pins as shown in the preceding figure.

### 7.3.2 TRST

No reset synchronization is required on this reset input, as all sequential elements in the debug logic in the core use an asynchronous reset topology.

## 8 Design Constraints

Designs containing core require the application of the following constraints in the design flow to allow timing-driven Place-And-Route (PAR) and timing analysis to be performed on the design. The procedure for adding the required constraints in the enhanced constraints flow in Libero SoC is as follows:

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

Assuming the the system clock `SYS_CLK`, which used to clock the core, is sourced from a CCC. Select **Derive Constraints** to automatically create a constraints file containing the CCC 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 CCC configuration in the design, update the contents of this file by clicking **Derive Constraints** again. 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 name it. Design constraints other than the system clock source derived constraints can be entered in this blank SDC file. Keeping derived and manually added constraints in separate SDC files allows the **Derive Constraints** stage to be re-performed, if changes are made to the CCC 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 a FlashPro, with a maximum frequency of 30 MHz supported by FlashPro5. Populate the following constraint with the TCK values and paste it into the blank SDC file:

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

As TCK is in a clock domain independent and asynchronous to the `SYS_CLK`, a `set_clock_groups` constraint is also required. A `set_clock_groups` constraint is also required.

```
set_clock_groups -name {group_name} \
-asynchronous \
-group [ get_clocks {...../SYS_CLK} ] \
-group [ get_clocks { TCK } ]
```

For example, the following constraints need to be applied for a design that uses a TCK frequency of 6 MHz with a CCC generated system clock called `OUT0`:

```
#Constraining the JTAG clock to 6 MHz
create_clock -name {TCK}\
-period 166.67 \
-waveform {0 83.33} \
[ get_ports {TCK}]

# JTAG and Mi-V clocks are independent - adding asynchronous clock group

set_clock_groups -name {asyncl}\
-asynchronous\
-group [ get_clocks {CCC_0_inst_0/CCC_0_0/p11_inst_0/OUT0}]\
-group [ get_clocks {TCK}]
```

4. 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 in.
5. Save the changes made in the **Constraint Manager > Timing** dialog.

## 9 SoftConsole

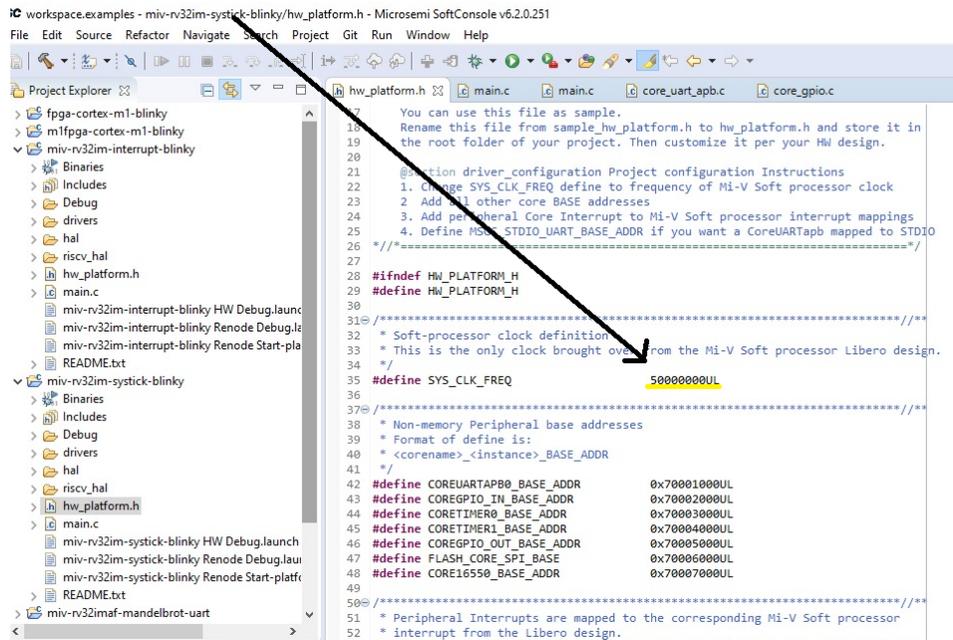
The SoftConsole version 6.0 or later is required to use the MIV\_RV32IMC. Each SoftConsole project requires the RISC-V Hardware Abstraction Layer (HAL) version 2.2 or greater. For more information on setting up project for RISC-V, see the SoftConsole release notes. The following steps briefly explain the changes that may be made to a SoftConsole project. More information on setting up SoftConsole for this core can be found in the **Quick Start Guide**. It can be found in the Help menu in the configurator, or if the core is selected in the Catalog.

### 9.1 Setting the System Clock Frequency and Peripheral Base Addresses

If UART is being used, the system clock frequency is provided to the software and is done in the `hw_platform.h` file by changing the `#define SYS_CLK_FREQ` to the clock frequency.

**Note:** This value should be in hertz.

**Figure 13 • Setting Clock Frequency and Peripheral Base Addresses**

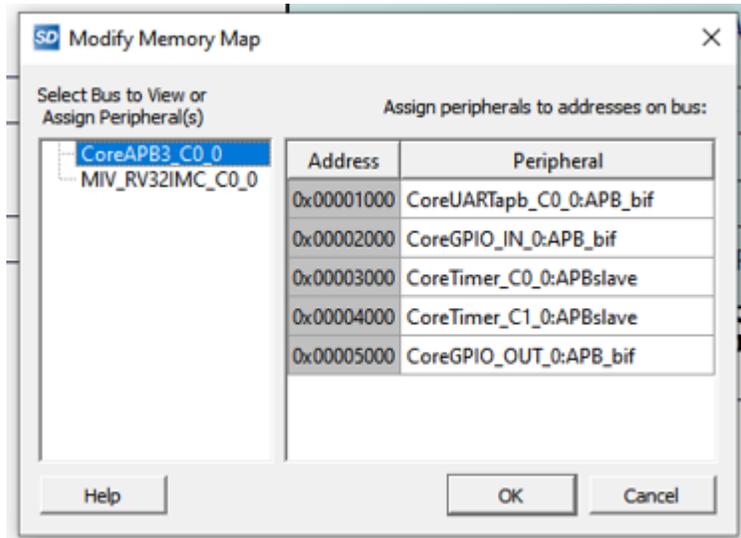


```

workspace.examples - miv-rv32im-systick-blinky/hw_platform.h - Microsemi SoftConsole v6.2.0.251
File Edit Source Refactor Navigate Search Project Git Run Window Help
Project Explorer
  fpga-cortex-m1-blinky
  m1fpga-cortex-m1-blinky
  miv-rv32im-interrupt-blinky
    Binaries
    Debug
    drivers
    hal
    riscv_hal
    hw_platform.h
    main.c
    miv-rv32im-interrupt-blinky HW Debug.launch
    miv-rv32im-interrupt-blinky Renode Debug.laun
    miv-rv32im-interrupt-blinky Renode Start-pla
  README.txt
  miv-rv32im-systick-blinky
    Binaries
    Includes
    Debug
    drivers
    hal
    riscv_hal
    hw_platform.h
    main.c
    miv-rv32im-systick-blinky HW Debug.launch
    miv-rv32im-systick-blinky Renode Debug.laun
    miv-rv32im-systick-blinky Renode Start-platf
  README.txt
  miv-rv32imaf-mandelbrot-uart
hw_platform.h
  18 You can use this file as sample.
  19 Rename this file from sample_hw_platform.h to hw_platform.h and store it in
  20 the root folder of your project. Then customize it per your HW design.
  21
  22 @section driver configuration Project configuration Instructions
  23 1. Change SYS_CLK_FREQ define to frequency of Mi-V Soft processor clock
  24 2. Add any other core BASE addresses
  25 3. Add peripheral Core Interrupt to Mi-V Soft processor interrupt mappings
  26 4. Define MIO_STDIO_UART_BASE_ADDR if you want a CoreUARTapb mapped to STDIO
  27
  28 //=====
  29 #ifndef HW_PLATFORM_H
  30 #define HW_PLATFORM_H
  31
  32 //=====
  33 * Soft-processor clock definition
  34 * This is the only clock brought over from the Mi-V Soft processor Libero design.
  35 #define SYS_CLK_FREQ 50000000UL
  36
  37 //=====
  38 * Non-memory Peripheral base addresses
  39 * Format of define is:
  40 * <corename>_<instance>_BASE_ADDR
  41 */
  42 #define COREUARTAPB0_BASE_ADDR 0x70001000UL
  43 #define COREGPIO_IN_BASE_ADDR 0x70002000UL
  44 #define CORETIMER0_BASE_ADDR 0x70003000UL
  45 #define CORETIMER1_BASE_ADDR 0x70004000UL
  46 #define COREGPIO_OUT_BASE_ADDR 0x70005000UL
  47 #define FLASH_CORE_SPI_BASE 0x70006000UL
  48 #define CORE16550_BASE_ADDR 0x70007000UL
  49
  50 //=====
  51 * Peripheral Interrupts are mapped to the corresponding Mi-V Soft processor
  52 * interrupt from the Libero design.
  
```

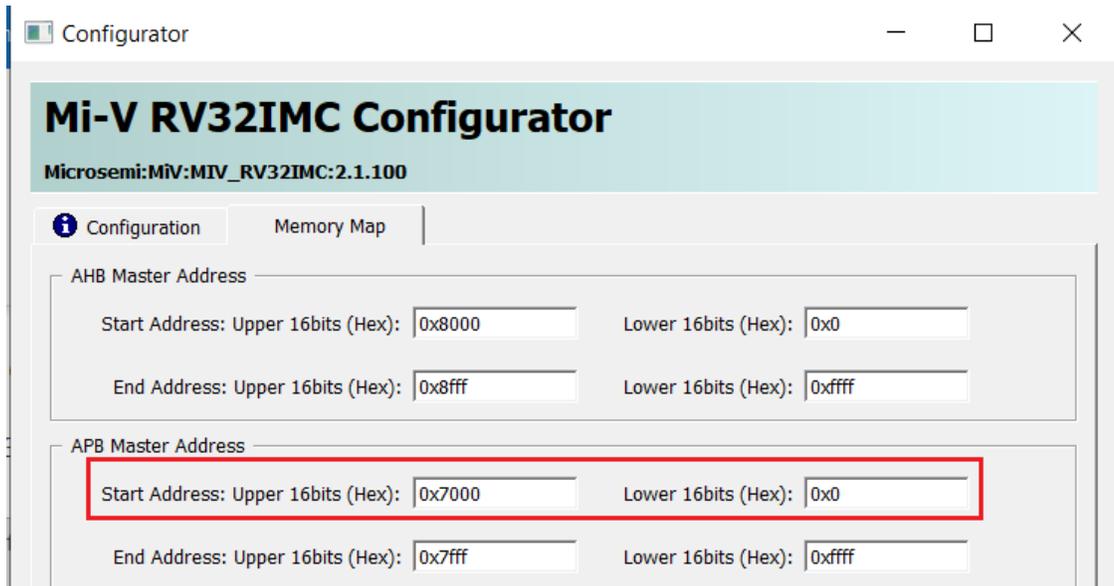
The `hw_platform.h` file sets the base address for peripherals. The base address of a peripheral can be found in the project memory map generated by Libero.

Figure 14 • Modify Memory Map Dialog



The peripheral address in the `hw_platform.h` file must match the address in Libero for the peripheral to function correctly. These peripherals are addressed for 0x0 because the RV32IMC core redirects these addresses accordingly. The following figure shows the RV32IMC configuration settings for peripherals.

Figure 15 • RV32IMC Configurator Memory Map



**Microsemi**

2355 W. Chandler Blvd.  
Chandler, AZ 85224 USA

Within the USA: +1 (480) 792-7200  
Fax: +1 (480) 792-7277

www.microsemi.com © 2020 Microsemi and its corporate affiliates. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation and its corporate affiliates. All other trademarks and service marks are the property of their respective owners.

Microsemi's product warranty is set forth in Microsemi's Sales Order Terms and Conditions. Information contained in this publication is provided for the sole purpose of designing with and using Microsemi products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is your responsibility to ensure that your application meets with your specifications. THIS INFORMATION IS PROVIDED "AS IS." MICROSEMI MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROSEMI BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE WHATSOEVER RELATED TO THIS INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROSEMI HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROSEMI'S TOTAL LIABILITY ON ALL CLAIMS IN RELATED TO THIS INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, YOU PAID DIRECTLY TO MICROSEMI FOR THIS INFORMATION. Use of Microsemi devices in life support, mission-critical equipment or applications, and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend and indemnify Microsemi from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microsemi intellectual property rights unless otherwise stated.

Microsemi Corporation, a subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), and its corporate affiliates are leading providers of smart, connected and secure embedded control solutions. Their easy-to-use development tools and comprehensive product portfolio enable customers to create optimal designs which reduce risk while lowering total system cost and time to market. These solutions serve more than 120,000 customers across the industrial, automotive, consumer, aerospace and defense, communications and computing markets. Headquartered in Chandler, Arizona, the company offers outstanding technical support along with dependable delivery and quality. Learn more at [www.microsemi.com](http://www.microsemi.com).

50200911