MIV_RV32 Handbook

Introduction

The MIV_RV32 is a processor core designed to implement the RISC-V instruction set for use in Microchip FPGAs.

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

There are three dedicated interrupts as well as six optional external interrupts in the MIV_RV32 processor core.

A quick start guide is available on how to create an MIV_RV32 Libero® design from the help menu in the core configurator. A migration guide is available detailing how to migrate a legacy MIV core design to use an MIV_RV32.

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 (TCM) is available and the size, up to 256 Kbytes, is defined by address range
- TCM APB Slave (TAS) to TCM
- Boot ROM feature to load an image and run from memory
- 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

Core Version

This Handbook applies to MIV_RV32 version 3.0.

The six 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_RV32 Quick Start Guide
- MIV_RV32 Migration Guide
- Supplementary Resource Utilization and Performance (RUP) tables

Supported Families

- PolarFire®
- PolarFire® SoC
- RT PolarFire®
- RTG4™

- IGLOO®2
- SmartFusion®2

Table of Contents

1. Resource Utilization and Performance

The device Resource Utilization and Performance (RUP) data is listed in tables Table 1-1 to Table 1-9 for the supported device families. The listed PolarFire information is also applicable to PolarFire SoC and RT PolarFire. 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 12.5 and Synplify Q-2020.03M. 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 1-1. RV32I APB TCM

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	1057	3723	4780	3788	108
RTG4	RTG4150L FCG1657	1091	3867	4958	3925	64
SmartFusion2	M2S150T FC1152	1094	3833	4927	3887	62
IGLOO2	M2GL150 FC1152	1091	3849	4940	3894	54
Configuration Parameters	**RISC-V Extensions:** I, **Multiplier:** n/a, **AHB Master:** n/a, **AHB Mirrored I/F:** n, **APB Master:** APB3, **APB Mirrored I/F:** n, **AXI Master:** n/a, **AXI Mirrored I/F:** n/a, **Reset Vector Address Upper 16bits:** 0x4000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** n, **Reconfigure BootROM:** n, **External IRQs:** 0, **Vectored Interrupts:** n, **TCM:** y (4k), **TCM APB Slave (TAS):** n/a, **Internal MTIME:** n, **Internal MTIME IRQ:** n, **Debug:** n, **Register Forwarding:** n, **ECC:** n, **GPR Registers:** n					

Table 1-2. RV32I APB All Features

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	943	3744	4687	3807	105
RTG4	RTG4150L FCG1657	946	3900	4846	3947	64
SmartFusion2	M2S150T FC1152	945	3902	4847	3972	65
IGLOO2	M2GL150 FC1152	946	3913	4859	3957	56
Configuration Parameters	**RISC-V Extensions:** IC, **Multiplier:** n/a, **AHB Master:** n/a, **AHB Mirrored I/F:** n, **APB Master:** APB3, **APB Mirrored I/F:** n, **AXI Master:** n/a, **AXI Mirrored I/F:** n/a, **Reset Vector Address Upper 16bits:** 0x4000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** n, **Reconfigure BootROM:** n, **External IRQs:** 0, **Vectored Interrupts:** n, **TCM:** y (4k), **TCM APB Slave (TAS):** n/a, **Internal MTIME:** n, **Internal MTIME IRQ:** n, **Debug:** n, **Register Forwarding:** n, **ECC:** n, **GPR Registers:** n					

Table 1-3. RV32IC APB TCM

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	943	3744	4687	3807	105
RTG4	RTG4150L FCG1657	946	3900	4846	3947	64
SmartFusion2	M2S150T FC1152	945	3902	4847	3972	65
IGLOO2	M2GL150 FC1152	946	3913	4859	3957	56
Configuration Parameters	**RISC-V Extensions:** IC, **Multiplier:** n/a, **AHB Master:** n/a, **AHB Mirrored I/F:** n, **APB Master:** APB3, **APB Mirrored I/F:** n, **AXI Master:** n/a, **AXI Mirrored I/F:** n/a, **Reset Vector Address Upper 16bits:** 0x4000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** n, **Reconfigure BootROM:** n, **External IRQs:** 0, **Vectored Interrupts:** n, **TCM:** y (4k), **TCM APB Slave (TAS):** n/a, **Internal MTIME:** n, **Internal MTIME IRQ:** n, **Debug:** n, **Register Forwarding:** n, **ECC:** n, **GPR Registers:** n					

Table 1-4. RV32IC APB All Features

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	3819	9542	13361	10144	66
RTG4	RTG4150L FCG1657	3849	9307	13156	9913	38
SmartFusion2	M2S150T FC1152	3848	9376	13224	10124	48
IGLOO2	M2GL150 FC1152	3849	9421	13270	10126	41
Configuration Parameters	**RISC-V Extensions:** IC, **Multiplier:** n/a, **AHB Master:** AHBLite, **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 16bits:** 0x8000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** y, **Reconfigure BootROM:** y, **External IRQs:** 6, **Vectored Interrupts:** y, **TCM:** y (4k), **TCM APB Slave (TAS):** y, **Internal MTIME:** y, **Internal MTIME IRQ:** y, **Debug:** y, **Register Forwarding:** y, **ECC:** y, **GPR Registers:** y					

Table 1-5. RV32IM (MACC-Pipelined) APB TCM

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	1228	4168	5396	4226	93
RTG4	RTG4150L FCG1657	1132	4425	5557	4484	61
SmartFusion2	M2S150T FC1152	1224	4468	5692	4531	61
IGLOO2	M2GL150 FC1152	1225	4445	5670	4486	53
Configuration Parameters	**RISC-V Extensions:** IM, **Multiplier:** MACC-Pipelined, **AHB Master:** n/a, **AHB Mirrored I/F:** n, **APB Master:** APB3, **APB Mirrored I/F:** n, **AXI Master:** n/a, **AXI Mirrored I/F:** n/a, **Reset Vector Address Upper 16bits:** 0x4000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** n, **Reconfigure BootROM:** n, **External IRQs:** 0, **Vectored Interrupts:** n, **TCM:** y (4k), **TCM APB Slave (TAS):** n/a, **Internal MTIME:** n, **Internal MTIME IRQ:** n, **Debug:** n, **Register Forwarding:** n, **ECC:** n, **GPR Registers:** n					

Table 1-6. RV32IM (MACC-Pipelined) All Features

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	4100	9836	13936	10453	65
RTG4	RTG4150L FCG1657	4130	9784	13914	10432	38
SmartFusion2	M2S150T FC1152	4022	9778	13800	10401	47
IGLOO2	M2GL150 FC1152	4028	9891	13919	10513	40
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 16bits:** 0x8000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** y, **Reconfigure BootROM:** y, **External IRQs:** 6, **Vectored Interrupts:** y, **TCM:** y (4k), **TCM APB Slave (TAS):** y, **Internal MTIME:** y, **Internal MTIME IRQ:** y, **Debug:** y, **Register Forwarding:** y, **ECC:** y, **GPR Registers:** y					

Table 1-7. RV32IMC (Fabric) APB TCM

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	1115	4985	6100	5045	102
RTG4	RTG4150L FCG1657	1114	4867	5981	4929	62
SmartFusion2	M2S150T FC1152	1115	4906	6021	4971	65
IGLOO2	M2GL150 FC1152	1117	4809	5926	4889	57
Configuration Parameters	**RISC-V Extensions:** IMC, **Multiplier:** Fabric, **AHB Master:** n/a, **AHB Mirrored I/F:** n, **APB Master:** APB3, **APB Mirrored I/F:** n, **AXI Master:** n/a, **AXI Mirrored I/F:** n/a, **Reset Vector Address Upper 16bits:** 0x4000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** n, **Reconfigure BootROM:** n, **External IRQs:** 0, **Vectored Interrupts:** n, **TCM:** y (4k), **TCM APB Slave (TAS):** n/a, **Internal MTIME:** n, **Internal MTIME IRQ:** n, **Debug:** n, **Register Forwarding:** n, **ECC:** n, **GPR Registers:** n					

Table 1-8. RV32IMC (Fabric) All features

Family	Part	Synthesis			P&R LEs	Performance/MHz
		DFF	4LUT	Total		
PolarFire	MPF500T-1 FCG1152E	3792	9498	13290	10926	73
RTG4	RTG4150L FCG1657	3795	9403	13198	10919	42
SmartFusion2	M2S150T FC1152	3789	9488	13277	10894	49
IGLOO2	M2GL150 FC1152	3790	9446	13236	10898	41
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 16bits:** 0x8000, **Reset Vector Address Lower 16bits:** 0x0, **BootROM:** y, **Reconfigure BootROM:** y, **External IRQs:** 6, **Vectored Interrupts:** y, **TCM:** y (4k), **TCM APB Slave (TAS):** 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.

1.1 Typical Resource Utilization

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

Table 1-9. Option Resources

Feature	Parts	Synthesis		
		Average DFF	Average 4LUT	Average Total
AHBL	MPF500T-1FCG1152E RTG4150L FCG1657 M2S150T FC1152 M2GL150 FC1152	108	102	210
APB		115	144	259
AXI		514	492	1006
Ext_sys_interrupts (6)		6	61	67
Vectored interrupts		35	154	189
BootROM		39	125	164
Reconfig BootROM		147	139	286
TCM (4k)		62	372	434
TAS		0	84	84
Mtime & Mtime irq		160	425	585
Debug		564	1756	2320
ECC		12	407	429
GPR Registers		989	1544	2533
Register Forwarding		5	289	294

1.2 Benchmarks

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	2.533
	TCM	IM	MACC	0	0	0	1.567
	TCM	IM	MACC Pipe	0	0	0	1.567
	TCM	IMC	MACC	0	0	0	1.567
	TCM	IMC	MACC	1	0	0	1.567
	TCM	IMC	MACC Pipe	0	0	0	1.5
	TCM	IM	Fabric	0	0	0	1.067
	TCM	I	n/a	1	1	0	1.067
	TCM	IMC	Fabric	0	0	0	1.033
	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	0.533
	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

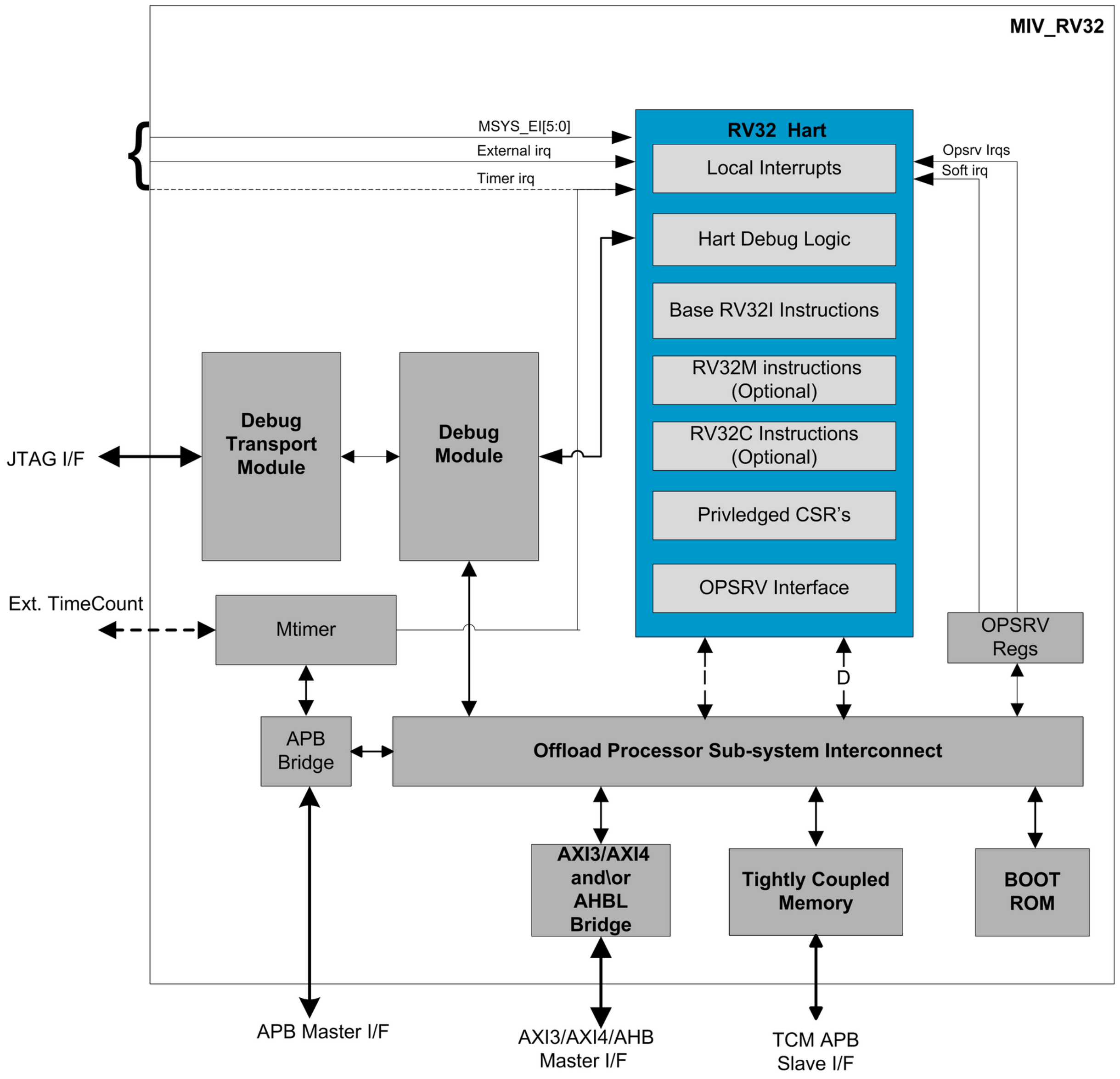
2. Functional Description

The following sections provide a functional description of the MIV_RV32 processor core.

2.1 MIV_RV32 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, TCM with a TCM APB Slave port (TAS), AHB\AXI\APB master interfaces, and OPSRV registers. The following figure shows the block level architecture of MIV_RV32 device.

Figure 2-1. MIV_RV32 Block Diagram



The following table lists the key features of the core.

Table 2-1. MIV_RV32 Architecture Features

Feature	Value	Units	Notes
ISA support	RV32IMC	—	Base RV32I, optional multiply and divide, and optional compressed extensions
Harts	1	—	MIV_RV32 processor
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	—	TAS provides access to TCM

2.2 Hart

The MIV_RV32 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 RISC-V 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.

2.3 Memory System

The core is non-cached. Up to 256 Kbytes of TCM is available for instruction and data storage. A range of system peripherals are accessed across AXI (AXI4/AXI3), AHB, and APB bus interfaces.

2.4 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 single and double bit ECC errors, GPR single bit ECC errors, 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.

2.5 Debug Support Through 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
- Single hardware breakpoint is available

2.6 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 might 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.

2.7 TCM

The core supports up to 256 Kbytes of TCM. The size of the memory is defined by the start and end address of the TCM. The processor can be booted from this memory region by setting the `RESET_VECTOR_ADDRESS` to the address of the TCM.

The TCM can be programmed through the TAS interface on the MIV_RV32. On PolarFire and PolarFire SoC devices, if enabled, the TCM can also be initialized by the System Controller using on-board storage, such as sNVM, uPROM, or SPI.

The TCM initialization feature necessitates the use of PolarFire Low Power RAM, which adds significantly to the core resource usage. For this reason, the TCM Initialization feature has not been fully adopted and is currently only available by enabling a local parameter, `l_cfg_hard_tcm0_en`, in the `miv_rv32_opsrv_cfg_pkg.v` file prior to synthesis, as shown in the following table.

Table 2-2. TCM Initialization

Local Parameter	Value	Description
`l_cfg_hard_tcm0_en`	1’b0 (default)	Use inferred RAM for TCM
	1’b1	Instantiate PF Low Power RAM for TCM which can be initialized by the System Controller

This package file is read-only, so when the value of `l_cfg_hard_tcm0_en` is modified in Libero, the **Remove the read-only attribute and modify the file directly** option must be selected.

2.8 TAS Port

The TAS port 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 TAS are of the same size to avoid memory read or write violations.

2.9 Clocks

The system clock frequency must be chosen to meet design timing requirements with clock constraints applied. The tables in the 1. Resource Utilization and Performance section list the upper clock frequency obtained for a specified device from each supported FPGA family, whilst meeting the 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 must remain within the maximum frequency permitted for the JTAG probe in use. The TCK must have clock constraints applied. For more information, see the 7. Design Constraints section.

2.10 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 6.3 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 RESETN and is 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 Table 4-36.

The JTAG_TRSTN is an active low reset signal for the JTAG Debug Transport module.

3. Interface

3.1 Configuration Parameters

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

Table 3-1. MIV_RV32 Parameters and Generics Descriptions

Name	Range	Default Value	Description
RESET_VECTOR_ADDR_1	0x1000-0xFFFF	0x8000	This is the address from which the processor starts executing 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 Note[1]
AXI_START_ADDR_1	0x1000-0xFFFF	0x6000	This is the AXI start address. AXI_START_ADDR_1 and A-XI_START_ADDR_0 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. AXI_END_ADDR_1 and AXI_ END_ADDR_0 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 Note[2]
AHB_START_ADDR_1	0x1000-0xFFFF	0x8000	This is the AHB start address. AHB_START_ADDR_1 and AHB_START_ADDR_0 represent the upper and lower 16 bits of the address respectively.
AHB_START_ADDR_0	0x0000-0xFFFF	0x0	

...........continued

Name	Range	Default Value	Description
AHB_END_ADDR_1	0x1000-0xFFFF	0x8FFF	This is the AHB end address. AHB_END_ADDR_1 and AHB_ END_ADDR_0 represent the upper and lower 16 bits of the address respectively.
AHB_END_ADDR_0	0xFFFF-0xFFFC	0xFFFF	
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 Note[2]
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 respectively.
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 respectively.
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 respectively.
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 respectively.
TCM_END_ADDR_0	0xFFFF-0xFFFC	0x0x3FFF	
TCM_TAS_PRESENT	0 or 1	0	TCM ABP Slave Present 0: Disabled 1: Enabled
TAS_START_ADDR_1	0x1000-0xFFFF	0x4000	This is the TCM ABP Slave start address. TAS_START_ADDR_1 and TAS_START_ADDR_0 represent the upper and lower 16 bits of the address respectively.
TAS_START_ADDR_0	0x0000-0xFFFF	0x0	

...........continued

Name	Range	Default Value	Description
TAS_END_ADDR_1	0x1000-0xFFFF	0x4000	This is the TCM APB Slave end address. TAS_END_ADDR_1 and TAS_END_ADDR_0 represent the upper and lower 16 bits of the address respectively.
TAS_END_ADDR_0	0xFFFF-0xFFFC	0x0x3FFF	
BOOT_ROM	0 or 1	0	BOOT_ROM Present 0: Disabled 1: Enabled
SRC_START_ADDR_1	0x1000-0xFFFF	0x8000	This is the SRC_START (Source starting) address read by the BOOT_ROM. SRC_START_ADDR_1 and SRC_START_ADDR_0 represent the upper and lower 16 bits of the address respectively.
SRC_START_ADDR_0	0x0000-0xFFFF	0x0	
SRC_END_ADDR_1	0x1000-0xFFFF	0x8000	This is the SRC_END (Source final) address read by the BOOT_ROM. SRC_START_ADDR_1 and SRC_START_ADDR_0 represent the upper and lower 16 bits of the address respectively.
DEST_ ADDR_1	0x1000-0xFFFF	0x4000	This is the DEST_ADDR (Destination starting) address written by the BOOT_ROM. DEST_ADDR_1 and DEST_ADDR_0 represent the upper and lower 16 bits of the address respectively.
DEST_ ADDR_0	0x0000-0xFFFF	0x3FFF	
RECONFIG_BOOT_ROM	0 or 1	0	Reconfigurable BOOT_ROM Present 0: Disabled 1: Enabled (Source/ destination addresses reconfigurable in software)
GEN_DECODE_RV32	0 to 3	3	RISCV ISA Extension Select 0: I 1: IC 2: IM 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 1: Enabled
NUM_EXT_IRQS	0 to 6	6	Number of external interrupts

...........continued

Name	Range	Default Value	Description
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: `MTIME timebase = MTIME_PRESCALAR/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

Notes:
1. This parameter is used only when an AXI Master is selected.
2. This parameter is used only when an AHB Master is selected.

3.2 I/O Signals

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

Figure 3-1. MIV_RV32 Full I/O View

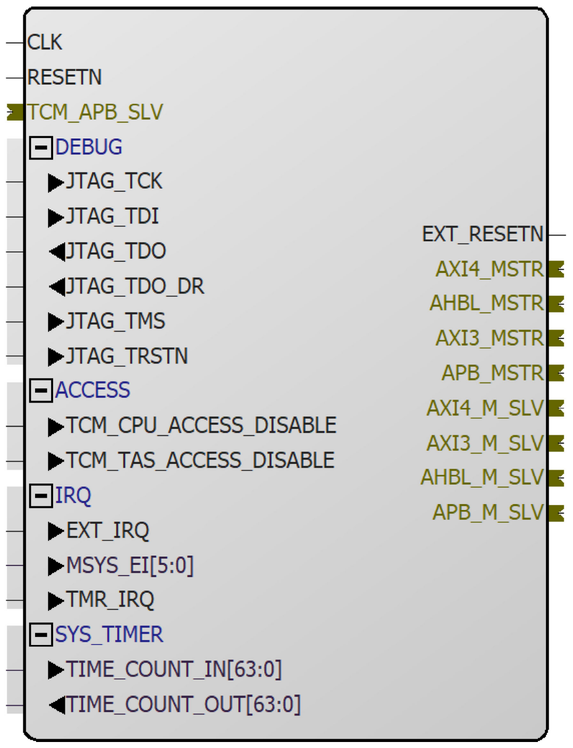


Table 3-2. MIV_RV32 I/O Signal Description

Port Name	Width	Direction	Description
Global Signals			
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).
JTAG Interface Signals			
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.
JTAG_TRSTN	1	In	Test Reset (TRSTN). This is an optional signal used to reset the TAP controllers state machine. This signal is active low.
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.
Interrupt Signals			

...........continued			
Port Name	**Width**	**Direction**	**Description**
EXT_IRQ	1	In	External interrupt from peripheral source. An active high level based interrupt signal. Tie this input low if unused.
MSYS_EI	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.
System Time Signals			
TIME_COUNT_IN	64	In	External system timer count
TIME_COUNT_OUT	64	Out	Internal system timer count
TCM Access Signals			
TCM_CPU_ACCESS_DISABLE	1	In	When asserted, CPU access to the TCM is disabled.
TCM_TAS_ACCESS_DISABLE	1	In	When asserted, TAS access to the TCM is disabled.
APB Master Interface			
APB_MSTR_PADDR	32	Out	APB Master Interface. The address range is 0x1000_0000 to 0xFFFF_FFF-F. 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	
AHB Master Interface			
AHB_MSTR_ HMASTLOCK	1	Out	AHB Master Interface. The address range is 0x1000_0000 to 0xFFFF_FFF-F. This interface can also be configured as a mirrored slave through the GUI.
AHB_MSTR_ HTRANS	2	Out	
AHB_MSTR_ HWRITE	1	Out	
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 is used only when AHB_SLAVE_MIRROR is set to 1.
AXI Master Interface			

...........continued			
Port Name	**Width**	**Direction**	**Description**
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	
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	
TCM APB Slave Interface (TAS)			
TCM_APB_SLV_PADDR	32	In	The address range is 0x1000_0000 to 0xFFFF_ FFFF.
TCM_APB_SLV_PSEL	1	In	
TCM_APB_SLV_PENABLE	1	In	
TCM_APB_SLV_PWRITE	1	In	
TCM_APB_SLV_PRDATA	32	Out	
TCM_APB_SLV_PWDATA	32	In	
TCM_APB_SLV_PREADY	1	Out	
TCM_APB_SLV_PSLVERR	1	Out	

4. 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). The M provides hardware support for these operations and the C 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.

4.1 Processor Operating States

The processor has the following operating modes.

- 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 while debugging using the JTAG interface.

4.2 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.

4.3 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.

4.4 General Purpose Registers

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

Table 4-1. 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
x9	s1	Saved Register

...........continued		
Register	**ABI Name**	**Description**
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

4.5 Machine Control and Status Registers

As the core only supports the 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 4-2. mvendorid CSR (0xF11)

Bits	31:0
Field	Vendor ID
R/W	RO
Reset	Preset value = l_core_vendorid

The vendor ID CSR reads the value defined by the `l_core_vendorid` constant configured at build time.

Table 4-3. marchid CSR(0xF12)

Bits	31:0
Field	Architecture ID
R/W	RO
Reset	Preset value = l_core_marchid

Table 4-4. mimpid CSR(0xF13)

Bits	31:0
Field	Implementation ID
R/W	RO
Reset	Preset value = l_core_mimpid

Table 4-5. mhartid CSR(0xF14)

Bits	31:0
Field	Hart ID
R/W	RO
Reset	Preset value = l_hart_id

Table 4-6. mstatus CSR(0x300)

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

Bits	6:15	4:13	2:11	10:9	8	7	6	5	4	3	2	1	0
Field	XS	FS	MPP	—	SPP	MPIE	—	SPIE	UPIE	MIE	—	SIE	UIE
R/W	RO	R0	RO	WPRI	RO	RW	WPRI	RW	RO	RO	WPRI	RO	RO
Reset	—	—	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 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 4-7. misa CSR (0x301)

Bits	31:30	29:26	25:0
Field	Base ISA	—	Extension
R/W	RO	WPRI	RO

Note: Base ISA is RV32 = 2’b01.

According to the RISC-V architecture, misa might 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 4-8. misa CSR Extension Bit Description

Bit	Character	Description	MIV_RV32 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

...........continued

Bit	Character	Description	MIV_RV32 Implementation
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 4-9. 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 4-10. 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	—	—	—

...........continued													
Bits	**15:12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	WPRI	RO	WPRI	RO	RO	RO	WPRI	RO	RO	RO	WPRI	RO	RO
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 4-11. 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 the 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 4-12. mtvec CSR (0x305)

Bits	31:2	1:0
Field	Exception vector base	mode
R/W	The mtvec exception base vector is a read/write RW register. .	RW
Reset	The reset value is not defined.	2'b00 (direct mode) 2'b01 (vectored mode)

Table 4-13. mepc CSR (0x341)

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

Note: `mepc[0]` is always 0. Therefore, it is hardwired 0.

Table 4-14. 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] is 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 4-15. mcause Exception Codes

Interrupt	Exception code	Description
1	3	MSI [Highest priority]
1	7	MTI
1	11	MEI
1	*16*	*MGEUI*
1	*17*	*MGECI*
1	*24–29*	*MSYS_EI0-MSYS_EI5*
1	30	*OPSRV IRQ*
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*	*Instruction fetch read bus error*
0	*25*	*Instruction fetch read parity error*
0	*26*	*Data load bus error*
0	*27*	*Data load parity error [Lowest priority]*

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

Table 4-16. mtval CSR (0x343)

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

Table 4-17. MTVAL Value Following Trap Taken

Interrupt/exception cause	MTVAL
MSI [Highest priority]	0
MTI	0
MEI	0
MGEUI	0
MGECI	0

...........continued

Interrupt/exception cause	MTVAL
MICNT	0
MCCNT	0
MHCNT	0
MSYS_EI0-MSYS_EI7	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
Instruction fetch read bus error	Program counter of the instruction retiring
Instruction fetch read parity error	Program counter of the instruction retiring
Data load bus error	Load address of faulting access
Data load parity error [Lowest priority]	Load address of faulting access

Table 4-18. mscratch CSR (0x340)

Bits	31:0
Field	mscratch
R/W	RW
Reset	—

Table 4-19. time CSR (0xC01)

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

Table 4-20. timeh CSR (0xC81)

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

Time is a read-only user CSR.

4.6 Debug Module

The MIV_RV32 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:

- Debug Transfer module with an interface module DMI
- Debug Unit—abstract command based with System Bus Access
- Hart Debug Logic—debug CSR's and halt or run logic

4.6.1 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 4-21. TAP Controller Instructions

Address	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 domain crossing 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

4.6.2 Debug Unit

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

Table 4-22. Debug Module Registers

Address	Mnemonic	Full name
0x00–0x03	—	Reserved
0x04	DATA0	Abstract Data 0
0x05	—	Abstract Data 1, not implemented
0x06–0x0F	—	Reserved

...........continued

Address	Mnemonic	Full name
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

4.6.3 Hart Debug Logic

The MIV_RV32 hart implements the halt\run logic and required debug CSR registers.

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

Table 4-23. Debug Control and Status Registers

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

4.6.3.1 Debug Control and Status CSR (DCSR–0x7B0)

The following table lists the debug control and status registers.

Table 4-24. Debug Control and Status Registers

Bits	Name	Access	Reset Value	Description
31:28	xdebugver	RO	4	The value of the field (4) indicates that the debug support exists as described in the *RISC-V Debug Spec version 0.13*.
27:16	rsrv3	RO	0	Reserved for future use.

...........continued

Bits	Name	Access	Reset Value	Description
15	ebreakm	RW	0	Encoding: • 0 - ebreak instructions in M-mode behave as described in the Privileged Specification. • 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 or nmip 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 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. The Mi-V debug has the field hardwired to 3. Only machine mode is supported.

4.6.3.2 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 4-25. 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 might write DPC to change where the hart resumes.

Table 4-26. DPC CSR Register

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

4.7 Trigger Unit

The MIV_RV32 core supports basic hardware trigger functionality. The Trigger Unit is only active in configurations were the Debug module is present. There is a single hardware trigger, which can be set to fire on the execution of instructions at a given memory address. Once the trigger fires, the core enters Debug mode and halts. Breakpoint exceptions are not supported. The following tables list and describe the implemented trigger registers.

Table 4-27. Trigger Registers

Address	Mnemonic	Full Name
0x7A0	TSELECT	Trigger select
0x7A1	TDATA1/MCONTROL	Trigger data 1 or Match control
0x7A2	TDATA2	Trigger data 2
0x7A3–0x7BF	—	Reserved

4.7.1 Trigger Data 1/Match Control

TDATA1/MCONTROL Register

Bits	Name	Access	Reset Value	Description
31:28	type	RW	2	This bit field is described as a part of the TDATA1 register. As MCONTROL is only implemented this field is hardwired to 4'h2.
27	dmode	RO	0	This field is Hard encoded, tied 1. Only Debug mode can write TDATA registers at the selected TSELECT. Writes from the Machine mode are ignored.
26:21	rsv1	RO	0	Reserved for future use. Maskmax = 0, sizehi = 0 (only Xlen =32 supported)
20	hit	RO	0	The hardware sets this bit when the given trigger matches. The trigger's user can set or clear it at any time. It determines the matching trigger(s).
19	select	RO	0	This bit is hardwired to 0, so that the trigger performs a match only on the virtual address.

...........continued

Bits	Name	Access	Reset Value	Description
18	timing	RO	0	The bit is hardwired to 0, so that the action for this trigger is taken just before the instruction that triggered it is executed, but after all preceding instructions are committed.
17:16	sizelo	RO	0	The field is hardwired to zero. The trigger attempts to match against an access of any size.
15:12	action	RO	0	Hard Encoded, tied 1. When the trigger fires, Debug Mode is entered.
11	chain	RO	0	Hard Encoded, tied 0: When this trigger matches, the configured action is taken.
10:7	match	RO	0	Hard Encoded, tied 0: matches when the value equals TDATA2.
6	m	RO	0	Hard Encoded, tied 1: trigger enabled in the Machine mode.
5:3	rsv0	RO	0	Reserved for future use.
2	execute	RW	0	When set, the trigger fires on the virtual address of an instruction that is executed.
1	store	RO	0	Not used, tied 0.
0	load	RO	0	Not used, tied 0.

4.7.2 Trigger Select

This register determines which trigger is accessible through the other trigger registers. The set of accessible triggers begin at 0, and are contiguous. In this instance, a single trigger is implemented as listed in the following table.

Table 4-28. TSELECT Register

Bits	Name	Access	Reset Value	Description
31:1	rsv0	RO	0	Reserved for future use
0	index	RO	0	This field determines which trigger is accessible through the other registers. An index of 1 is supported in this instance.

4.7.3 Trigger Data 2

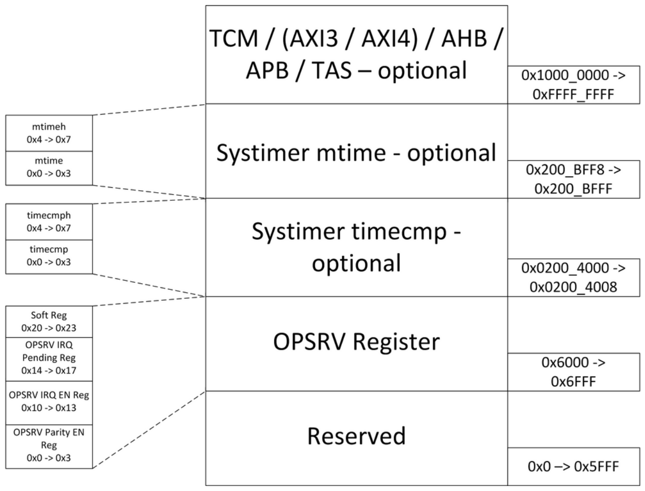
Table 4-29. TDATA2 Register

Bits	Name	Access	Reset Value	Description
31:0	data	RW	—	The field contains trigger-specific data.

4.8 Memory Map

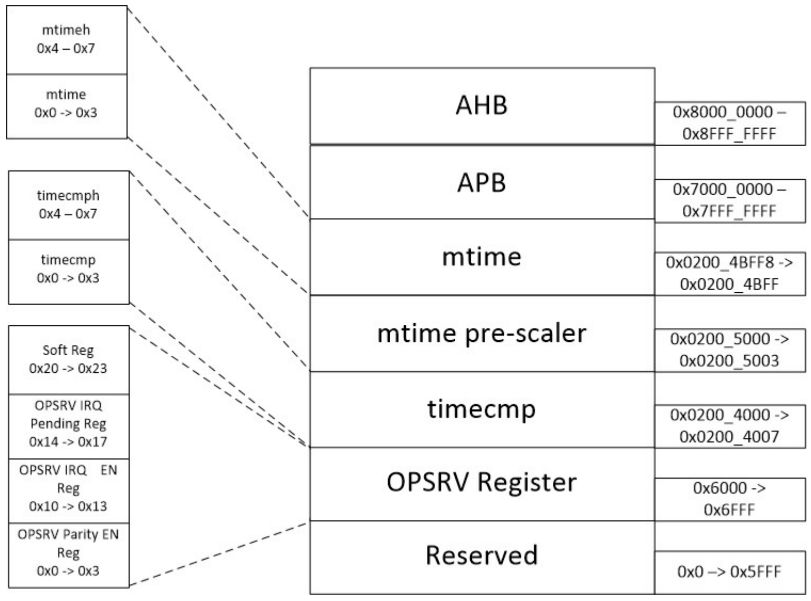
The memory map of the core is highly configurable. The GPRs, CSRs, and debug registers are contained in the reserved range. The OPSRV Register is memory mapped to 0x6000. The optional 64-bit timecmp register is mapped to 0x0200_4000. The mtime pre-scaler is mapped to 0x0200_5000. The 64-bit mtime register is mapped to 0x0200_BFF8. The following figure shows the memory map.

Figure 4-1. Memory Map


TCM / (AXI3 / AXI4) / AHB / APB / TAS – optional
0x1000_0000 -> 0xFFFF_FFFF
Systimer mtime - optional
0x200_BFF8 -> 0x200_BFFF
Systimer timecmp - optional
0x0200_4000 -> 0x0200_4008
OPSRV Register
0x6000 -> 0x6FFF
Reserved
0x0 –> 0x5FFF
mtimeh 0x4 -> 0x7
mtime 0x0 -> 0x3
timecmph 0x4 -> 0x7
timecmp 0x0 -> 0x3
Soft Reg 0x20 -> 0x23
OPSRV IRQ Pending Reg 0x14 -> 0x17
OPSRV IRQ EN Reg 0x10 -> 0x13
OPSRV Parity EN Reg 0x0 -> 0x3

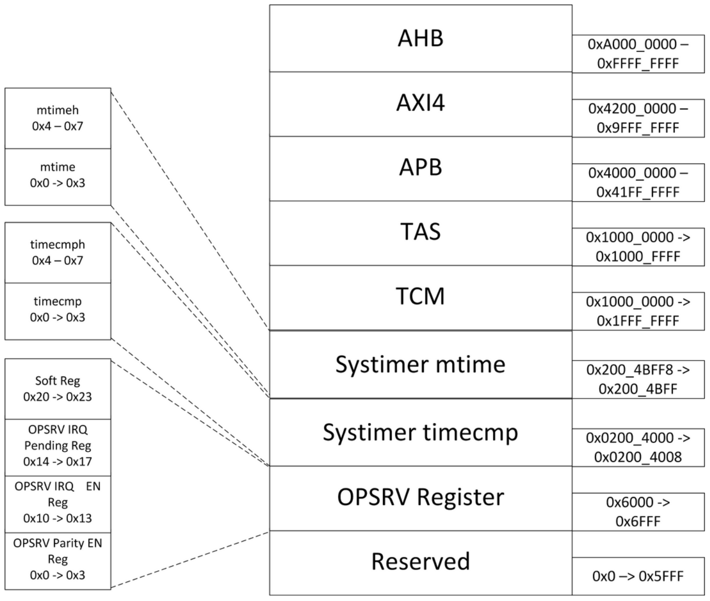
The following figure shows the default memory map for the core.

Figure 4-2. Default Memory Map


AHB
0x8000_0000 – 0x8FFF_FFFF
APB
0x7000_0000 – 0x7FFF_FFFF
mtime
0x0200_4BFF8 -> 0x0200_4BFF
mtime pre-scaler
0x0200_5000 -> 0x0200_5003
timecmp
0x0200_4000 -> 0x0200_4007
OPSRV Register
0x6000 -> 0x6FFF
Reserved
0x0 –> 0x5FFF
mtimeh 0x4 – 0x7
mtime 0x0 -> 0x3
timecmph 0x4 – 0x7
timecmp 0x0 -> 0x3
Soft Reg 0x20 -> 0x23
OPSRV IRQ Pending Reg 0x14 -> 0x17
OPSRV IRQ EN Reg 0x10 -> 0x13
OPSRV Parity EN Reg 0x0 -> 0x3

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

Figure 4-3. Example System Memory Map


AHB
0xA000_0000 – 0xFFFF_FFFF
AXI4
0x4200_0000 – 0x9FFF_FFFF
APB
0x4000_0000 – 0x41FF_FFFF
TAS
0x1000_0000 -> 0x1000_FFFF
TCM
0x1000_0000 -> 0x1FFF_FFFF
Systimer mtime
0x200_4BFF8 -> 0x200_4BFF
Systimer timecmp
0x0200_4000 -> 0x0200_4008
OPSRV Register
0x6000 -> 0x6FFF
Reserved
0x0 –> 0x5FFF
mtimeh 0x4 – 0x7
mtime 0x0 -> 0x3
timecmph 0x4 – 0x7
timecmp 0x0 -> 0x3
Soft Reg 0x20 -> 0x23
OPSRV IRQ Pending Reg 0x14 -> 0x17
OPSRV IRQ EN Reg 0x10 -> 0x13
OPSRV Parity EN Reg 0x0 -> 0x3

4.9 Subsystem Restrictions

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

4.10 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.

4.10.1 Vectored and Non-Vectored Interrupts

In the Vectored mode, the BASE field in the mtvec register is configured to reset vector + 0x04. Each interrupt makes the PC to jump to a vector location determined by the mcause value.

In the Non-vectored mode, the mtvec is a RW register. All the interrupts make the PC to jump to the fixed value determined by the mtvec register. The cause of the exception can then be determined by checking the value in the mcause register. The default value of the mtvec register in this case is reset vector + 0x04.

The implementation is in accordance with the *section 3.1.12 of the privileged specification*.

4.10.2 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.

4.10.3 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 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 generates 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.

4.10.4 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.

4.10.5 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 4-30. 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

4.10.6 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 4-31. 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 4-32. 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.

4.11 OPSRV Register

The offload processor subsystem interfaces with the hart and provides an interconnect for the interfaces. The `TAS`, `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 4-33. opsrv_cfg (0x6000)

Bits	31:1	0
Field	—	`opsrv_parity_en`
R/W	R0	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 4-34. 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	R0	RW	R0	RW	RW

...........continued					
Bits	**31:5**	**4**	**3:2**	**1**	**0**
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 `opsrv_irq`. Machine interrupts still needs to be enabled.

Table 4-35. opsrv_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	R0	RW	R0	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 `opsrv_irq` is triggered, if the corresponding enable bit is set. Writing a 1 to any set bit clears it.

Table 4-36. opsrv_soft_reg (0x6020)

Bits	31:3	2	1	0
Field	—	`core_gpr_ded_reset`	`soft_irq`	`soft_rst`
R/W	R0	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 must 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.

4.12 BOOT ROM

An optional Boot ROM feature is available for this core. Once enabled in the GUI, on reset, the core copies data from a memory mapped `source` memory into a `destination` memory location and then the core will boot from the destination memory location. The `source` start or end addresses and the`destination` start address can be provided through GUI inputs. If the RECONFIG_BOOTROM option is enabled, then the addresses become software reconfigurable, which can be used with a soft reset to reboot and run alternative code. The BOOT ROM associated address registers are listed in the following table.

Table 4-37. src_start_addr (0xA100)

Bits	31:0
Field	src_start_addr
R/W	RW
Reset	Last written value

Table 4-38. src_end_addr (0xA104)

Bits	31:0
Field	src_start_addr

...........continued	
Bits	**31:0**
R/W	RW
Reset	Last written value

Table 4-39. destination_addr (0xA108)

Bits	31:0
Field	destination_addr
R/W	RW
Reset	Last written value

4.13 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 4-40. mtimecmp (0x200_4000)

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

Table 4-41. 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 must be enabled for this interrupt to be taken.

Table 4-42. mtime_prescaler (0x200_5000)

Bits	31:0
Field	mtime_prescaler
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 4-43. mtime (0x200_BFF8)

Bits	31:0
Field	mtime
R/W	RW
Reset	0

Table 4-44. mtimeh (0x200_BFFC)

Bits	31:0
Field	mtimeh
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`≥ `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`(`mtimcmph`) 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 must not occur when using the `mtime` compare register. For example, with a prescaler of 50 and a 50 MHz system clock, it takes approximately 500,000 years to reach overflow. However, if a lower prescaler value is set, this occurs sooner, or the software must 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 must be set.

4.14 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 (SECDED).

If RAM based GPRs are used (default) and ECC is enabled, a fabric encoder and decoder is instantiated for the RAM. If the TCM is used, an encoder and decoder isl also 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 SECDED 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 occurs on a DED regardless of the interrupts being enabled.

When ECC is enabled for the TCM, ECC bit is not initialized. This process must 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 TAS also generates parity if ECC is enabled.

5. Tool Flow

5.1 License

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

5.1.1 RTL

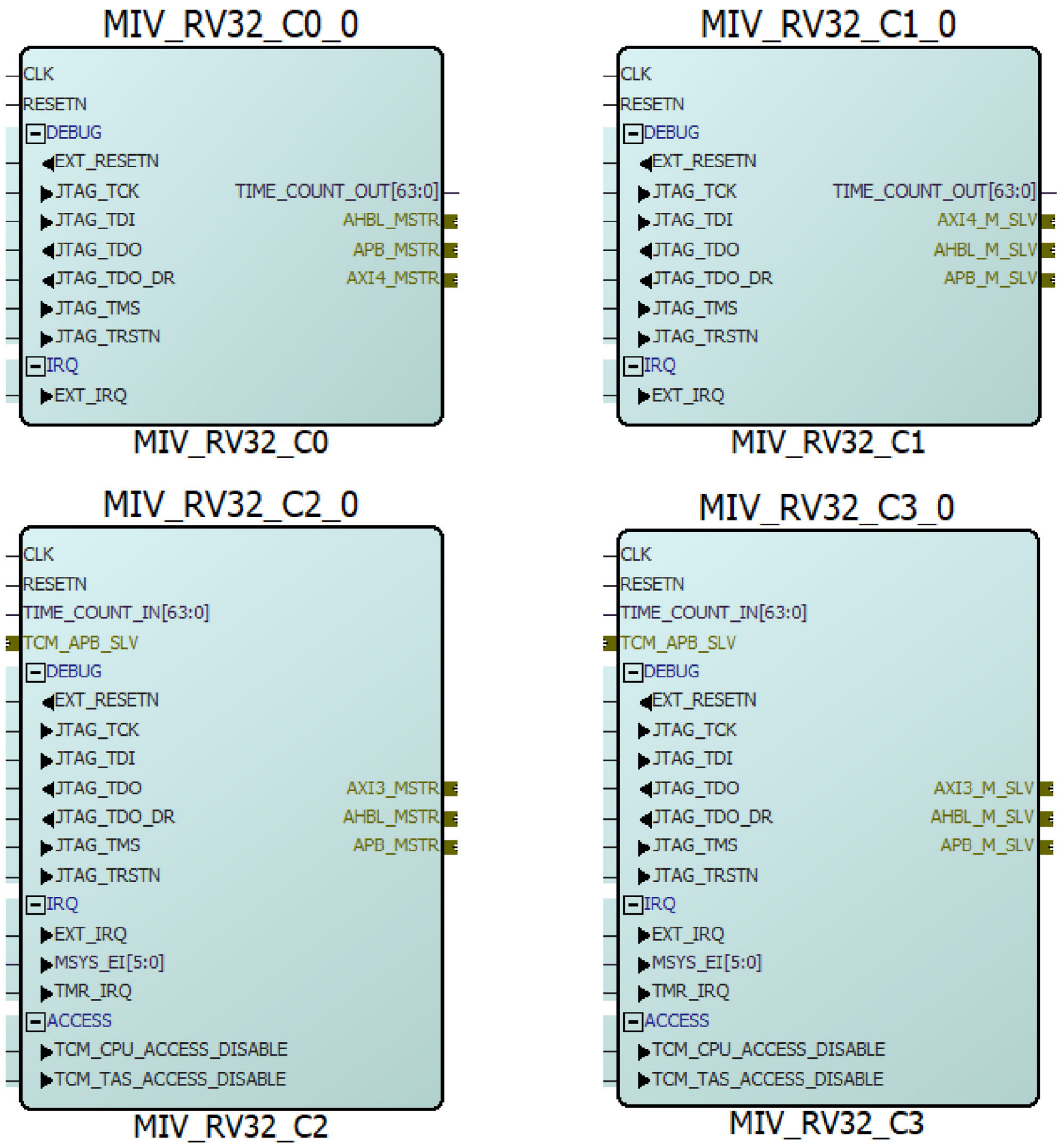
Complete Verilog source code is provided for the core. The core can be readily instantiated within SmartDesign. Simulation, synthesis, and layout can be performed within Libero SoC.

5.2 SmartDesign

The MIV_RV32 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 5-1. SmartDesign MIV_RV32 Instance Views

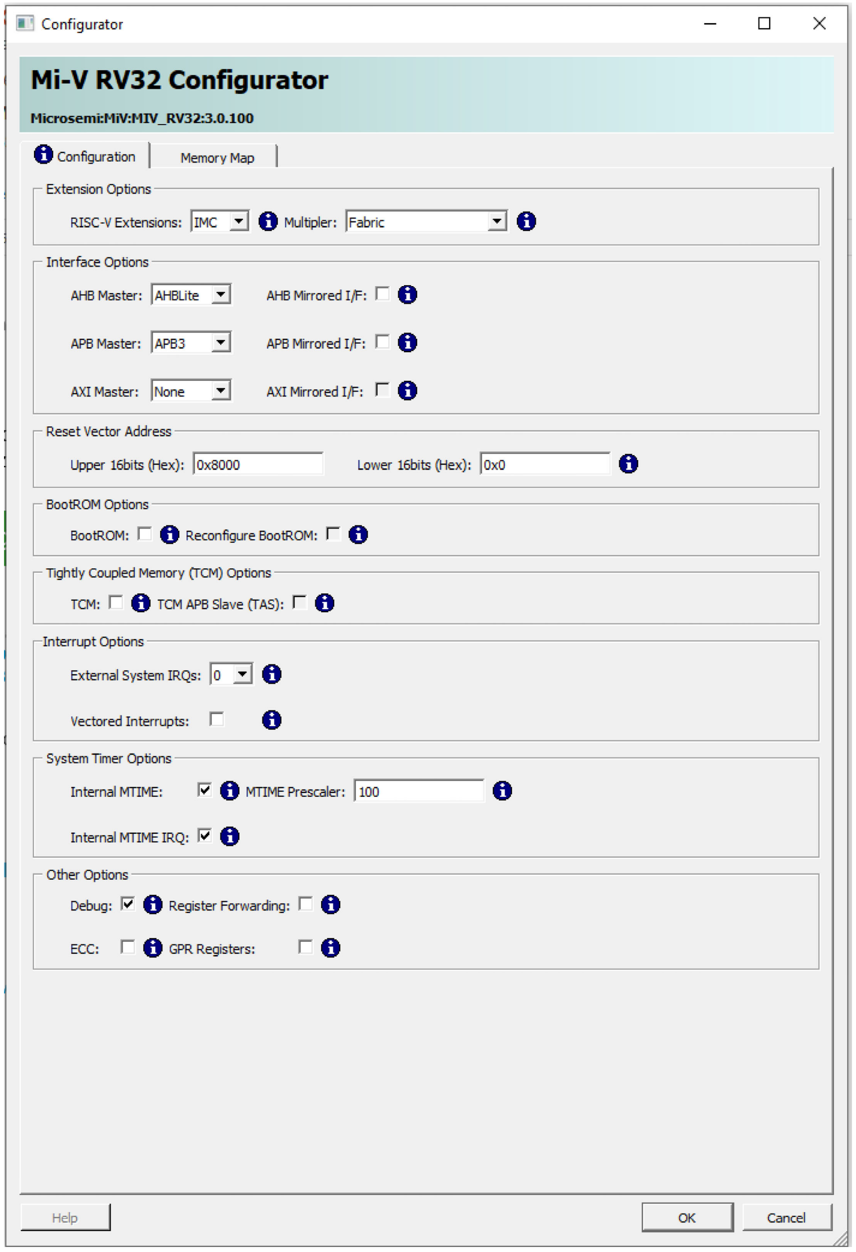


5.3 Configuring MIV_RV32

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 5-2. Configuration GUI for MIV_RV32 in SmartDesign



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Extension Options

RISC-V Extensions: IMC   Multiplier: Fabric

Interface Options

AHB Master: AHBLite   AHB Mirrored I/F: ☐

APB Master: APB3   APB Mirrored I/F: ☐

AXI Master: None   AXI Mirrored I/F: ☐

Reset Vector Address

Upper 16bits (Hex): 0x8000   Lower 16bits (Hex): 0x0

BootROM Options

BootROM: ☐   Reconfigure BootROM: ☐

Tightly Coupled Memory (TCM) Options

TCM: ☐   TCM APB Slave (TAS): ☐

Interrupt Options

External System IRQs: 0

Vectored Interrupts: ☐

System Timer Options

Internal MTIME: ✓   MTIME Prescaler: 100

Internal MTIME IRQ: ✓

Other Options

Debug: ✓   Register Forwarding: ☐

ECC: ☐   GPR Registers: ☐

5.3.1 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.

5.3.2 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.

5.3.3 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-ilgoo2.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.

5.3.4 Interrupt Options

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

5.3.5 TCM Options

Under the TCM Options heading, the option to enable TCM along with the additional option to enable a TAS is available. This option allows reading and writing to the TCM by an external core over the APB Interface. The TCM depth is limited to a maximum of 256 Kbytes. TCM depth is determined by the address space allocated in the interface memory mapping configuration.

5.3.6 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.

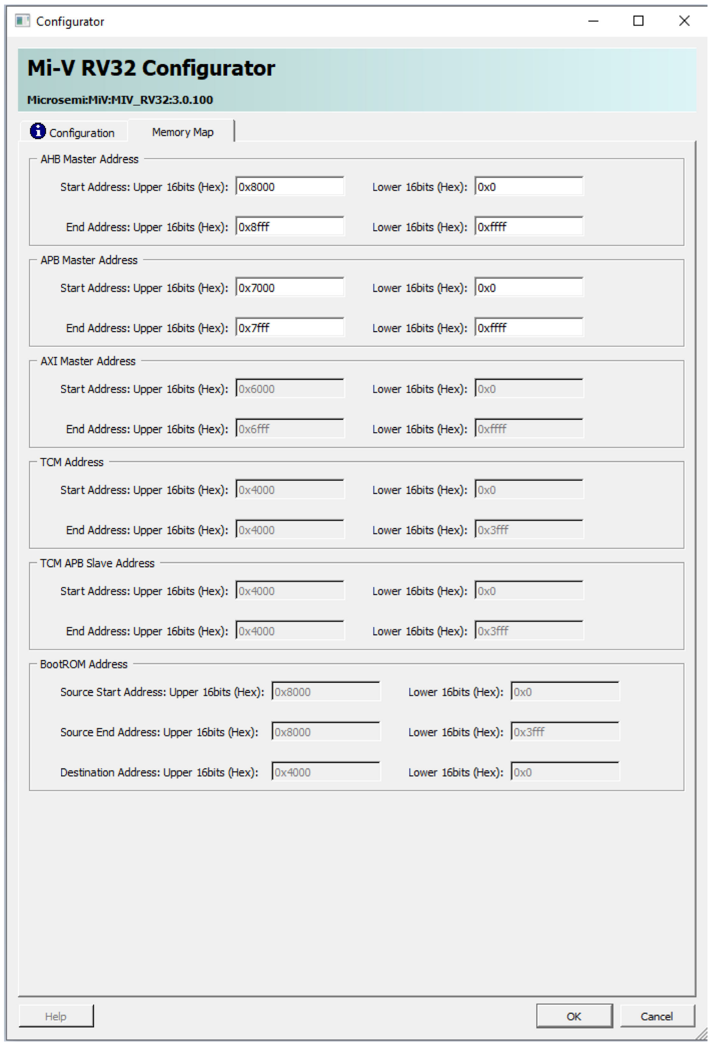
Option to enable ECC is available, which instantiates fabric ECC encoding and decoding logic for the TCM and GPRs, if they are RAM based.

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

5.3.7 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 5-3. Memory Map GUI for MIV_RV32 in SmartDesign



Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

AHB Master Address

Start Address: Upper 16bits (Hex):	0x8000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x8fff	Lower 16bits (Hex):	0xffff

APB Master Address

Start Address: Upper 16bits (Hex):	0x7000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x7fff	Lower 16bits (Hex):	0xffff

AXI Master Address

Start Address: Upper 16bits (Hex):	0x6000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x6fff	Lower 16bits (Hex):	0xffff

TCM Address

Start Address: Upper 16bits (Hex):	0x4000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x4000	Lower 16bits (Hex):	0x3fff

TCM APB Slave Address

Start Address: Upper 16bits (Hex):	0x4000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x4000	Lower 16bits (Hex):	0x3fff

BootROM Address

Source Start Address: Upper 16bits (Hex):	0x8000	Lower 16bits (Hex):	0x0
Source End Address: Upper 16bits (Hex):	0x8000	Lower 16bits (Hex):	0x3fff
Destination Address: Upper 16bits (Hex):	0x4000	Lower 16bits (Hex):	0x0

Help | OK | Cancel

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.

5.4 Debugging

The CoreJTAGDebug v3.1.100 or later, is used to enable debugging of MIV_RV32. This is available in the Libero Catalog.

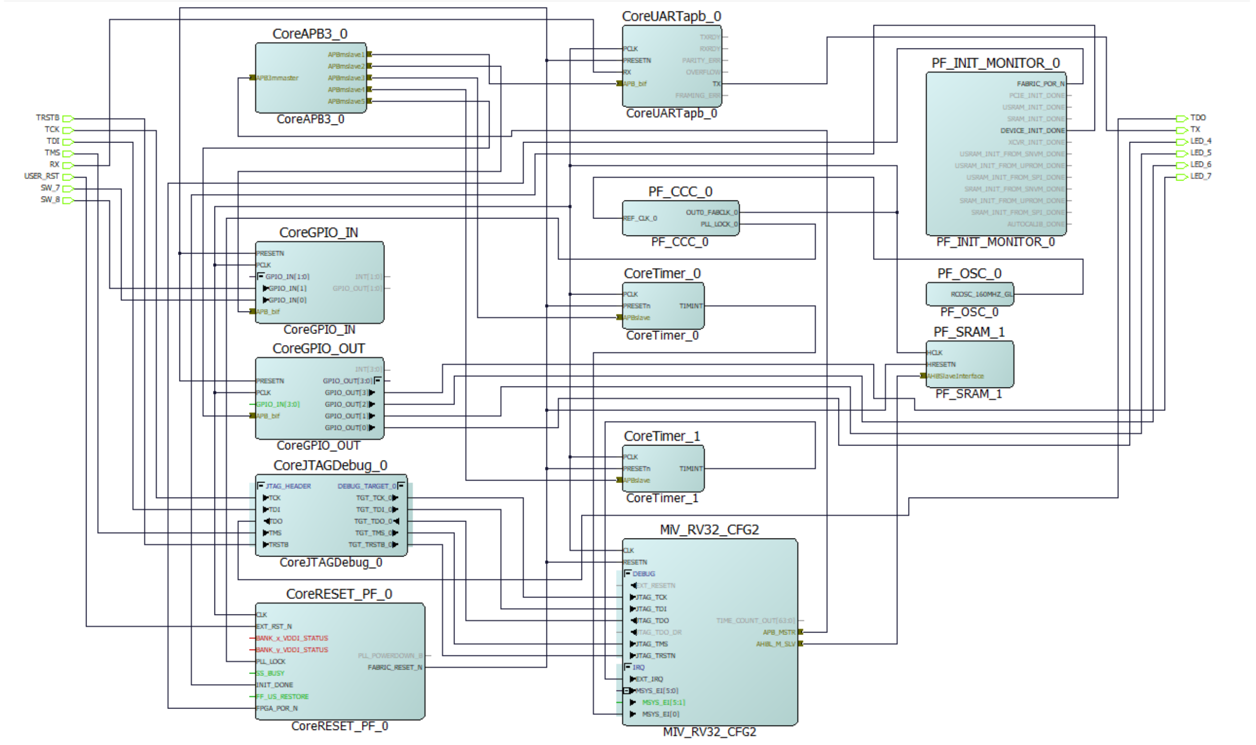
Note: Ensure that the TRST polarity is configured for active low operation as per the MIV_RV32 requirements.

5.5 Simulation Flows

The user testbench for MIV_RV32 is not included in this release.

The MIV_RV32 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 it 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 5-4. Example Subsystem



5.6 Synthesis in Libero

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

5.7 Place-and-Route in Libero

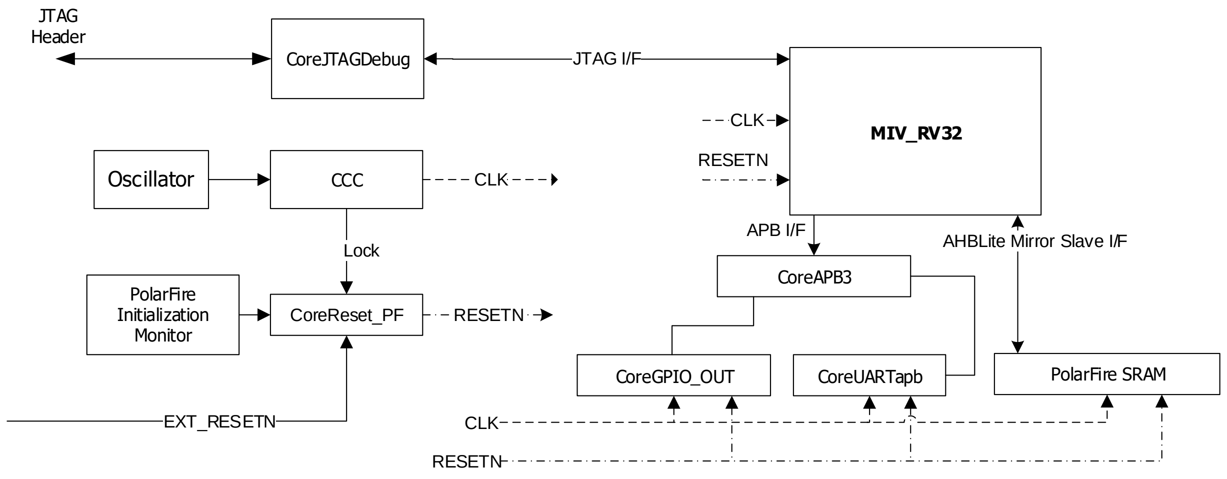
After the design is synthesized, run the compilation and the place-and-route tools. Click the **Layout** icon in Libero SoC to invoke designer.

6. System Integration

6.1 PolarFire Example System

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

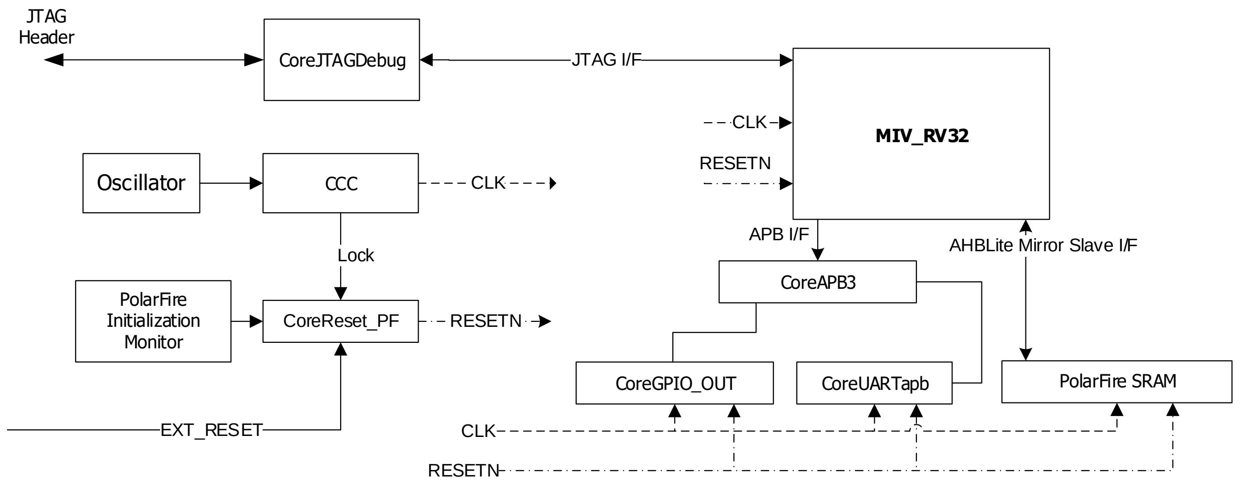
Figure 6-1. MIV_RV32: An Example System



6.2 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 6-2. MIV_RV32: An Example System for RTG4/SF2/IGL2



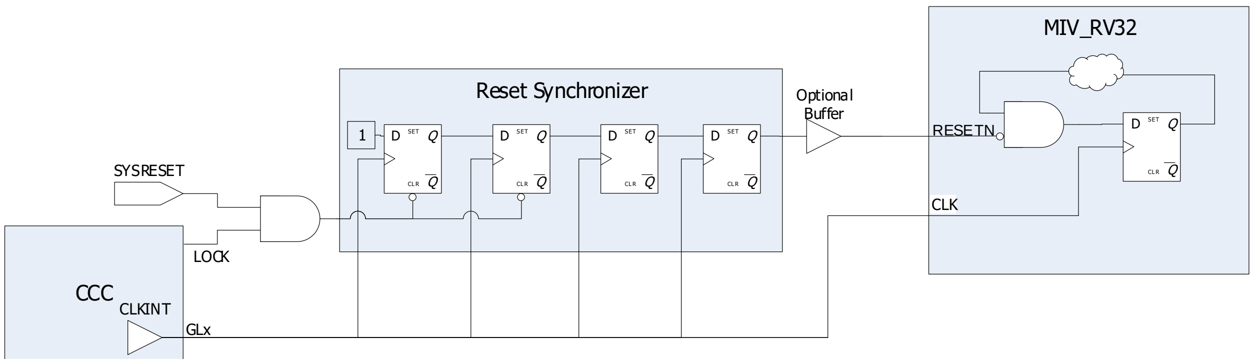
Note: The difference between the PolarFire and the other design is the use of the PolarFire Initialization Monitor and `CoreReset_PF`. The HDL reset synchronizer acts as the `CoreReset_PF`, as this IP is only available on PolarFire. For other families, the reset synchronizer is used. The HDL code for the reset synchronizer is available in 4.2 Reset Operation.

6.3 Reset Synchronization

6.3.1 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 Figure 6-2.

Figure 6-3. RESETN Reset Synchronization



The following Verilog code snippet implements the reset synchronizer block shown in Figure 5-1. 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_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
```

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.

6.3.2 JTAG_TRSTN

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.

7. 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 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 {async1}\
   -asynchronous\
   -group [ get_clocks {CCC_0_inst_0/CCC_0_0/pll_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.

8. SoftConsole

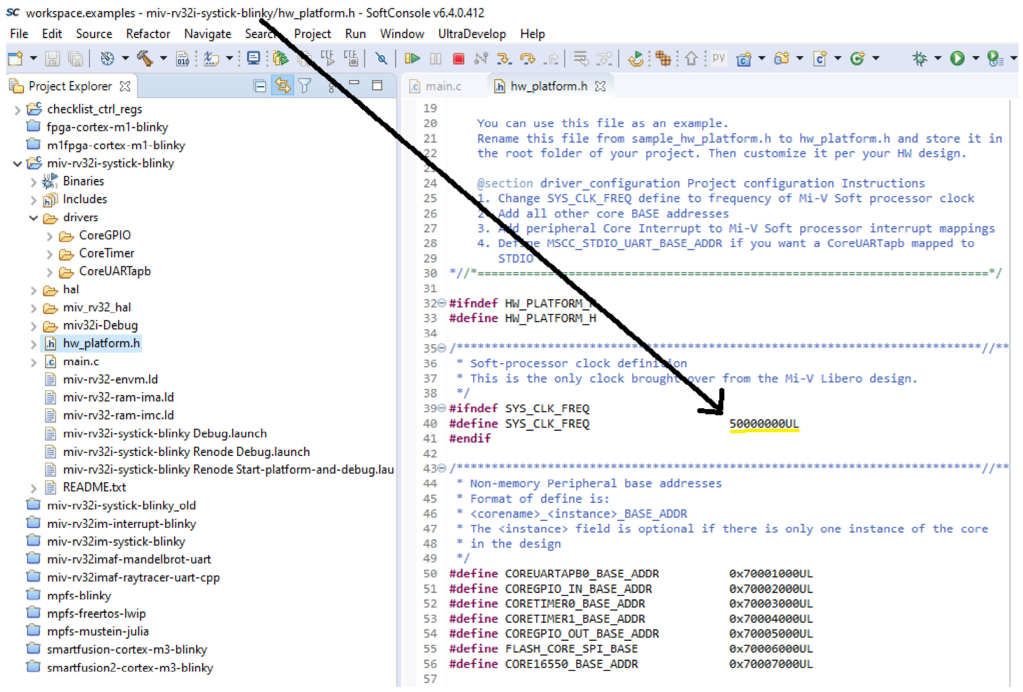
The SoftConsole version 6.4 or later is required to use the MIV_RV32. Each SoftConsole project requires the MIV_RV32 Hardware Abstraction Layer (HAL) version 3.0 or greater. For more information on setting up project for RISC-V, see the *SoftConsole release notes*. The following steps briefly explain the changes that might 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.

8.1 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 8-1. Setting Clock Frequency and Peripheral Base Addresses

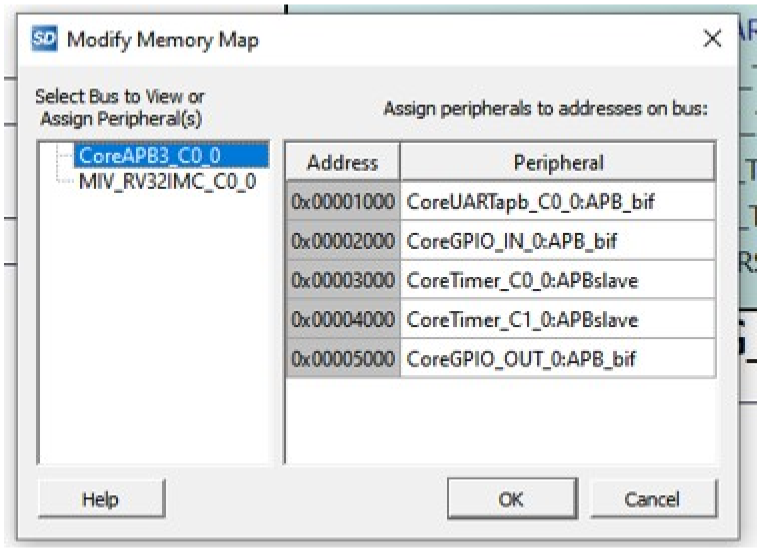

workspace.examples - miv-rv32i-systick-blinky/hw_platform.h - SoftConsole v6.4.0.412
File Edit Source Refactor Navigate Search Project Run Window UltraDevelop Help
Project Explorer
checklist_ctrl_regs
fpga-cortex-m1-blinky
m1fpga-cortex-m1-blinky
miv-rv32i-systick-blinky
Binaries
Includes
drivers
CoreGPIO
CoreTimer
CoreUARTapb
hal
miv_rv32_hal
miv32i-Debug
hw_platform.h
main.c
miv-rv32-envm.ld
miv-rv32-ram-ima.ld
miv-rv32-ram-imc.ld
miv-rv32i-systick-blinky Debug.launch
miv-rv32i-systick-blinky Renode Debug.launch
miv-rv32i-systick-blinky Renode Start-platform-and-debug.lau
README.txt
miv-rv32i-systick-blinky_old
miv-rv32im-interrupt-blinky
miv-rv32im-systick-blinky
miv-rv32imaf-mandelbrot-uart
miv-rv32imaf-raytracer-uart-cpp
mpfs-blinky
mpfs-freertos-lwip
mpfs-mustein-julia
smartfusion-cortex-m3-blinky
smartfusion2-cortex-m3-blinky
main.c
hw_platform.h

```
    You can use this file as an example.
    Rename this file from sample_hw_platform.h to hw_platform.h and store it in
    the root folder of your project. Then customize it per your HW design.

    @section driver_configuration Project configuration Instructions
    1. Change SYS_CLK_FREQ define to frequency of Mi-V Soft processor clock
    2. Add all other core BASE addresses
    3. Add peripheral Core Interrupt to Mi-V Soft processor interrupt mappings
    4. Define MSCC_STDIO_UART_BASE_ADDR if you want a CoreUARTapb mapped to
       STDIO
*//*==========================================================================*/

#ifndef HW_PLATFORM_H
#define HW_PLATFORM_H

/****************************************************************************//**
 * Soft-processor clock definition
 * This is the only clock brought over from the Mi-V Libero design.
 */
#ifndef SYS_CLK_FREQ
#define SYS_CLK_FREQ                        50000000UL
#endif

/****************************************************************************//**
 * Non-memory Peripheral base addresses
 * Format of define is:
 * <corename>_<instance>_BASE_ADDR
 * The <instance> field is optional if there is only one instance of the core
 * in the design
 */
#define COREUARTAPB0_BASE_ADDR              0x70001000UL
#define COREGPIO_IN_BASE_ADDR               0x70002000UL
#define CORETIMER0_BASE_ADDR                0x70003000UL
#define CORETIMER1_BASE_ADDR                0x70004000UL
#define COREGPIO_OUT_BASE_ADDR              0x70005000UL
#define FLASH_CORE_SPI_BASE                 0x70006000UL
#define CORE16550_BASE_ADDR                 0x70007000UL
```

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.

Note: Some versions of Libero do not support the memory map feature as shown in the following figure.

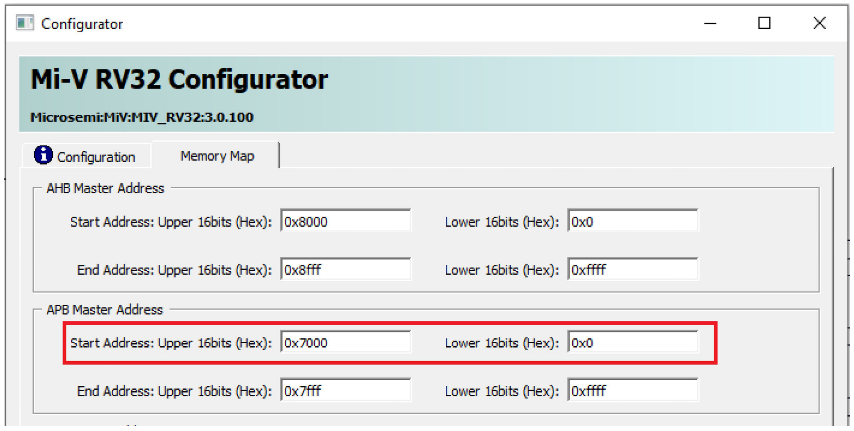
Figure 8-2. Modify Memory Map Dialog


Modify Memory Map
Select Bus to View or Assign Peripheral(s)
CoreAPB3_C0_0
MIV_RV32IMC_C0_0
Assign peripherals to addresses on bus:
Help
OK
Cancel

Address	Peripheral
0x00001000	CoreUARTapb_C0_0:APB_bif
0x00002000	CoreGPIO_IN_0:APB_bif
0x00003000	CoreTimer_C0_0:APBslave
0x00004000	CoreTimer_C1_0:APBslave
0x00005000	CoreGPIO_OUT_0:APB_bif

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 MIV_RV32 core redirects these addresses accordingly. The following figure shows the MIV_RV32 configuration settings for peripherals.

Figure 8-3. RV32 Configurator Memory Map



AHB Master Address

Start Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x8fff Lower 16bits (Hex): 0xffff

APB Master Address

Start Address: Upper 16bits (Hex): 0x7000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x7fff Lower 16bits (Hex): 0xffff

9. Revision History

Revision	Date	Description
A	October 2020	Following is the summary of changes: • Document formatted to Microchip template and DS number assigned. • Document updated with the configuration neutral core name MIV_RV32 to allow for future expansion support for additional RISC-V ISA extension options.
1.0	March 2020	This is the first publication of the MIV_RV32IMC IP.

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-7047-2

Quality Management System

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820