MIV_RV32 Migration Guide

Introduction

This document describes the Libero® SoC design migration process for Mi-V Soft Processors. The legacy `CoreRISCV_AXI4`, `MIV_RV32IMA_L1_AHB`, `MIV_RV32IMA_L1_AXI`, and `MIV_RV32IMAF_L1_AHB` soft processor cores are to be replaced with a single highly-configurable `MIV_RV32` soft processor core. The objective of this document is to ease the customer Hardware (HW) and Firmware (FW) migration process to the `MIV_RV32` platform.

In this document, the `CoreRISCV_AXI4`, `MIV_RV32IMA_L1_AHB`, `MIV_RV32IMA_L1_AXI`, and `MIV_RV32IMAF_L1_AHB` soft processor cores are collectively referred to as `MIV_Legacy` cores. The `MIV_RV32IMC v2.1.100` and `MIV_RV32 v3.0.100` or greater are collectively referred to as `MIV_RV32`, unless otherwise stated.

Table of Contents

1. Reasons to Migrate

The following sections explain why an existing Libero SoC and accompanying Software design should be migrated to `MIV_RV32`.

1.1 MIV_RV32 Core

The `CoreRISCV_AXI4` core is no longer recommended for new designs. The `MIV_RV32 IMA_L1_AHB/MIV_RV32 IMA_L1_AXI/MIV_RV32 IMAF_L1_AHB` Mi-V cores are minimally configurable, and the unused features are left in place during post Synthesis. If the requirement is low resource and medium performance without the need for cache, the `MIV_RV32` core should be used. Where cache is required, the `MIV_Legacy` core should be retained. The `MIV_RV32` will be enhanced over time to supersede the `MIV_Legacy` cores.

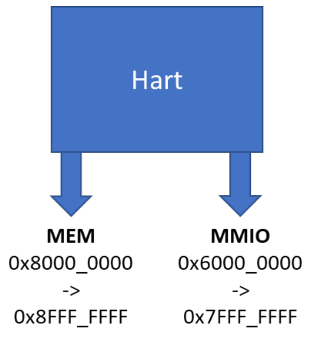
1.2 MIV_RV32 HAL

MIV_RV32 HAL v3.0 or greater contains bug fixes and adds support for the `MIV_RV32` core.

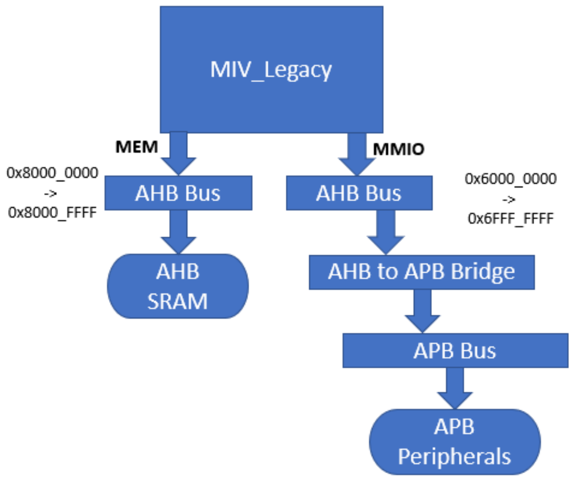
2. Migrating Hardware Configurations

Migrating designs from a `MIV_Legacy` configuration to a `MIV_RV32` configuration is relatively straight forward. The `MIV_Legacy` cores have a fixed memory map based on their Hardware Architecture. They use the MEM interface for cached instructions and data, and the MMIO interface for peripherals and non-cached memory.

The following figure shows the fixed memory map of the `MIV_Legacy` cores.



The following figure shows a typical system.



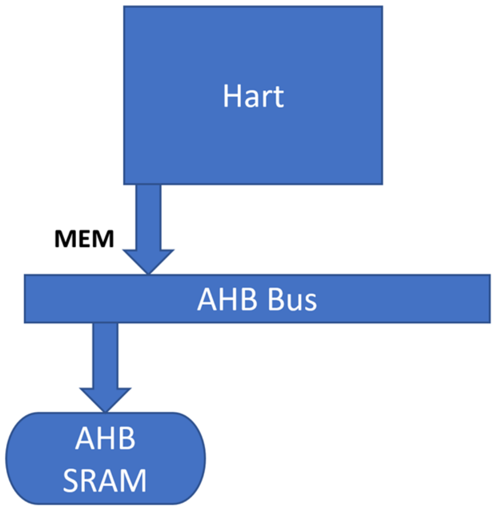
2.1 Peripherals Connected to the MEM Interface

The primary function of the MEM interface in a `MIV_Legacy` core is to allow cached access to software code and data. It can be connected to SRAM embedded within the FPGA or to discrete DDR memory devices. The MEM interface has a restricted address range on the `MIV_Legacy` cores from `0x8000_0000` to `0x8FFF_FFFF`. The `MIV_RV32` core does not feature a cache, instead it features a Tightly Coupled Memory (TCM).

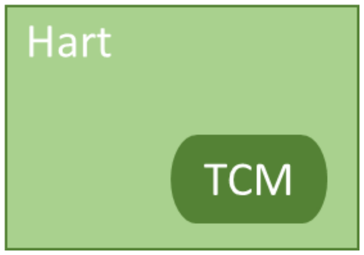
Note: The TCM must be used in preference to SRAM in systems where the processor requires faster memory accesses. The address range for the MEM interface on the `MIV_RV32` core is much less restrictive and is described in the following sections.

2.1.1 SRAM

The typical use of the MEM interface is interfacing a memory. When using SRAM, the configuration can be an AHB or an AXI as shown in the following figure.



The TCM in the `MIV_RV32` core operates in the same way as external SRAM, except with lower latency due to it being internally coupled to the core.

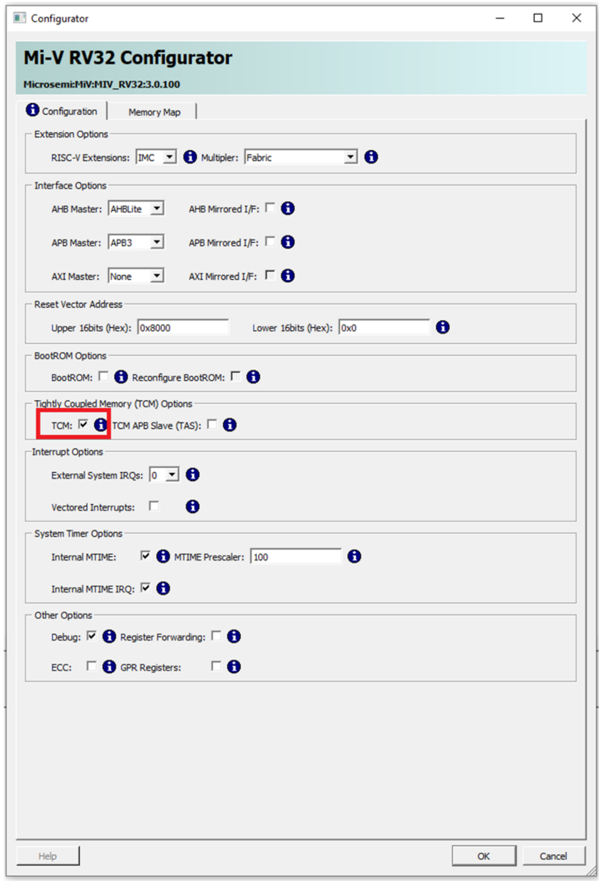


2.1.1.1 Configuring the TCM

Review the maximum size of TCM available in the relevant *MIV_RV32 Handbook*. The TCM must have a start address greater than 0x1000_0000.

Note: The TCM on the MIV_R32 is limited to a maximum size of 256 Kbytes in v3.0.x.

The TCM is enabled from the **Configuration** tab of the **Configuration** window.



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

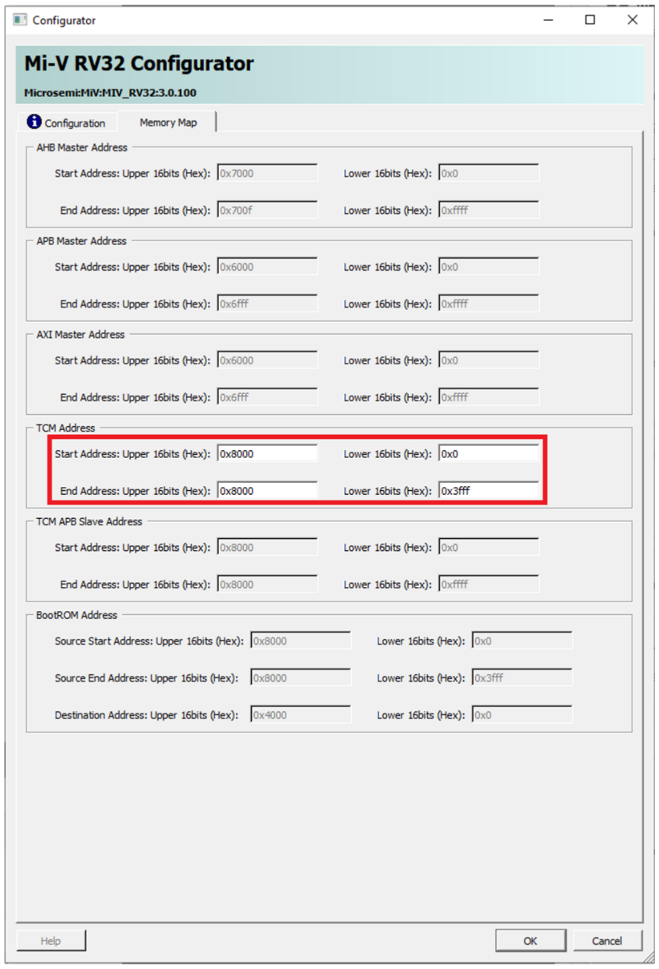
Internal MTIME IRQ: ☑

Other Options

Debug: ☑ Register Forwarding: ☐

ECC: ☐ GPR Registers: ☐

The depth of the TCM, up to the maximum defined TCM size, is calculated from its accessible range in the **Memory Map** tab of the **Configurator** window.



AHB Master Address

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

End Address: Upper 16bits (Hex): 0x700f Lower 16bits (Hex): 0xffff

APB Master Address

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

End Address: Upper 16bits (Hex): 0x6fff 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): 0x8000 Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x3fff

TCM APB Slave Address

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

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

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

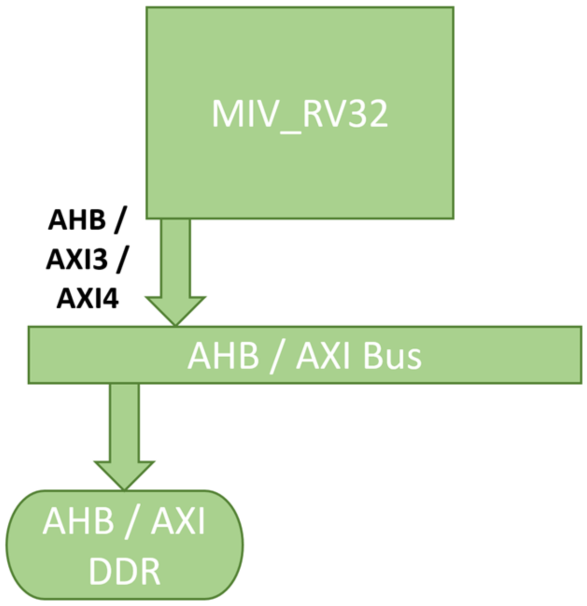
The following table gives examples of TCM address widths and their corresponding memory depth.

Start Address	End Address	Depth	# 32-Bit Words	Kbytes
0x8000_0000	0x8000_03FF	0x400	256	1
0x8000_0000	0x8000_07FF	0x800	512	2
0x8000_0000	0x8000_0FFF	0x1000	1024	4
0x8000_0000	0x8000_1FFF	0x2000	2048	8
0x8000_0000	0x8000_3FFF	0x4000	4096	16
0x8000_0000	0x8000_7FFF	0x8000	8192	32
0x8000_0000	0x8000_FFFF	0x1_0000	16384	64
0x8000_0000	0x8001_FFFF	0x2_0000	32768	128

2.1.2 DDR

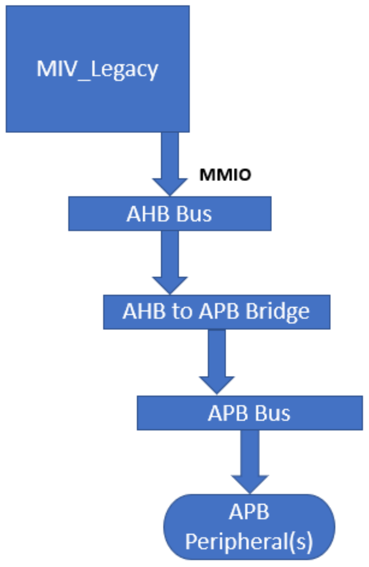
DDR can be used as external memory available to the core. As the core features AHB and AXI3/AXI4 interfaces with no addressing restrictions, except a start address greater than 0x1000_0000, the DDR can be connected to either of these interfaces depending on the slave interface type and the accessible range given in the **Memory** tab.

The MIV_RV32 data interfaces are 32 bits wide. In many instances DDR can require 64-bit data access. In this case, an IP core such as CoreAXI4Interrconnect can be used to provide data width conversion for DDR memory. It should be noted that MIV_RV32 does not feature an L1 cache and as such AXI burst transactions are not available. In this instance, careful consideration should be given before migrating to MIV_RV32 as performance with DDR will be limited.

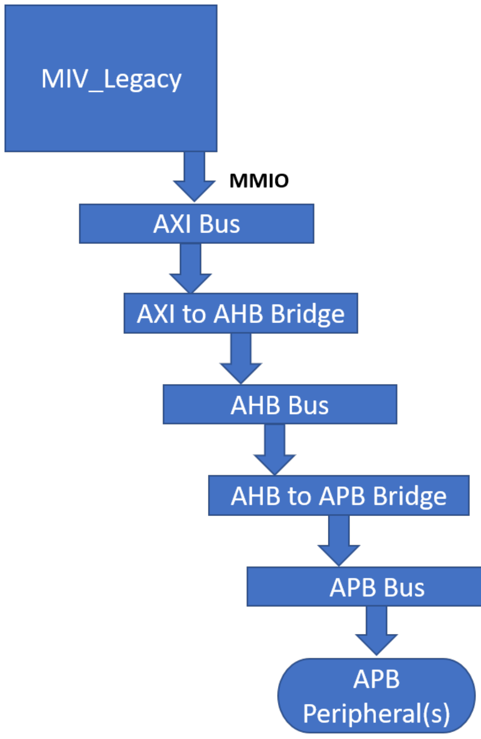


2.2 Peripherals Connected to the MMIO Interface

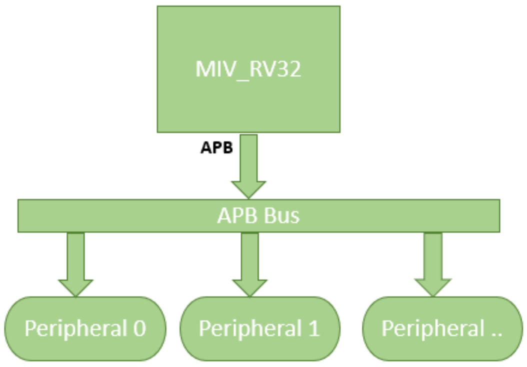
Each peripheral connected to a Mi-V Legacy core has an APB interface and is connected to an APB bus. As the MIV_Legacy core does not have an APB interface, use the APB bus bridges as shown in the following figure.



The following figure shows the bridges required to convert from AXI to APB when using a MIV_Legacy core.

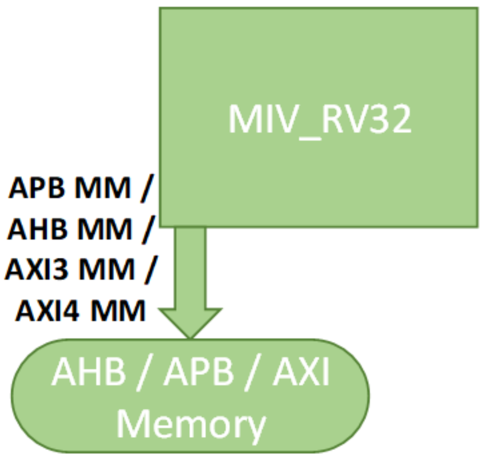


The following figure shows that as `MIV_RV32` features an APB interface, no conversion is required.

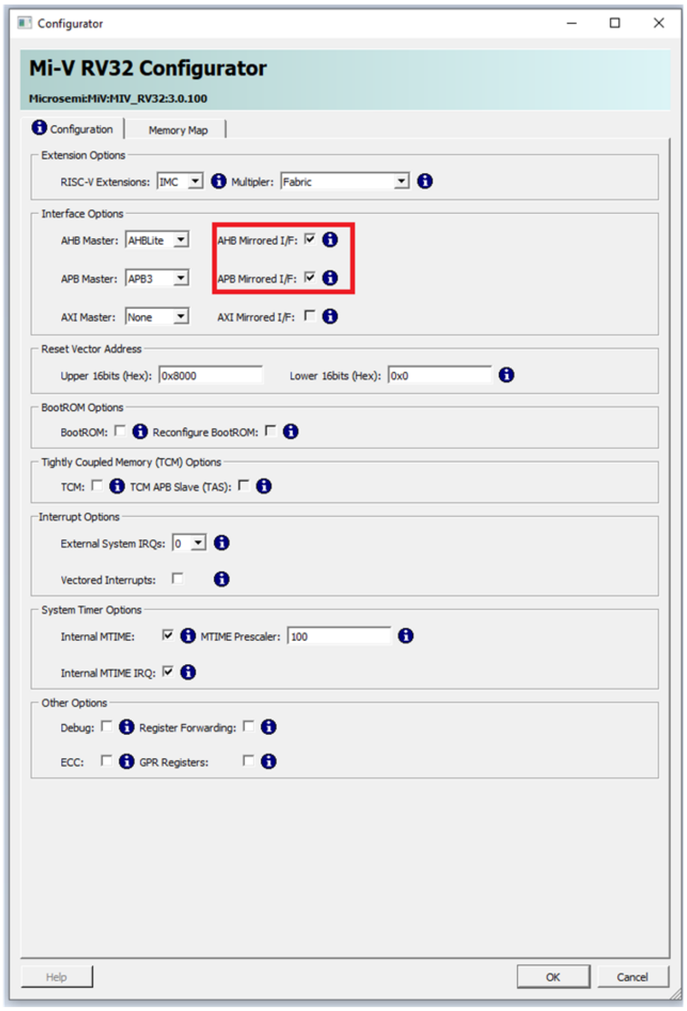


2.3 Mirrored Master Interfaces

If `MIV_RV32` is the only core that is going to access a memory or a peripheral and there are no additional peripherals connected on the interface, the Mirrored Master mode can be selected to allow a direct connection. It improves performance and reduces area as a bus master is not used.



It can be enabled by selecting the Mirrored Master options under the **Interface Options** in the **Configurator** window.


Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

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: ☐

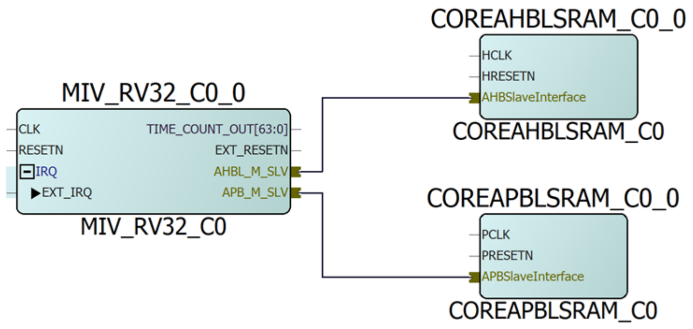
Tightly Coupled Memory (TCM) Options
TCM: ☐ TCM APB Slave (TAS): ☐

Interrupt Options
External System IRQs: 0
Vectored Interrupts: ☐

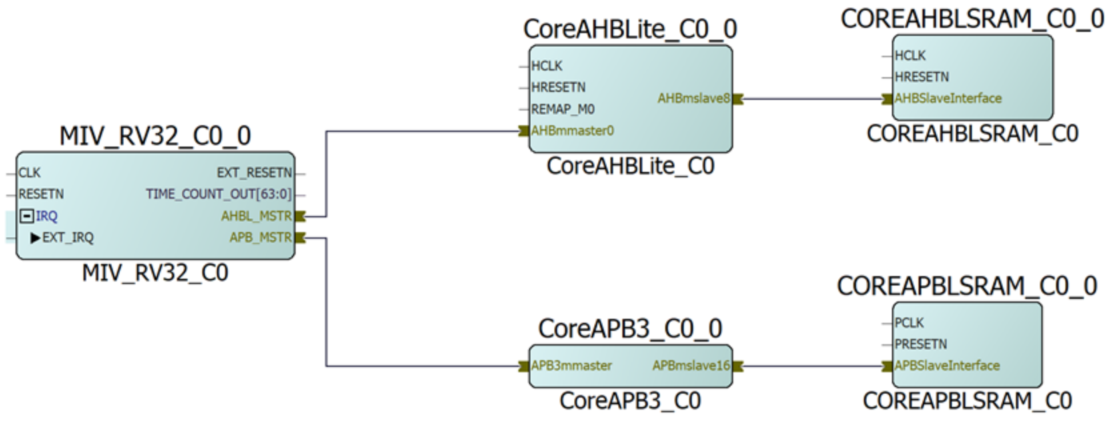
System Timer Options
Internal MTIME: ☑ MTIME Prescaler: 100
Internal MTIME IRQ: ☑

Other Options
Debug: ☐ Register Forwarding: ☐
ECC: ☐ GPR Registers: ☐

For example, the following figure shows APB and AHB SRAMs connected directly to the `MIV_RV32` using the Mirrored Master configuration. After place-and-route, the following design has used 4774 logic elements.



The following figure is the equivalent design, without the mirrored masters. After place-and-route, the following design has used 4927 logic elements.



2.4 Interfaces and Memory Maps

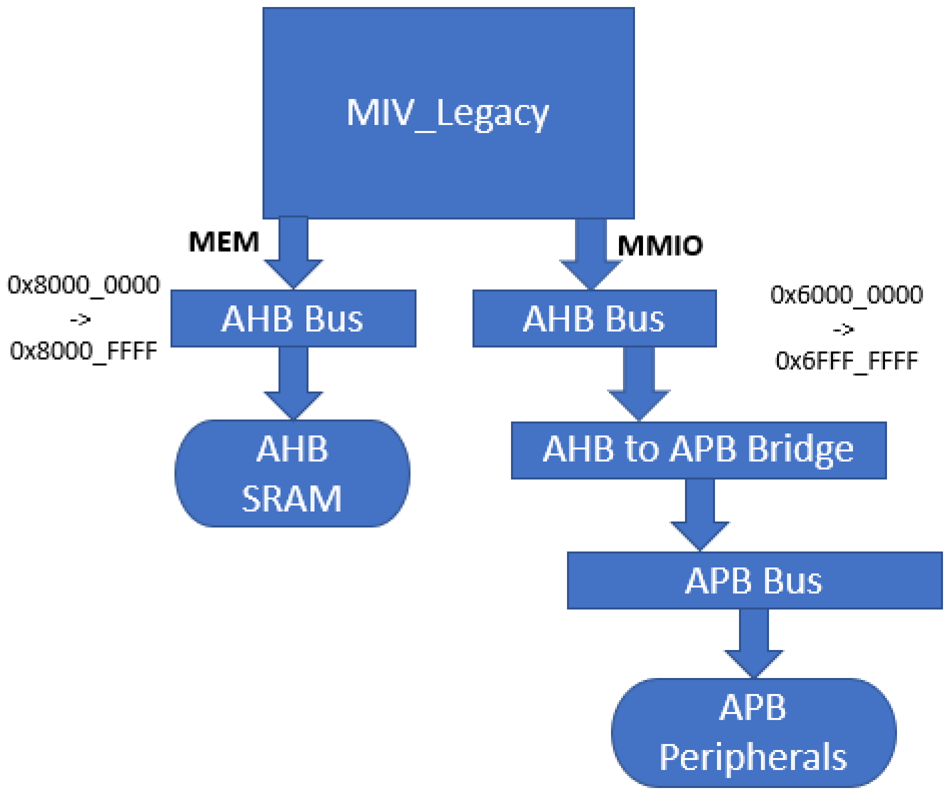
When migrating a design from an `MIV_Legacy` core to an `MIV_RV32` core, there are several ways to configure the updated design to retain the functionality of the original, while taking advantage of the benefits of the `MIV_RV32` core.

Sample designs are shown in the following sections, featuring an AHB as the primary configuration. The same configurations can be applied to the AXI cores as well.

2.4.1 Sample Design 1 – Base Design

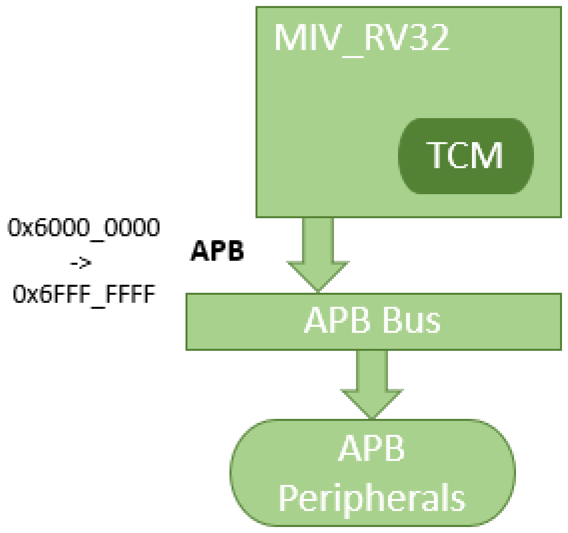
2.4.1.1 MIV_Legacy Configuration

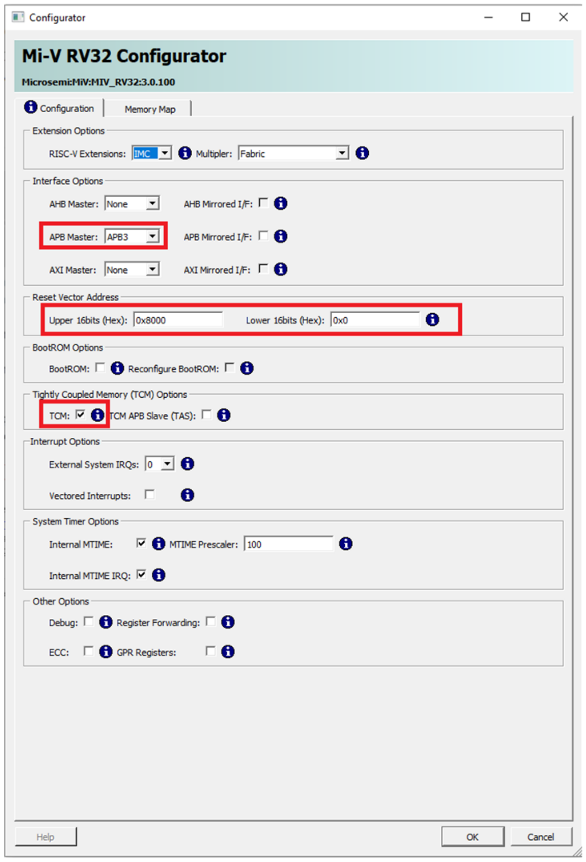
Memory is connected to the MEM interface at 0x8000_0000. Peripherals are connected to the MMIO interface at 0x6000_0000.



2.4.1.2 MIV_RV32 Configuration

TCM is enabled and set to start at 0x8000_0000 and end at 0x8000_FFFF. The APB bus is enabled and configured to start at 0x6000_0000 and end at 0x6FFF_FFFF. The configuration settings for this example are shown in the following figures.





Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Extension Options

RISC-V Extensions: IMC   Multipler: Fabric

Interface Options

AHB Master:	None	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

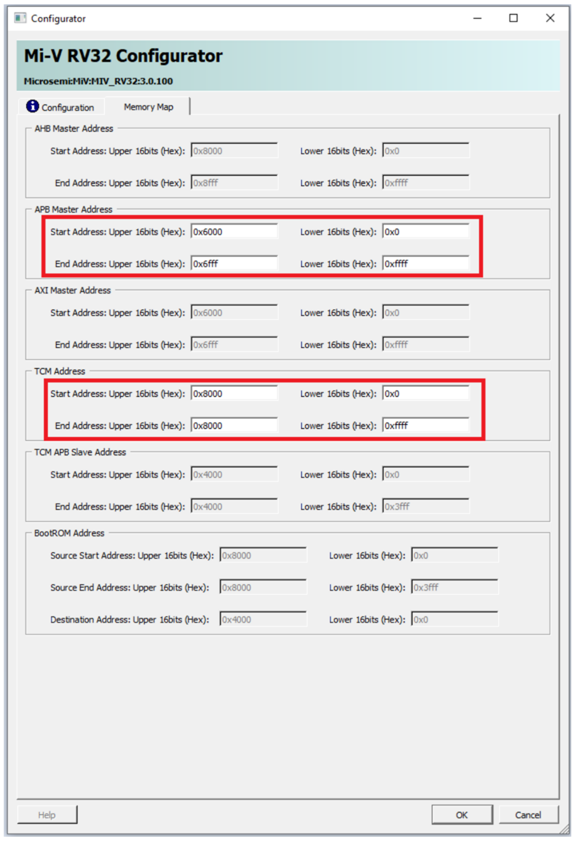
Internal MTIME: ☑   MTIME Prescaler: 100

Internal MTIME IRQ: ☑

Other Options

Debug: ☐   Register Forwarding: ☐

ECC: ☐   GPR Registers: ☐



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

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): 0x6000   Lower 16bits (Hex): 0x0

End Address: Upper 16bits (Hex): 0x6fff   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): 0x8000   Lower 16bits (Hex): 0x0

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

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

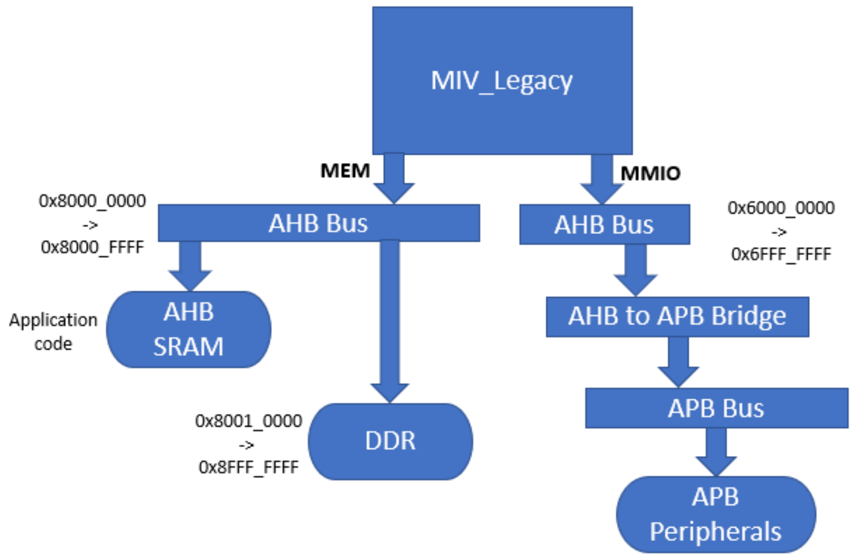
Source End Address: Upper 16bits (Hex): 0x8000   Lower 16bits (Hex): 0x3fff

Destination Address: Upper 16bits (Hex): 0x4000   Lower 16bits (Hex): 0x0

2.4.2 Sample Design 2 – Base Design with DDR

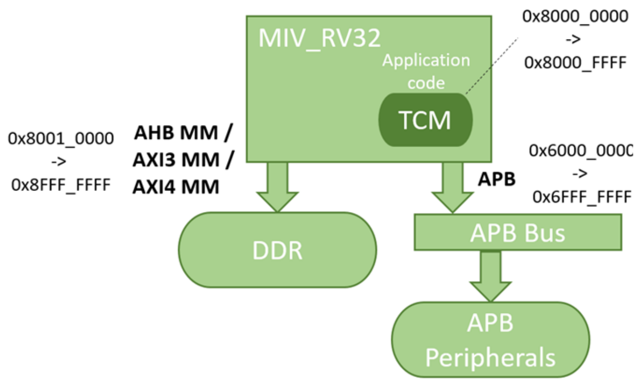
2.4.2.1 MIV_Legacy Configuration

SRAM is connected to the MEM interface at 0x8000_0000 with DDR connected at 0x8001_0000. Peripherals are connected to the MMIO interface at 0x6000_0000.

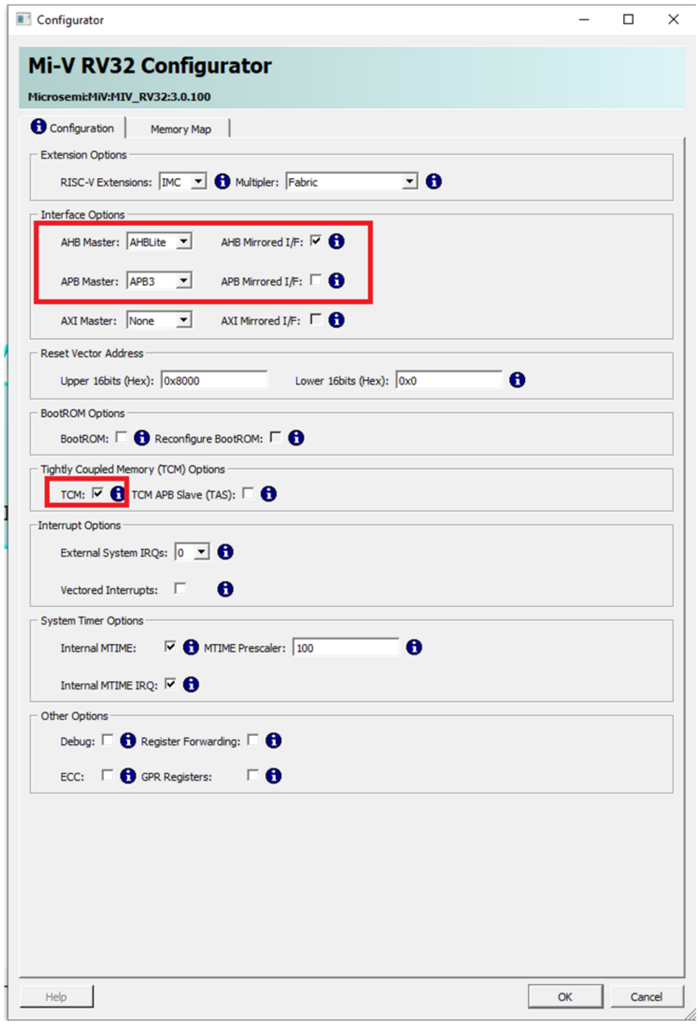


2.4.2.2 MIV_RV32 Configuration

TCM is enabled and set to a range from 0x8000_0000 to 0x8000_FFFF to run the application code. The AHB or AXI interfaces can be used to access the DDR in mirrored master mode with a range from 0x8001_0000 to 0x8FFF_FFFF. The APB interface is enabled with a range from 0x6000_0000 to 0x6FFF_FFFF.



By enabling the AHB master, it allows the AHB Master address fields of the memory map tab to be edited. The same applies to the APB and AXI masters along with the TCM.



Configurator

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

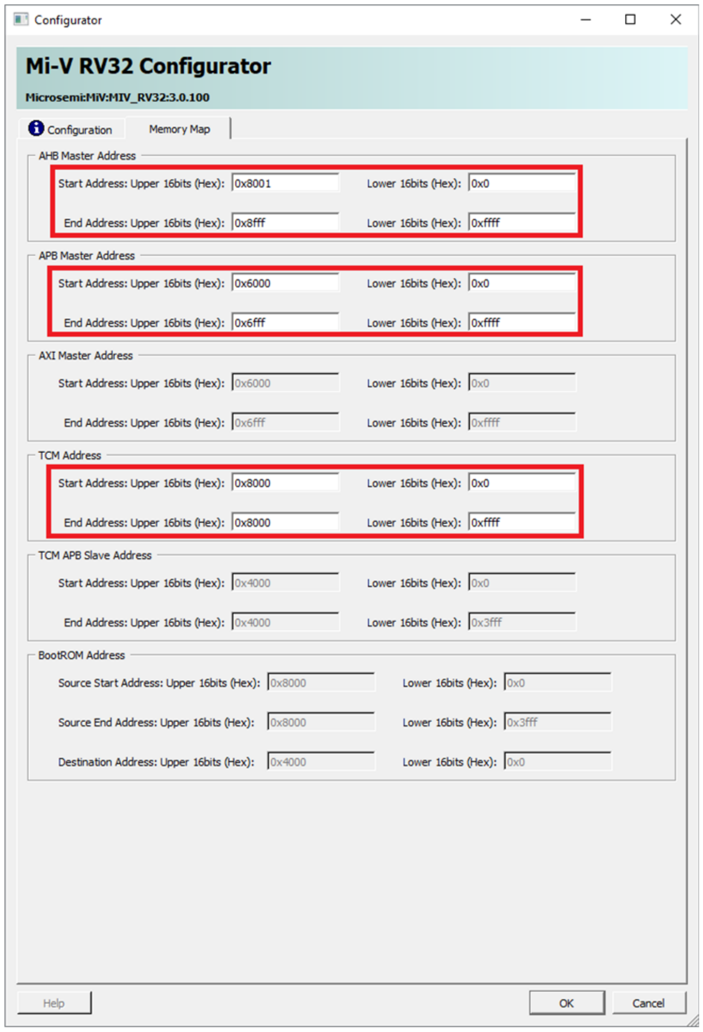
Internal MTIME: ✓ MTIME Prescaler: 100

Internal MTIME IRQ: ✓

Other Options

Debug: ☐ Register Forwarding: ☐

ECC: ☐ GPR Registers: ☐



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

AHB Master Address

Start Address: Upper 16bits (Hex):	0x8001	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x8fff	Lower 16bits (Hex):	0xffff

APB Master Address

Start Address: Upper 16bits (Hex):	0x6000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x6fff	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):	0x8000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x8000	Lower 16bits (Hex):	0xffff

TCM APB Slave Address

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

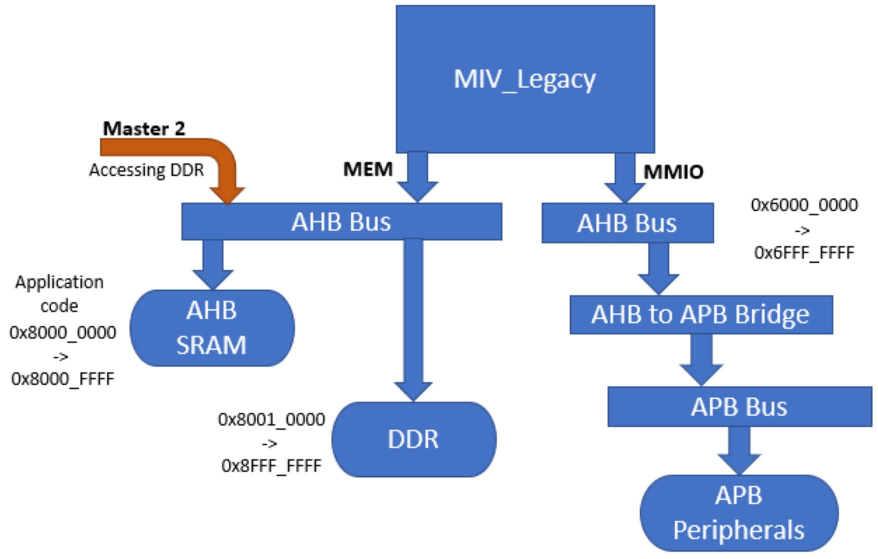
BootROM Address

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

2.4.3 Sample Design 3 – Base Design with DDR and a Second Master

2.4.3.1 MIV_Legacy Configuration

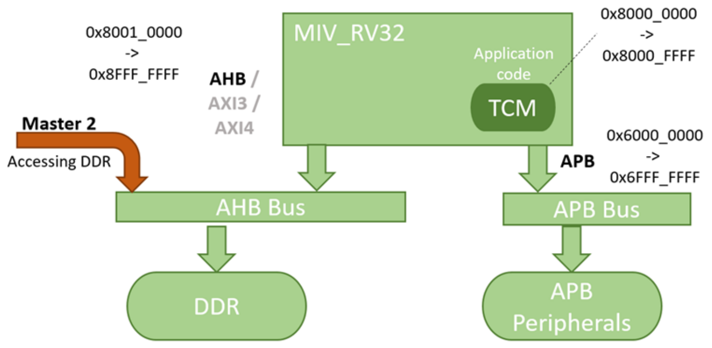
SRAM is connected to the MEM interface at 0x8000_0000 with DDR connected at 0x8001_0000. Peripherals are connected to the MMIO interface at 0x6000_0000. Master 2 is connected to the AHB bus used by the MEM interface accessing DDR.

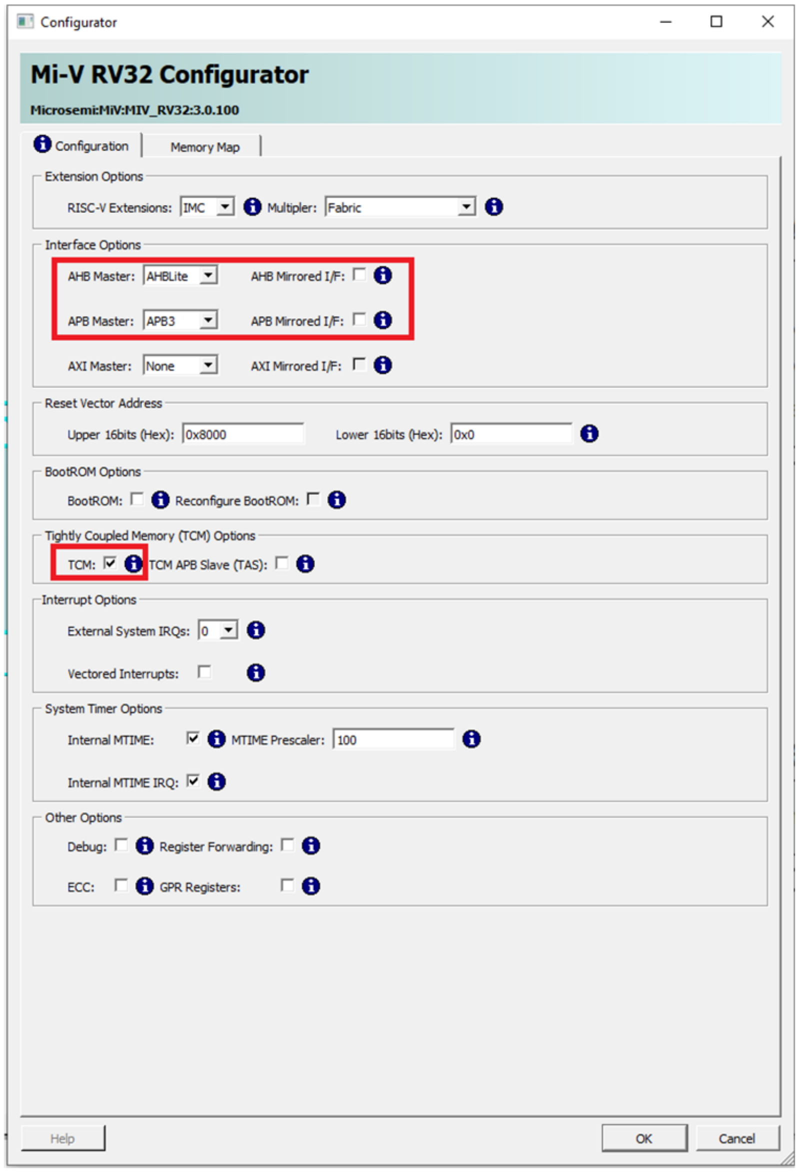


2.4.3.2 MIV_RV32 Configuration

TCM is enabled and set to a range from 0x8000_0000 to 0x8000_FFFF to run the application code. The AHB or AXI interfaces can be used to access the AHB bus and DDR with a range from 0x8001_0000 to 0x8FFF_FFFF. The APB interface is enabled with a range from 0x6000_0000 to 0x6FFF_FFFF. Master 2 can access DDR through the AHB bus.

The block diagram and configuration windows show how to enable this setup.





Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Extension Options

RISC-V Extensions: IMC Multipler: 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

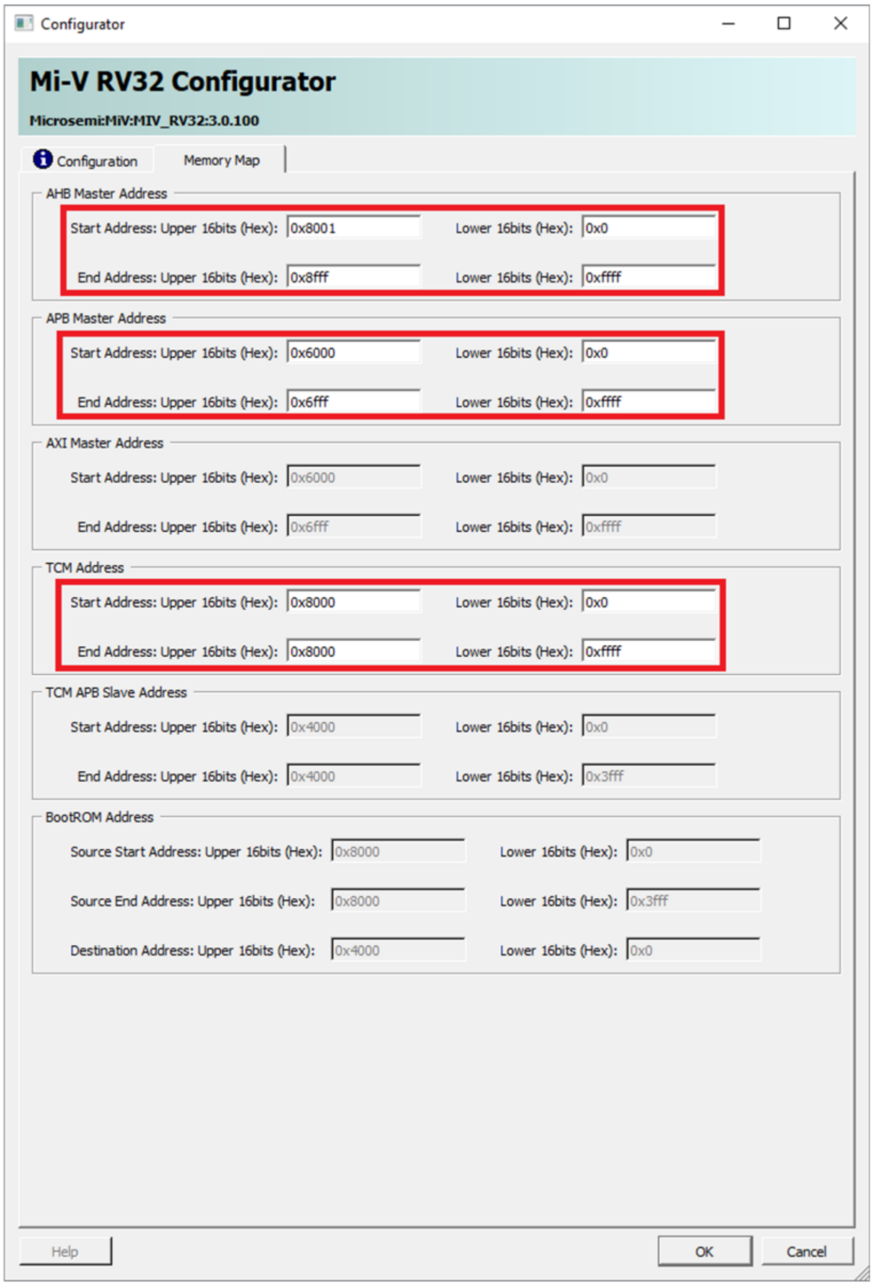
Internal MTIME: ☑ MTIME Prescaler: 100

Internal MTIME IRQ: ☑

Other Options

Debug: ☐ Register Forwarding: ☐

ECC: ☐ GPR Registers: ☐



Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

AHB Master Address

Start Address: Upper 16bits (Hex):	0x8001	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x8fff	Lower 16bits (Hex):	0xffff

APB Master Address

Start Address: Upper 16bits (Hex):	0x6000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x6fff	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):	0x8000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x8000	Lower 16bits (Hex):	0xffff

TCM APB Slave Address

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

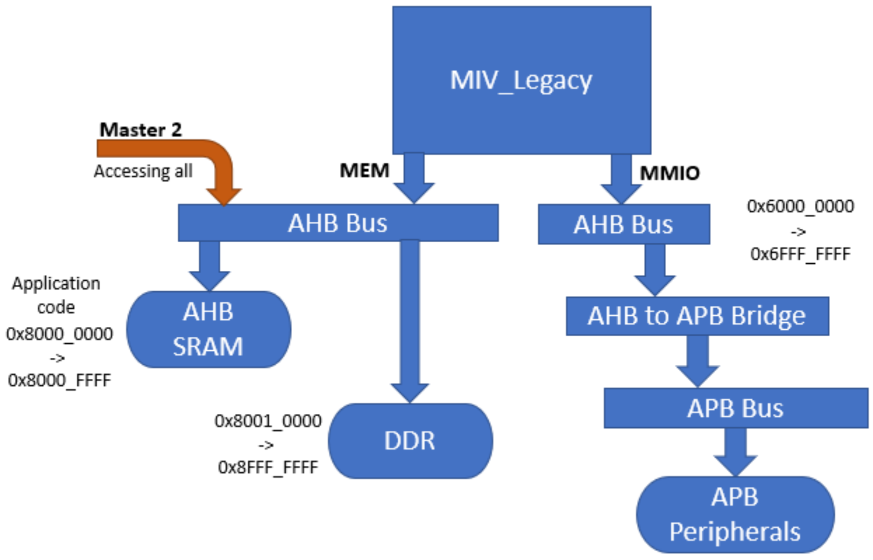
BootROM Address

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

2.4.4 Sample Design 4 - Base Design with DDR and Second Master

2.4.4.1 MIV_Legacy Configuration

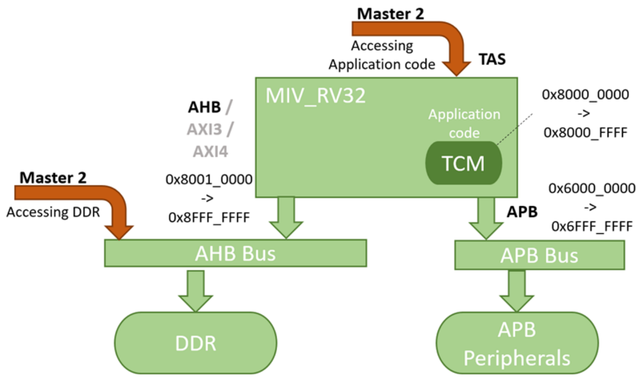
SRAM is connected to the MEM interface at 0x8000_0000 with DDR connected at 0x8001_0000. Peripherals are connected to the MMIO interface at 0x6000_0000. Master 2 is connected to the cached MEM AHB bus accessing the application code and DDR.

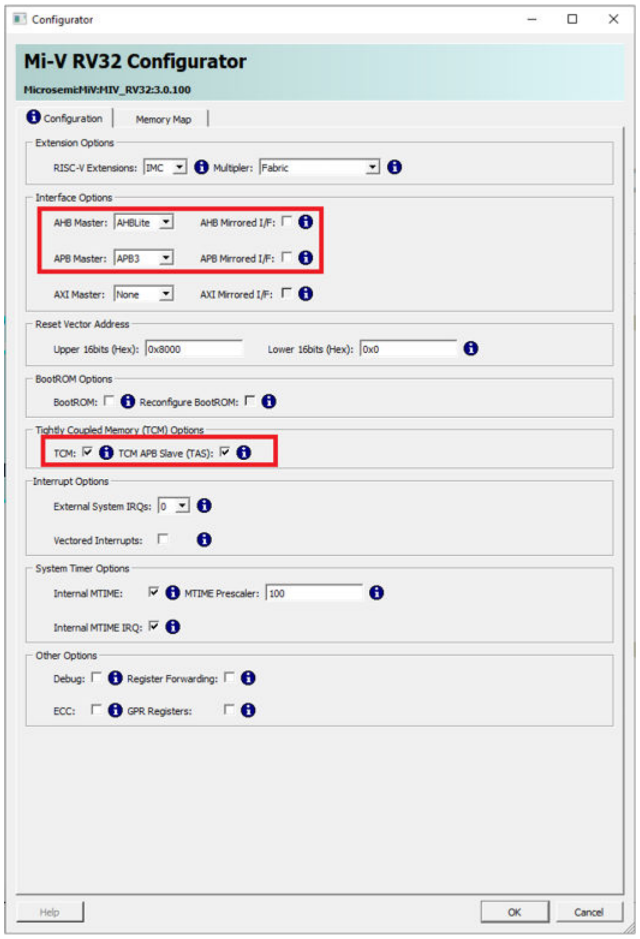


2.4.4.2 MIV_RV32 Configuration

TCM is enabled and set to a range from 0x8000_0000 to 0x8000_FFFF to run the application code. The AHB or AXI interfaces can be used to access the AHB bus and the DDR with a range from 0x8001_0000 to 0x8FFF_FFFF. The APB interface is enabled with a range from 0x6000_0000 to 0x6FFF_FFFF. Master 2 can access the DDR through the AHB bus and can access the application code in TCM using the TAS interface; making this change requires an APB master interface on Master 2.

The following block diagram and configuration windows show how to enable this setup.




Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

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

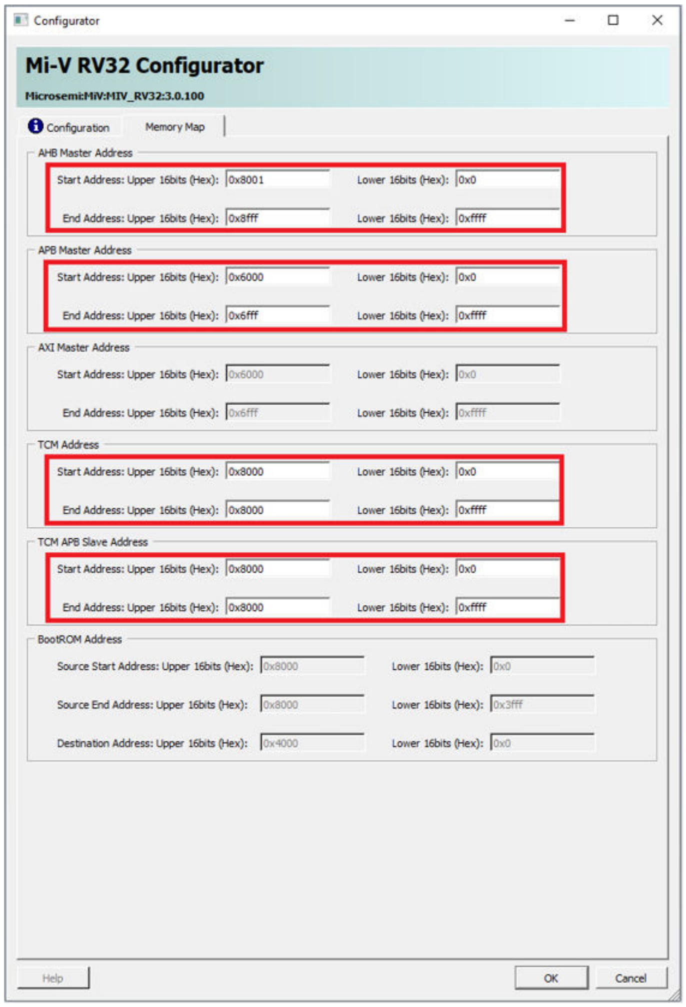
Internal MTIME IRQ: ✓

Other Options

Debug: ☐ Register Forwarding: ☐

ECC: ☐ GPR Registers: ☐

Help OK Cancel



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

AHB Master Address

Start Address: Upper 16bits (Hex):	0x8001	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x8fff	Lower 16bits (Hex):	0xffff

APB Master Address

Start Address: Upper 16bits (Hex):	0x6000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x6fff	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):	0x8000	Lower 16bits (Hex):	0x0
End Address: Upper 16bits (Hex):	0x8000	Lower 16bits (Hex):	0xffff

TCM APB Slave Address

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

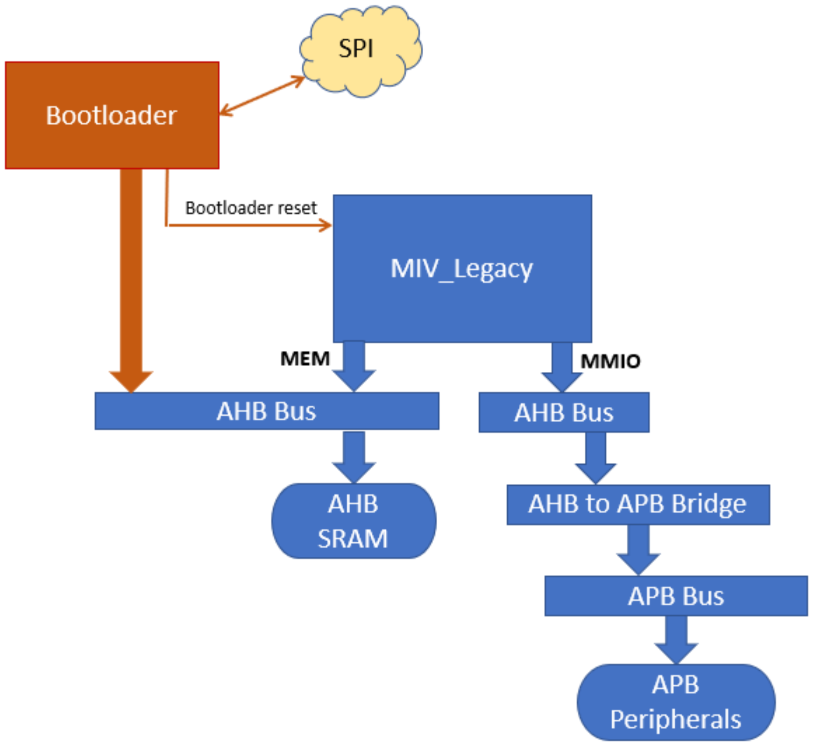
BootROM Address

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

2.4.5 Sample Design 5 - Base Design Bootloader from SPI Flash

2.4.5.1 MIV_Legacy Configuration

Memory is connected to the MEM interface at 0x8000_0000. Peripherals are connected to the MMIO interface at 0x6000_0000. The bootloader is configured to pull data from a SPI flash. The bootloader reset holds the `MIV_Legacy` in reset while the memory is initialized.

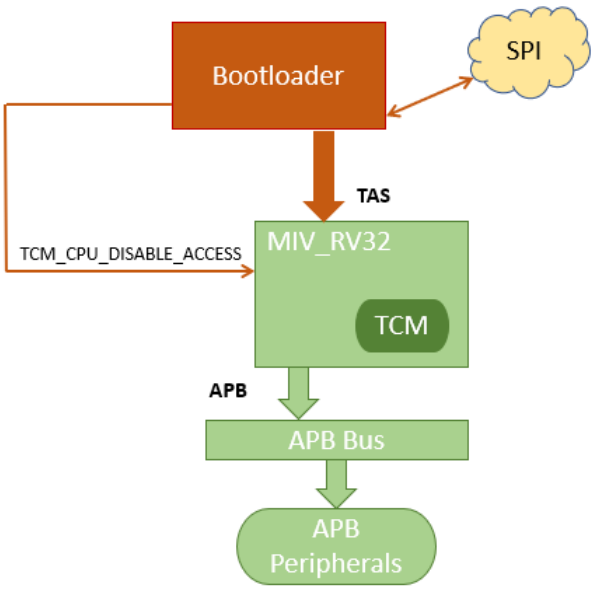


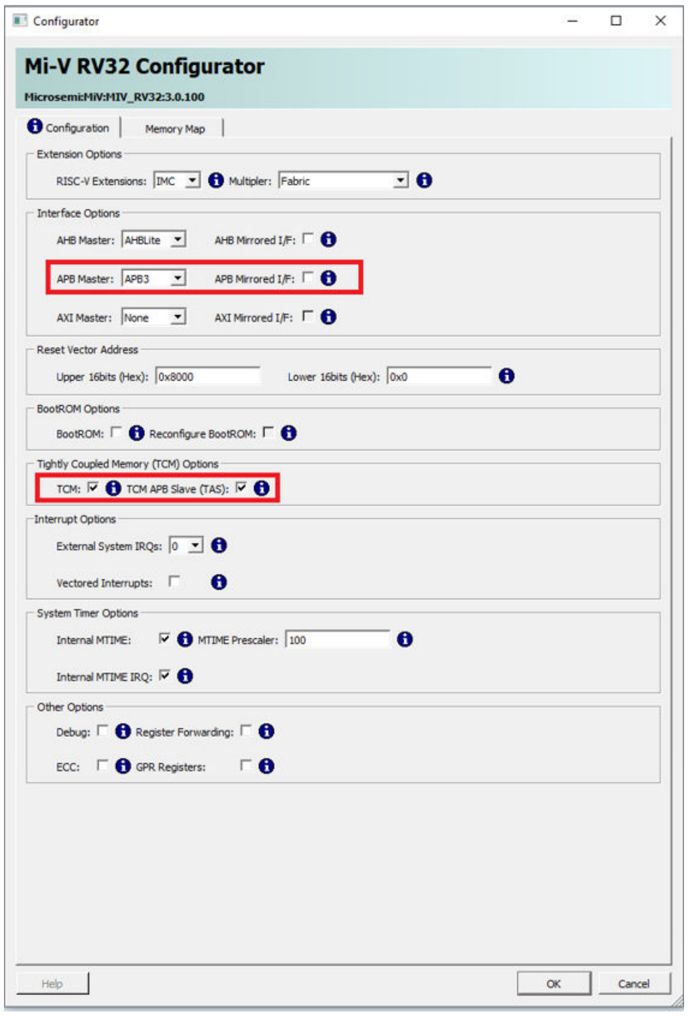
2.4.5.2 MIV_RV32 Configuration

TCM is enabled and set to start at 0x8000_0000 and end at 0x8000_FFFF. The APB master is enabled and configured to start at 0x6000_0000 and end at 0x6FFF_FFFF. The TCM APB Slave (TAS) interface is enabled to allow the bootloader to write data to the TCM. The bootloader holds the `TCM_CPU_DISABLE_ACCESS` input high to prevent the core reading from the TCM, this input becomes available when the TAS is enabled. It means that the core is not held in reset. If the core is held in reset, the interface logic for the TCM and the TAS will also be reset, causing the write operation to the TCM to fail.

Note: It maybe the case that the core requires a reset after the initialization has completed.

The following block diagram and configuration windows show how to enable this setup.





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

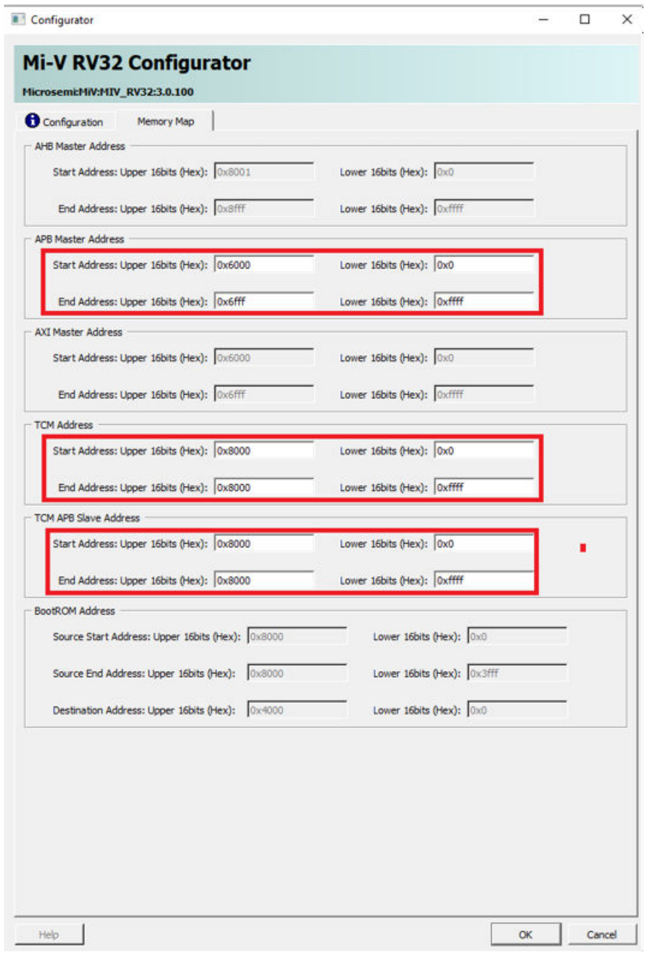
Internal MTIME: ☑   MTIME Prescaler: 100

Internal MTIME IRQ: ☑

Other Options

Debug: ☐   Register Forwarding: ☐

ECC: ☐   GPR Registers: ☐



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

AHB Master Address
Start Address: Upper 16bits (Hex): 0x8001 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x8fff Lower 16bits (Hex): 0xffff

APB Master Address
Start Address: Upper 16bits (Hex): 0x6000 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x6fff 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): 0x8000 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0xffff

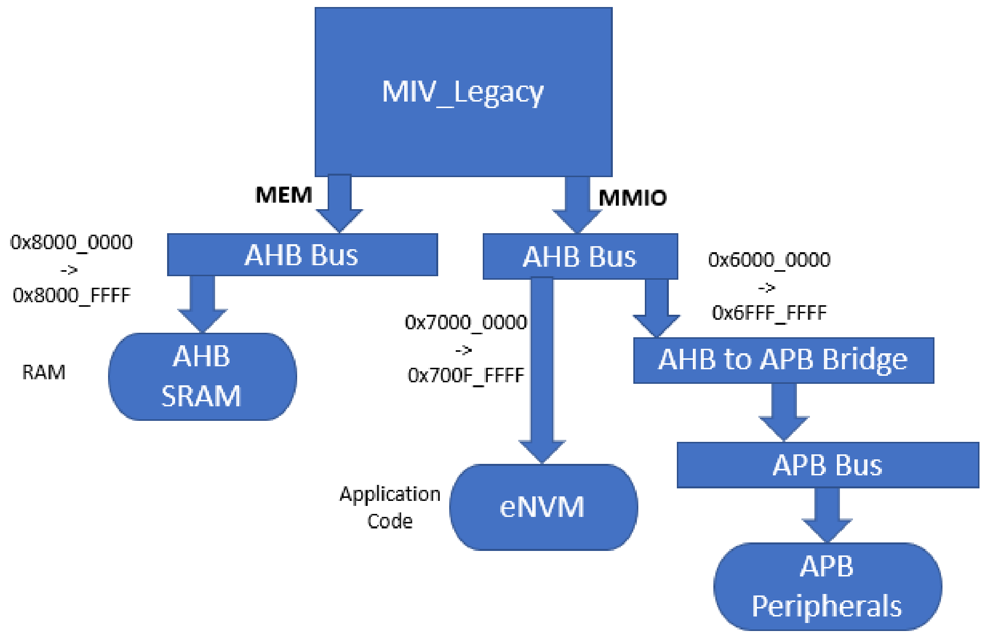
TCM APB Slave Address
Start Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0xffff

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

2.4.6 Sample Design 6 – Base Design Booting from eNVM and SRAM Used as RAM

2.4.6.1 MIV_Legacy Configuration

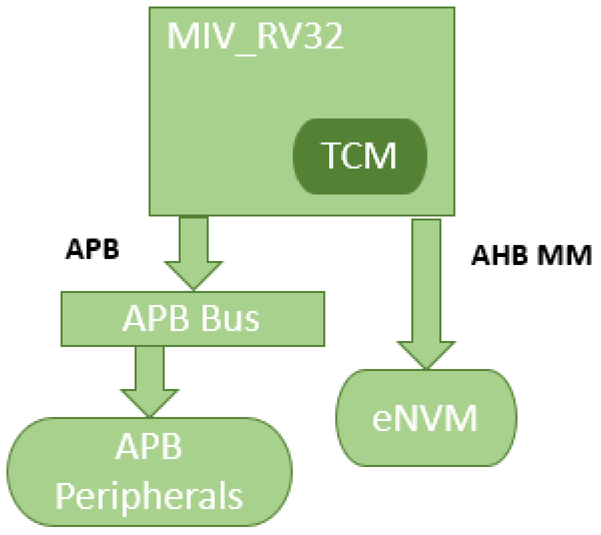
SRAM is connected to the MEM interface at 0x8000_0000 and used as RAM. eNVM is connected to the MMIO interface at 0x7000_0000 and used to store the read only application code. Peripherals are connected to the MMIO interface at 0x6000_0000. The core resets and boots from eNVM and uses the SRAM to hold the application data. As eNVM is read only, there is no code corruption, if an error occurs during execution.

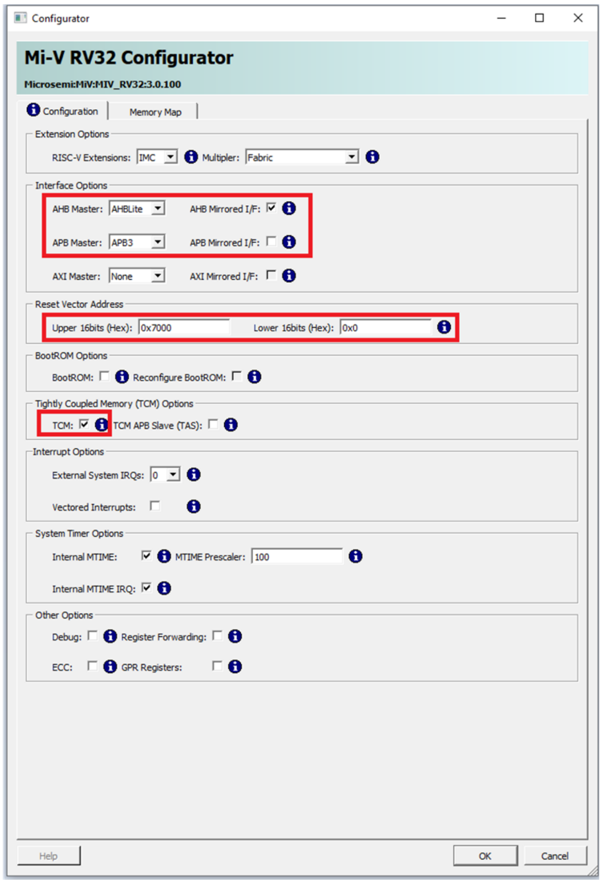


2.4.6.2 MIV_RV32 Configuration

TCM is enabled and set to start at 0x8000_0000 and end at 0x8000_FFFF and is used in place of SRAM shown in the preceding figure. The AHB interface is enabled to start at 0x7000_0000 and end at 0x700F_FFFF to interface the eNVM. The APB master is enabled and configured to start at 0x6000_0000 and end at 0x6FFF_FFFF.

The following block diagram and configuration windows show how to enable this setup.





Extension Options
RISC-V Extensions: IMC Multipler: 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): 0x7000 Lower 16bits (Hex): 0x0

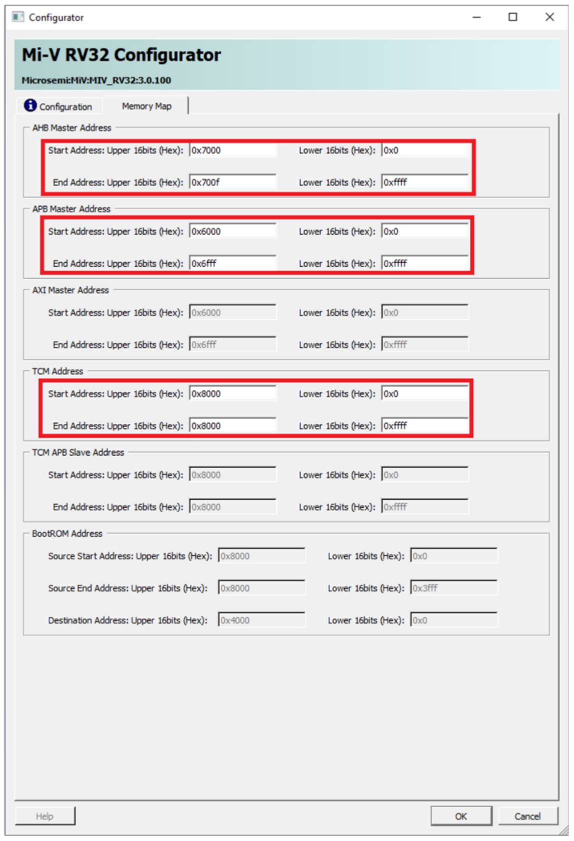
BootROM Options
BootROM: ☐ Reconfigure BootROM: ☐

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: ☐



AHB Master Address
Start Address: Upper 16bits (Hex): 0x7000 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x700f Lower 16bits (Hex): 0xffff

APB Master Address
Start Address: Upper 16bits (Hex): 0x6000 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x6fff 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): 0x8000 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0xffff

TCM APB Slave Address
Start Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0x0
End Address: Upper 16bits (Hex): 0x8000 Lower 16bits (Hex): 0xffff

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

2.5 System Time

The MIV_RV32 core features an internal 64-bit internal counter. The internal counter has the same function as the counter found in the PRCI module of the MIV_Legacy cores. It can be used:

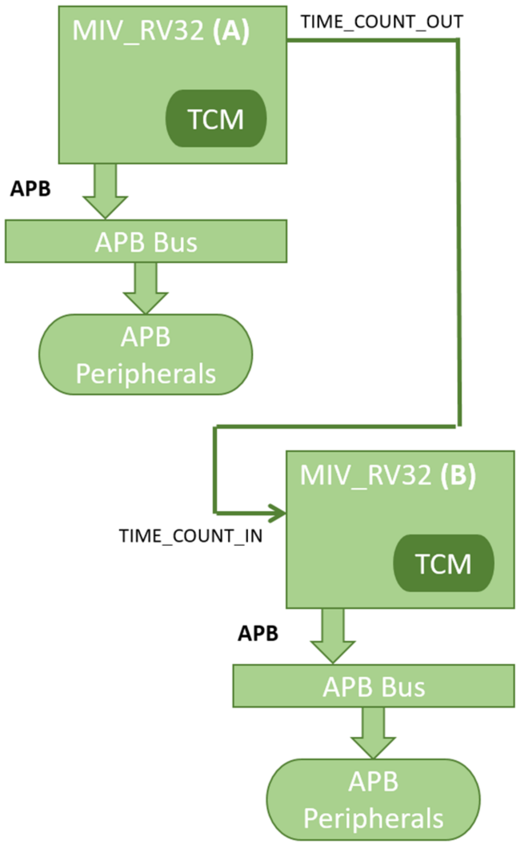
- To generate a time value for the processor.
- To generate a time value for the system.

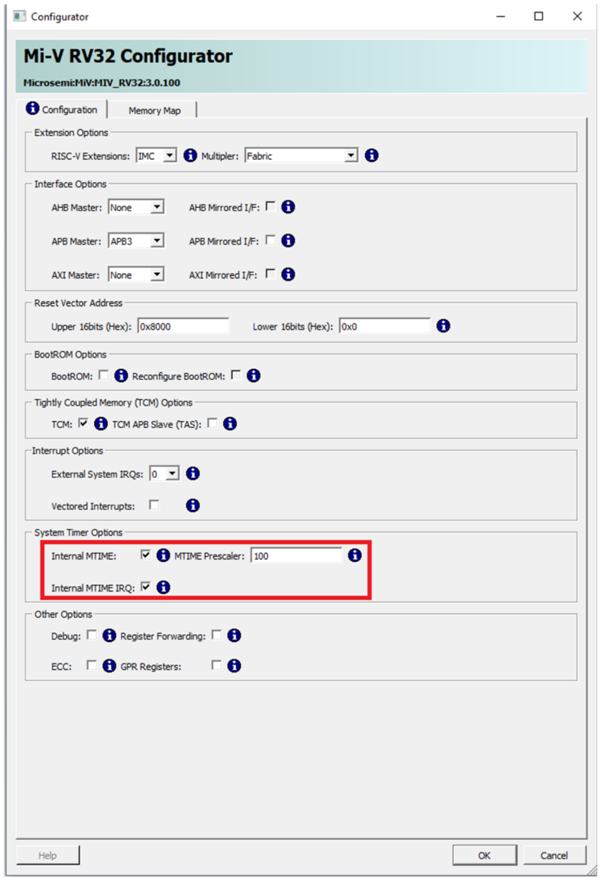
This counter is disabled by default and must be enabled for use. Once enabled, a 64-bit top-level output (TIME_COUNT_OUT) is exposed to provide a time value to the system. In the default mode (counter disabled), a 64 bit top-level input is available (TIME_COUNT_IN) to provide a time value directly to the processor.

The processor also features a 64-bit compare register, which can be used to generate interrupts to the processor's timer interrupt. This can be enabled if needed, and the processors timer interrupt input is connected to the time count compare register. If it is not needed, the disabled top-level TMR_IRQ input is available on the core.

2.5.1 Sample Design 7—Internal MTIME and Internal MTIME IRQ

In this design as shown in the following figure, MIV_RV32 (A) has its internal counter enabled and MIV_RV32 (B) has its counter disabled. The MIV_RV32 (B) receives a time value from the "TIME_COUNT_OUT" of the MIV_RV32 (A). Both the processors have their internal compare registers enabled to generate independent periodic interrupts.




Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

Extension Options
- RISC-V Extensions: IMC
- Multipler: Fabric

Interface Options
- AHB Master: None — 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: ☐

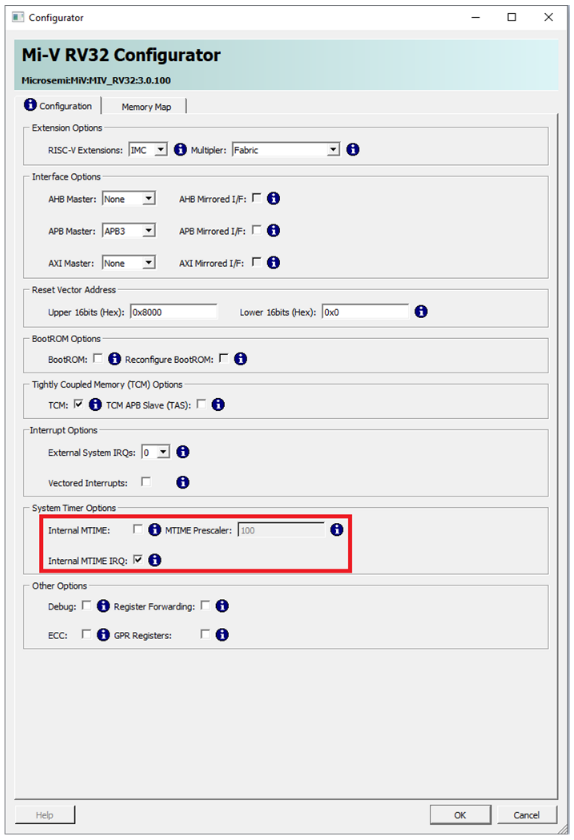
Tightly Coupled Memory (TCM) Options
- TCM: ☑ — TCM APB Slave (TAS): ☐

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: ☐

Help | OK | Cancel


Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

Extension Options
- RISC-V Extensions: IMC
- Multipler: Fabric

Interface Options
- AHB Master: None — 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: ☐

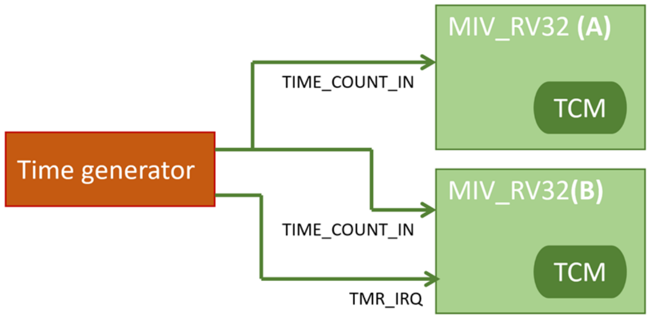
System Timer Options
- Internal MTIME: ☐ — MTIME Prescaler: 100
- Internal MTIME IRQ: ☑

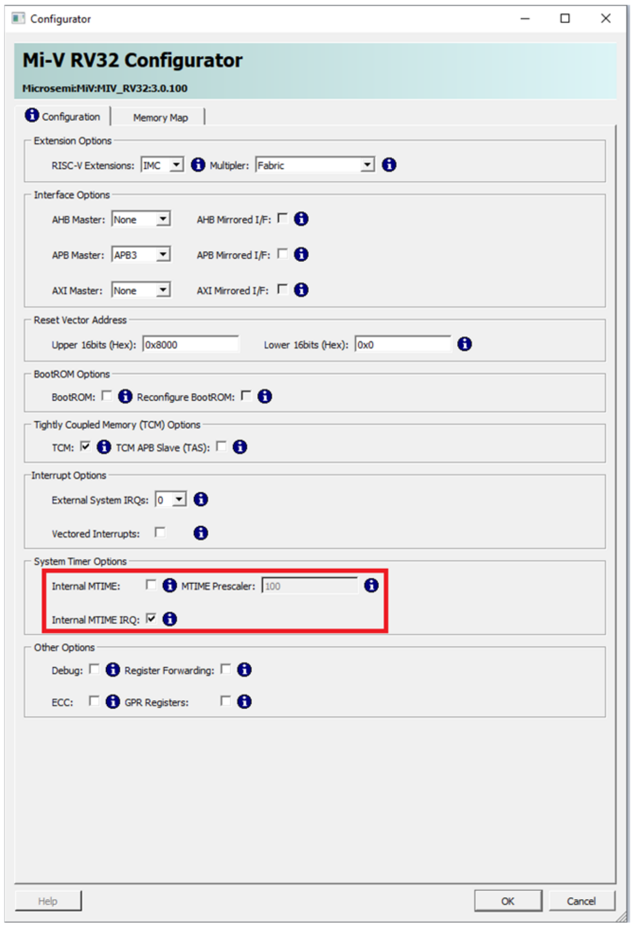
Other Options
- Debug: ☐ — Register Forwarding: ☐
- ECC: ☐ — GPR Registers: ☐

Help | OK | Cancel

2.5.2 Sample Design 8—External MTIME and External MTIME IRQ

In this sample design, `MIV_RV32 (A)` receives time from a system time generator and internally generates an interrupt. `MIV_RV32 (B)` receives time and a timer interrupt from the time generator.





Extension Options
RISC-V Extensions: IMC Multipler: Fabric

Interface Options
AHB Master: None 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

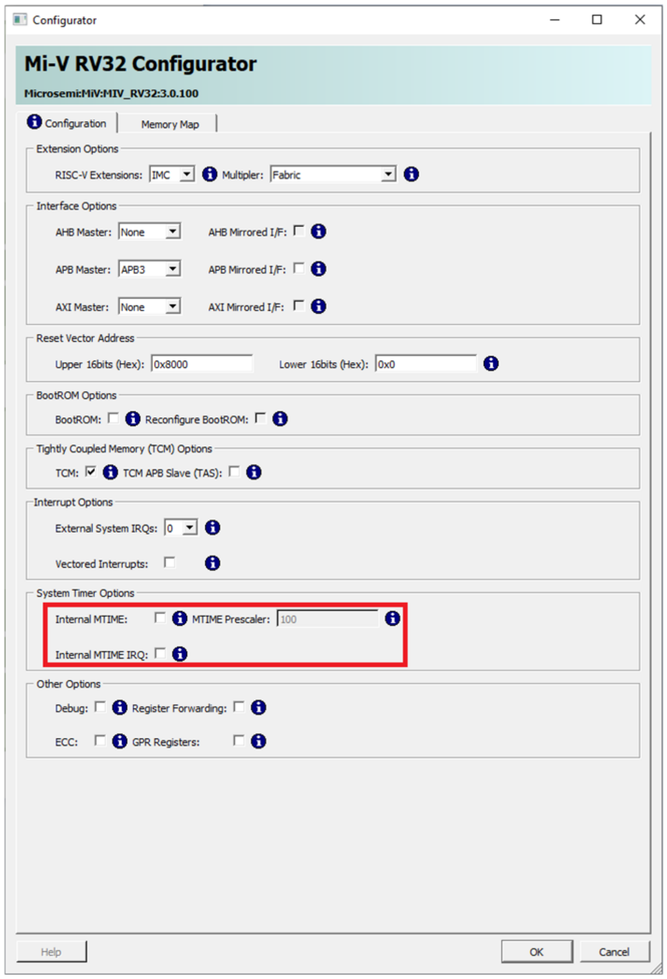
BootROM Options
BootROM: ☐ Reconfigure BootROM: ☐

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: ☐



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Extension Options

RISC-V Extensions: IMC Multiplier: Fabric

Interface Options

AHB Master: None 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: ☐

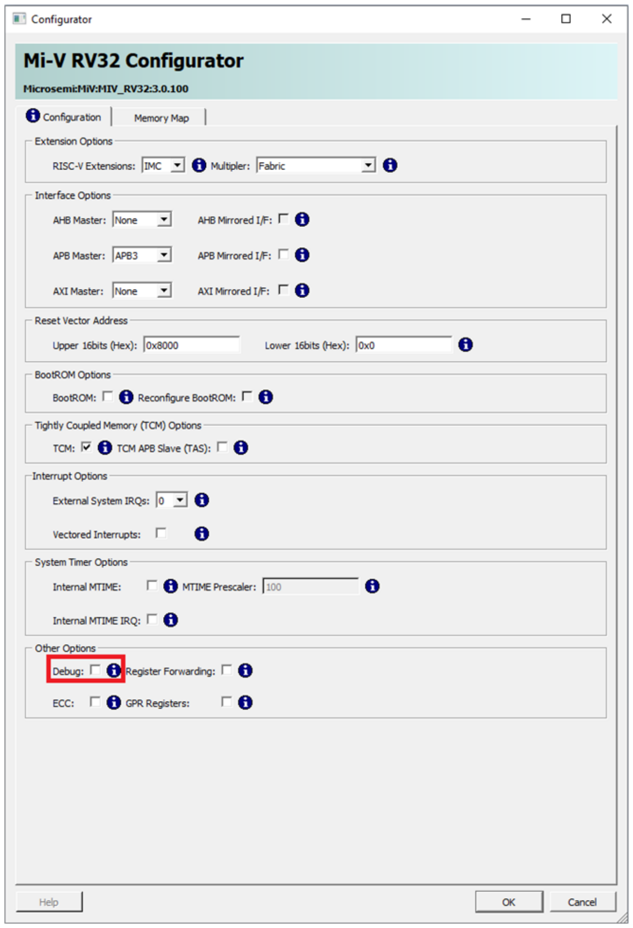
Other Options

Debug: ☐ Register Forwarding: ☐

ECC: ☐ GPR Registers: ☐

2.6 Debug

`MIV_RV32` features a JTAG compliant debug unit. A key difference between this debugger and the `MIV_Legacy` cores debugger is that the debugger is optional in the `MIV_RV32`. If the debug is not needed in a design, the feature can be disabled in the **Configurator** window.



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Extension Options

RISC-V Extensions: IMC Multiplier: Fabric

Interface Options

AHB Master: None 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

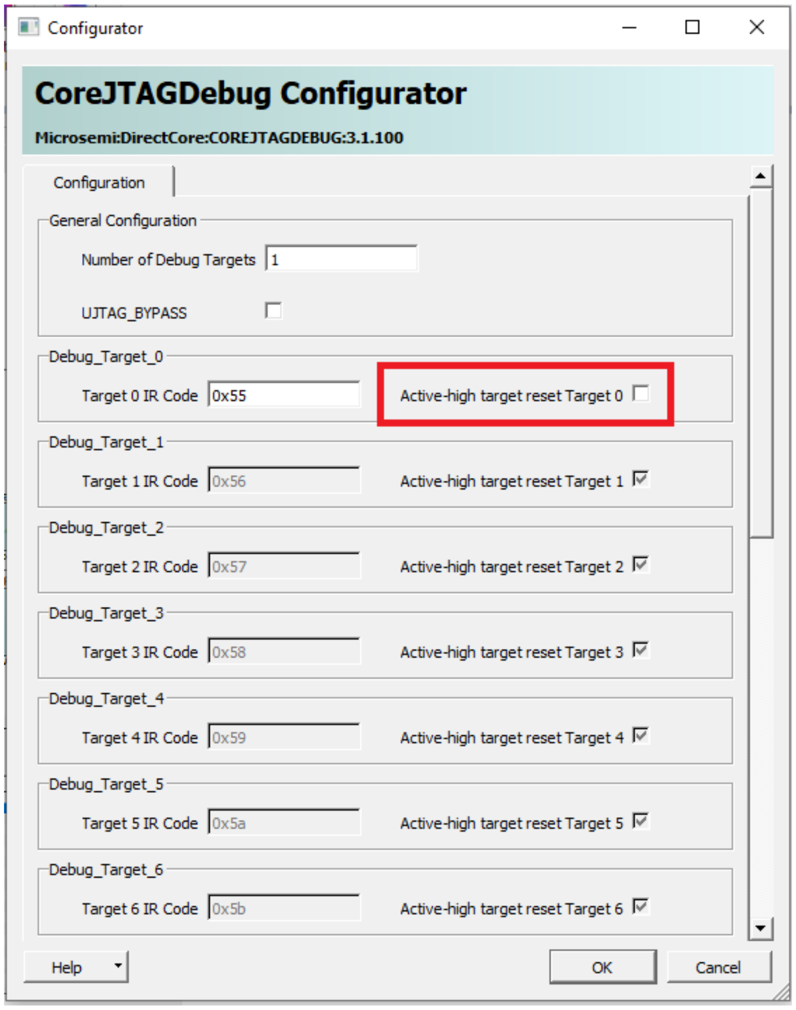
Internal MTIME IRQ: ☐

Other Options

Debug: ☐ Register Forwarding: ☐

ECC: ☐ GPR Registers: ☐

A critical debug difference relates to the `JTAG TRST` polarity. The `MIV_Legacy` cores are active high `JTAG_TRST`, whereas the `MIV_RV32` from v3.0.100 onwards uses an active low `JTAG_TRSTN`. A typical Libero `MIV_RV32` design with debug features uses `CoreJTAGDebug` IP. Therefore, the user needs to ensure the correct polarity is used for the `MIV_RV32`. The following figures illustrate a typical design and the configuration of `JTAG_TRST` polarity on `CoreJTAGDebug`.



Configurator

CoreJTAGDebug Configurator

Microsemi:DirectCore:COREJTAGDEBUG:3.1.100

Configuration

General Configuration

Number of Debug Targets: 1

UJTAG_BYPASS: ☐

Debug_Target_0

Target 0 IR Code: 0x55 — Active-high target reset Target 0: ☐

Debug_Target_1

Target 1 IR Code: 0x56 — Active-high target reset Target 1: ☑

Debug_Target_2

Target 2 IR Code: 0x57 — Active-high target reset Target 2: ☑

Debug_Target_3

Target 3 IR Code: 0x58 — Active-high target reset Target 3: ☑

Debug_Target_4

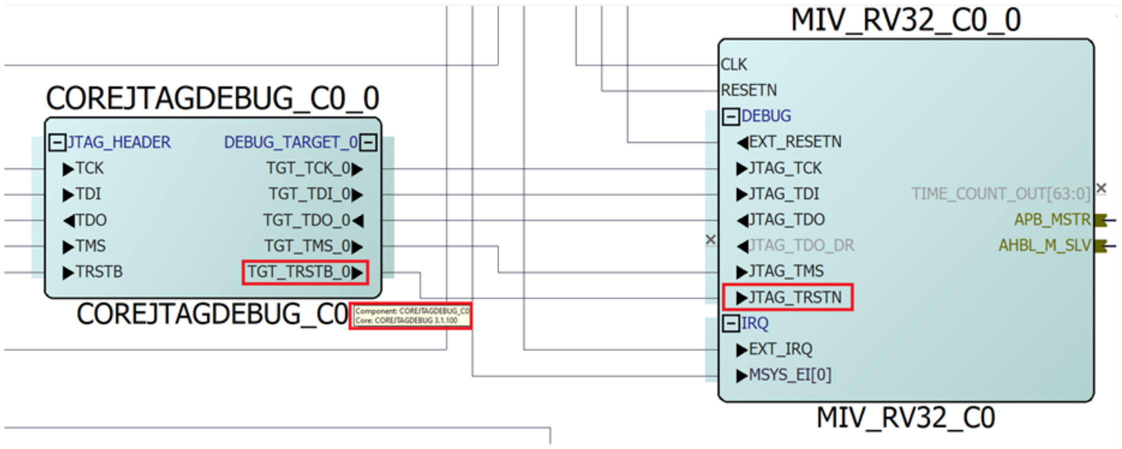
Target 4 IR Code: 0x59 — Active-high target reset Target 4: ☑

Debug_Target_5

Target 5 IR Code: 0x5a — Active-high target reset Target 5: ☑

Debug_Target_6

Target 6 IR Code: 0x5b — Active-high target reset Target 6: ☑

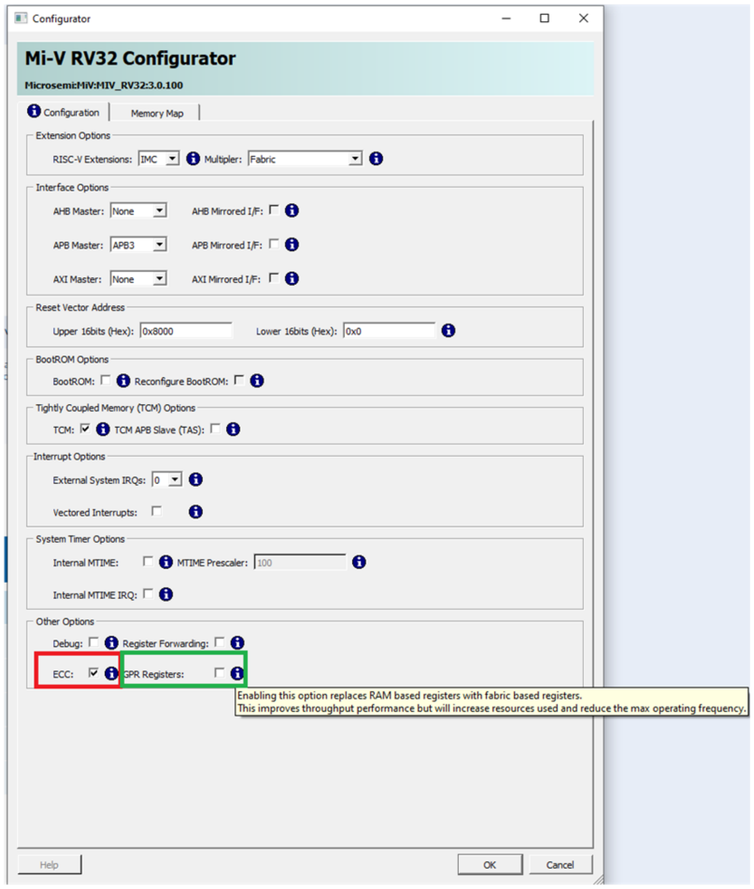


2.7 ECC

Some of the `MIV_Legacy` cores have support for ECC on their caches. As the `MIV_RV32` does not have a cache, it does not need this protection, but there are SRAM implementations within the core that can be protected from errors.

1. In its standard configuration, the `MIV_RV32` core uses RAM-based General Purpose Resources (GPRs). These are susceptible to errors.

   1.1. By enabling the **GPR Registers** option for the core, generates GPRs as registers, which are not susceptible to the same errors.
   1.2. By enabling the **ECC** option for the core, generates a fabric EDAC wrapper around the RAM-based GPRs and any single bit errors are corrected and cause an interrupt to be generated to the hart. Double bit errors cause a soft reset.
2. If the TCM is enabled, it may also need error protection.
   2.1. By enabling the **ECC** option for the core, generates a fabric EDAC wrapper around the RAM-based GPRs and any errors cause interrupts to be generated to the hart.



Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Extension Options
RISC-V Extensions: IMC
Multipler: Fabric

Interface Options
AHB Master: None
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: ☐

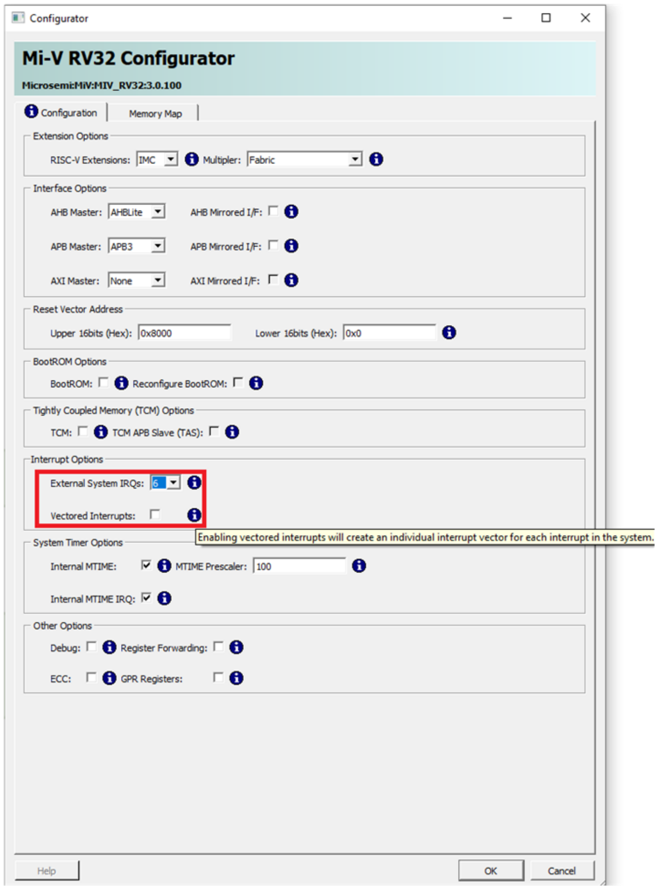
System Timer Options
Internal MTIME: ☐
MTIME Prescaler: 100
Internal MTIME IRQ: ☐

Other Options
Debug: ☐
Register Forwarding: ☐
ECC: ✓
GPR Registers: ☐

Enabling this option replaces RAM based registers with fabric based registers.
This improves throughput performance but will increase resources used and reduce the max operating frequency.

2.8 Interrupts

`MIV_RV32` does not feature a PLIC like the `MIV_Legacy` cores. It has support for the three standard interrupts defined in the RISC-V Spec (Soft, Timer, and External) and also has the option to generate up to six additional external interrupts. An option to use Vectored Interrupts is also provided on the `MIV_RV32` configuration GUI as shown in the following figure.



Configurator

Mi-V RV32 Configurator
Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

Extension Options
RISC-V Extensions: IMC Multipler: 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: 6
Vectored Interrupts: ☐
Enabling vectored interrupts will create an individual interrupt vector for each interrupt in the system.

System Timer Options
Internal MTIME: ✓ MTIME Prescaler: 100
Internal MTIME IRQ: ✓

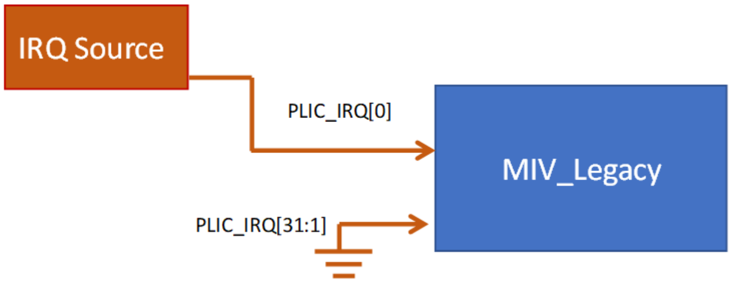
Other Options
Debug: ☐ Register Forwarding: ☐
ECC: ☐ GPR Registers: ☐

Help | OK | Cancel

2.8.1 Sample Design 9 – Single Interrupt Source

2.8.1.1 MIV_Legacy Configuration

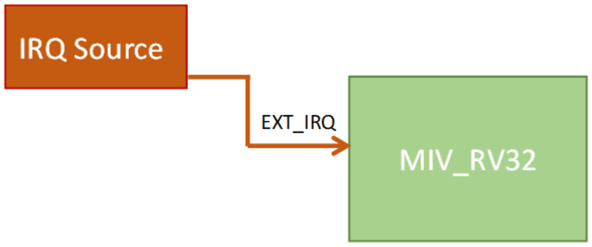
In this sample design, `MIV_Legacy` has one interrupt source with the remaining 31 PLIC interrupts tied low.



```
uint8_t External_1_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
```

2.8.1.2 MIV_RV32 Configuration

The interrupt source is connected to `EXT_IRQ` input of `MIV_RV32`.



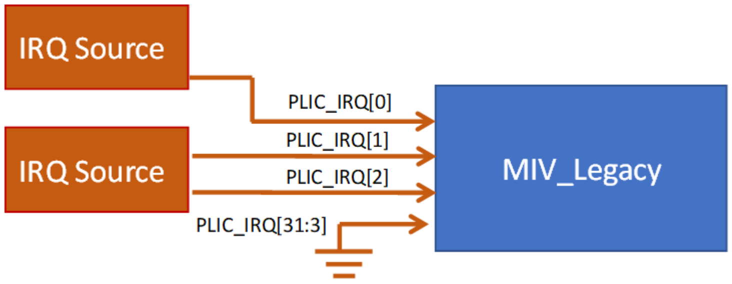
There are no configuration options that need to be selected to use `EXT_IRQ`. If required, you can enable the Vectored mode.

```
uint8_t External_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
int main(int argc, char **argv)
{
    HAL_enable_interrupts();

    asm volatile("wfi");
}
```

2.8.2 Sample Design 10 – Multiple Interrupt Sources

2.8.2.1 MIV_Legacy Configuration

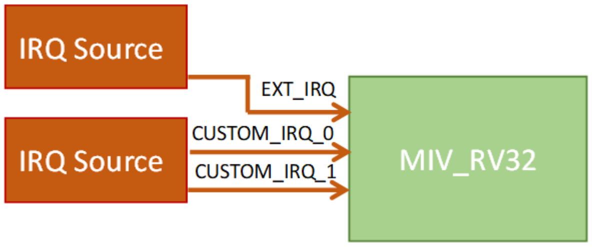
In this sample design, `MIV_Legacy` has an interrupt source generating an interrupt for PLIC_IRQ[0], and a second source generating interrupts for PLIC_IRQ[1] and PLIC_IRQ[2] with the remaining PLIC interrupts tied low.



```
uint8_t External_1_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}

uint8_t External_2_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}

uint8_t External_3_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
```

2.8.2.2 MIV_RV32 Configuration

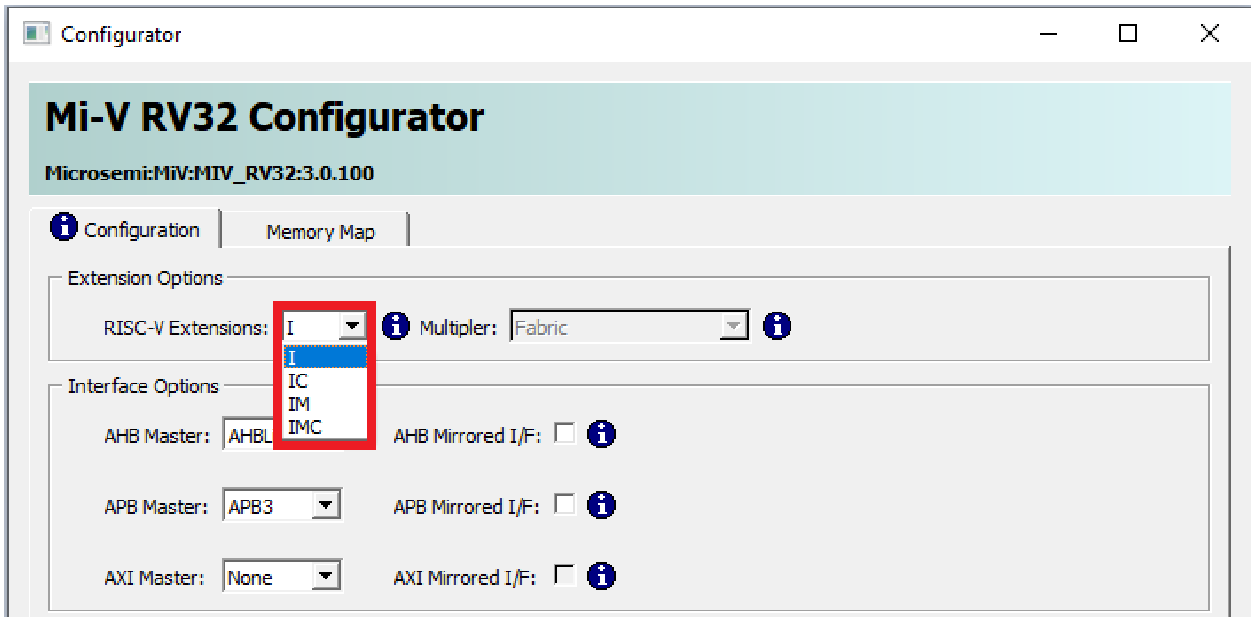
The interrupt source generating a single interrupt is connected to the `EXT_IRQ`, and the source generating the two second interrupts is connected to two of the custom external interrupts.



```
void External_IRQHandler()
{

}

void MSYS_E10_IRQHandler(void)
{

}
void MSYS_E11_IRQHandler(void)
{

}
```

2.9 RISC-V Extensions

The MIV_RV32 core can use the base RISC-V Integer extension along with the Multiply and/or Compressed extensions as shown in the following figure. The multiply extension can be used with several versions of multipliers, depending on the processor frequency required and processor performance needed; multiplication can be completed in 1 cycle, 2 cycles or 32 cycles. The MIV_Legacy cores featured the Integer, Multiplication and Atomic extensions. The I and M extensions can be enabled in the MIV_RV32 core and the Atomic extension is used for mutli-core systems, if atomics are required, an MIV_Legacy core must be used.



RISC-V Extensions: I
Multipler: Fabric
AHB Master: AHBL
AHB Mirrored I/F: ☐
APB Master: APB3
APB Mirrored I/F: ☐
AXI Master: None
AXI Mirrored I/F: ☐

2.9.1 RISC-V I Extension

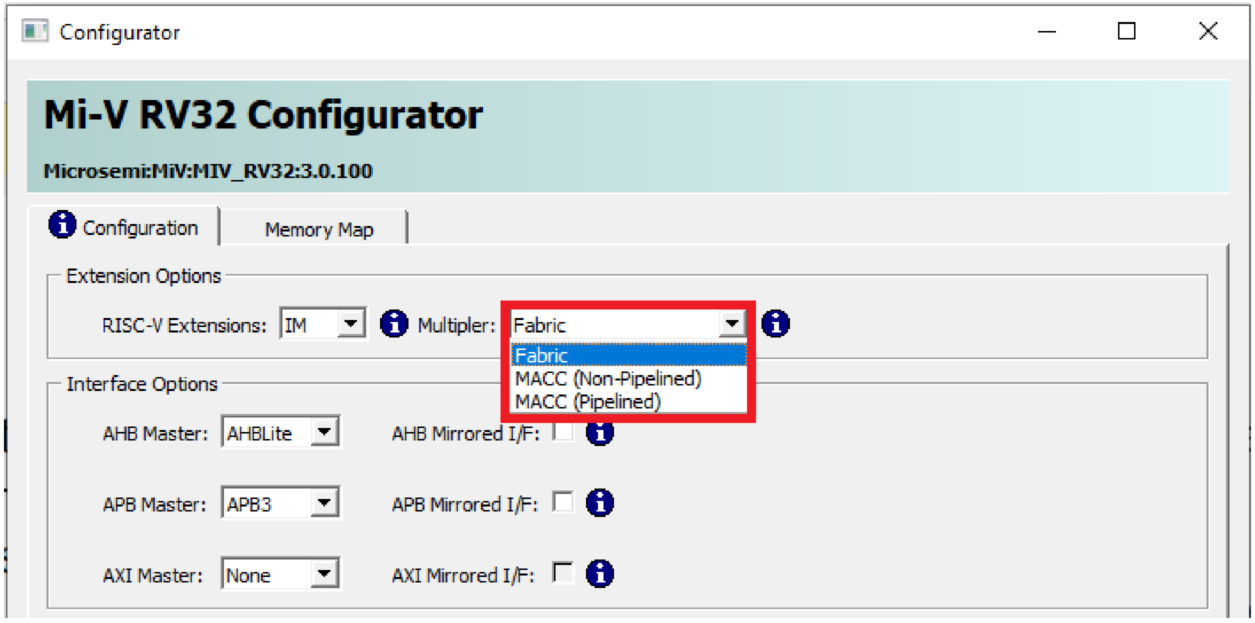
This is the base RISC-V extension and is required in all cores.

2.9.2 RISC-V M Extension

The M extension adds multiply and divide instructions to the core. These can be used in place of software equivalents to improve code performance while increasing the area of MIV_RV32.

A benefit will only be seen from the RISC-V Multiply extension, if multiply operations are used frequently by software.

The multiplier in `MIV_RV32` can be one of several types: MACC, MACC Pipelined, and Fabric, as seen in the following figure.



Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

Extension Options

RISC-V Extensions: IM

Multipler: Fabric

- Fabric
- MACC (Non-Pipelined)
- MACC (Pipelined)

Interface Options

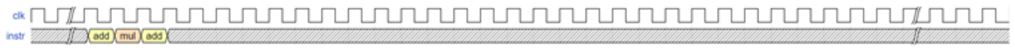
AHB Master: AHBLite — AHB Mirrored I/F: ☐

APB Master: APB3 — APB Mirrored I/F: ☐

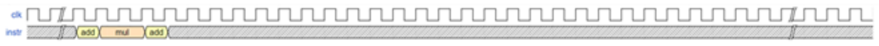
AXI Master: None — AXI Mirrored I/F: ☐

The MACC options use the math blocks included in the FPGA fabric to carry out the multiplication operations, while the fabric option instantiates a fabric multiplier.

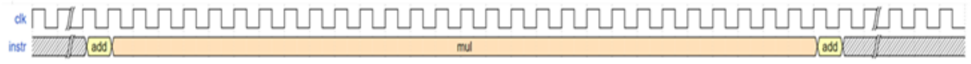
Using the non-pipelined multiplier option, operations complete in one cycle.



Using the pipelined multiplier option, operations complete in two cycles.



Using the fabric multiplier option, operations complete in 32 cycles.



Using the 32-cycle multiplier can still be very beneficial, depending on the values being multiplied. Software multiplication (that is, only using the RV32I extension) can take many multiples of 32-cycle to complete and will not take the same number of cycles for different values. The fabric multiplier is still faster than this and completes multiplication in 32-cycle regardless of values.

For application that rely heavily on multiplication operations, a MACC option is recommended. For those applications that require less or none at all, a fabric multiplier can be used or the M extension can be excluded respectively.

2.9.2.1 Using Software Multiplication

Using the M extension with the following C code:

```
uint32_t val0 = 5;
uint32_t val1 = 7;
val0 = val0 * val1;
```

Compiles to the following RISC-V assembly:

```
lw    a4,-28(s0)
lw    a5,-24(s0)
```

```
mul    a5,a4,a5
sw    a5,-28(s0)
```

With the highlighted “mul” instruction, taking a fixed number of cycles to complete depending on the multiplier type chosen.

Using software multiplication, the same C code complies to the following RISC-V assembly:

```
lw    a5,-28(s0)
lw    a4,-24(s0)
mv    a1,a4
mv    a0,a5
jal    ra,80001330 <__mulsi3>
mv    a5,a0
sw    a5,-28(s0)

__mulsi3():
mv    a2,a0
li    a0,0
andi    a3,a1,1
beqz    a3,80001344 <__mulsi3+0x14>
add    a0,a0,a2
srli    a1,a1,0x1
slli    a2,a2,0x1
bnez    a1,80001338 <__mulsi3+0x8>
ret
```

The C code* for the loop being executed by the `__mulsi3():` function is as follows:

```
unsigned int
__mulsi3 (unsigned int a, unsigned int b)
{
  unsigned int r = 0;
  while (a)
    {
      if (a & 1)
    r += b;
      a >>= 1;
      b <<= 1;
    }
  return r;
}
```

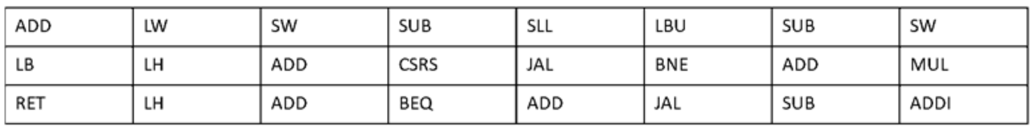
This loop executes until the multiplication operation has completed as opposed to the “mul” instruction available with the M extension.

Note: This function is included in the standard C library, included by GCC automatically when building your code, if the M extension is not selected.

2.9.3 RISC-V C Extension

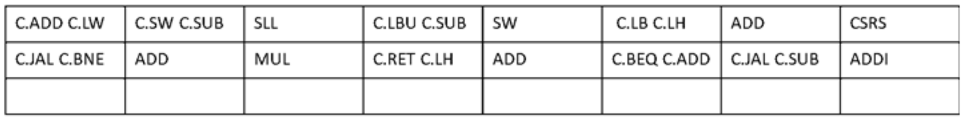
Twenty-five of the base RV32I instructions have a compressed variant, which can be used in place of the base instruction. The compressed variant is only 16 bits instead of 32. This allows for a 20%–30% reduction in overall code size for a given application.

The following figure is an example chunk of RISC-V instructions, each cell is a 32-bit memory location.



ADD	LW	SW	SUB	SLL	LBU	SUB	SW
LB	LH	ADD	CSRS	JAL	BNE	ADD	MUL
RET	LH	ADD	BEQ	ADD	JAL	SUB	ADDI

The following figure is the same chunk of instructions, but this time the C extension is included and the 16-bit instructions are mixed with the 32-bit instructions.



C.ADD C.LW	C.SW C.SUB	SLL	C.LBU C.SUB	SW	C.LB C.LH	ADD	CSRS
C.JAL C.BNE	ADD	MUL	C.RET C.LH	ADD	C.BEQ C.ADD	C.JAL C.SUB	ADDI

Using the C extension, it allows for a reduction in code size with a small increase in core area. The reduced code size allows for a smaller TCM and reduced RAM usage, which outweighs the increase in area from adding the extension. The C extension is recommended in most circumstances to reduce the code size.

3. Migrating Software Projects

MIV_RV32 HAL v3.0.100 or greater is required to use `MIV_RV32`.

3.1 Prerequisites

- Download and install latest SoftConsole at https://www.microsemi.com/product-directory/design-tools/4879-softconsole#downloads.
- Download and install latest Firmware Catalog at https://www.microsemi.com/product-directory/design-tools/4880-firmware-catalog#downloads.

Note: If you have Libero® SoC Software installed, you need not install the Firmware Catalog as it is included in the Libero SoC Software.

3.2 Recommended Migration Process

The recommended way to migrate is to use the default Mi-V RV32IMA application from the SoftConsole workspace.

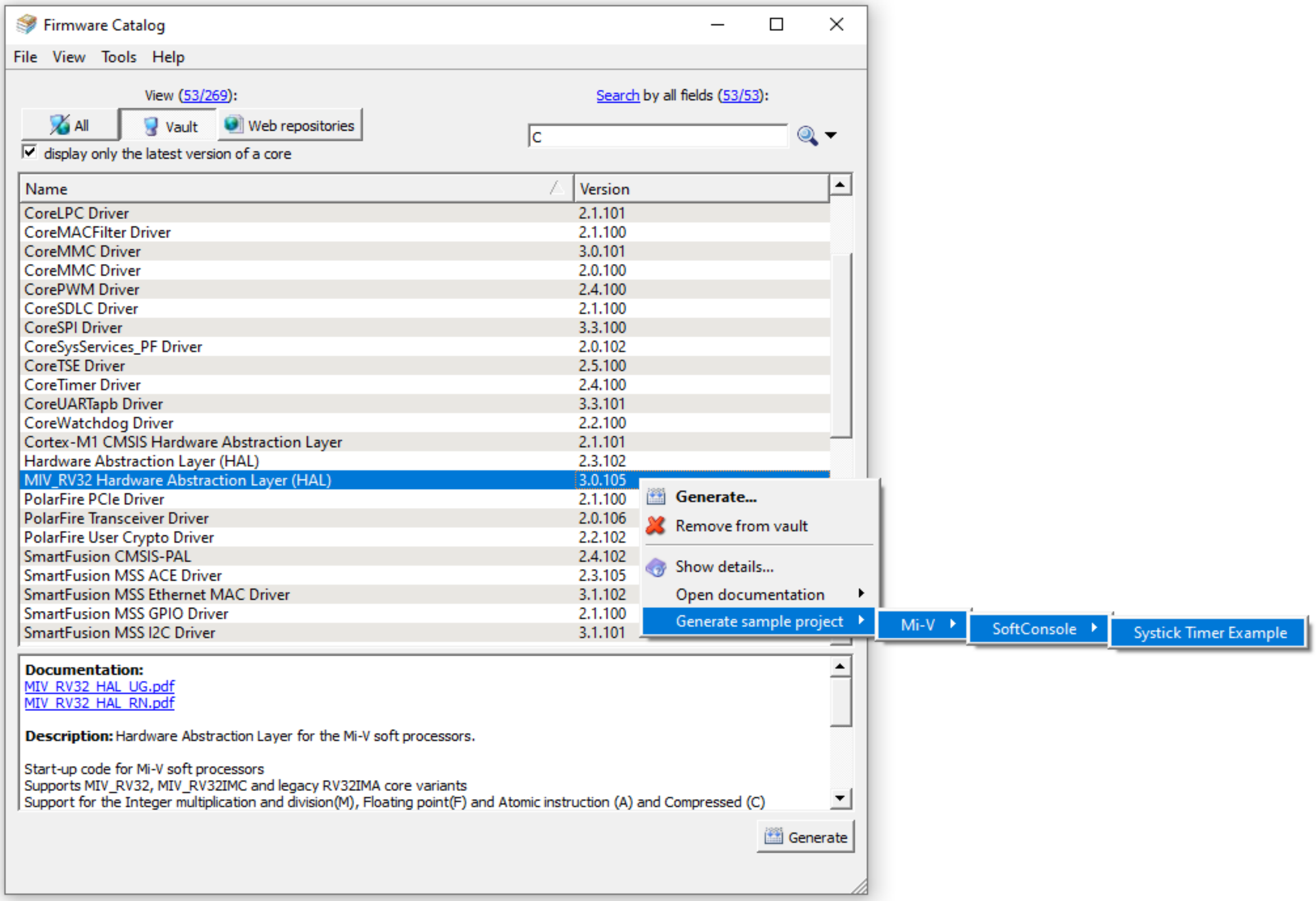
The migration process involves the following steps:

1. Generate the SoftConsole example projects from MIV_RV32 HAL v3.0, or greater, package in the firmware catalog.
2. Import the `miv-rv32i-systick-blinky` example project into workspace.
3. Copy your application specific files (`main.c` and other application specific files including driver) into the `miv-rv32i-systick-blinky` example project.
4. Replicate your application project properties like pre-processor, include paths, optimization levels, and so on in the `miv-rv32i-systick-blinky` example project.
   - The `readme.txt` document located in the root directory of `miv-rv32i-systick-blinky` example project describes the linker script and macro combinations required for conditional compilation. If you have any application specific modifications in the linker script, then those should also be ported to the new linker script you are going to use for `miv-rv32i-systick-blinky` project.
   - The default debug and release build configurations are provided with the `miv-rv32i-systick-blinky` example project.
5. Build the Debug or Release target. Fix any build errors, if they occur.
6. Debug the application using debug or release launch configuration.

3.3 Example of Recommended Migration Process

The following steps describe migration to an `MIV_RV32` core SoftConsole application.

1. In the Firmware catalog, search for the latest MIV_RV32 HAL v3.0.x, or greater. Right-click **MIV_RV32 Hardware Abstraction Layer (HAL)** to generate a sample project, as shown in following figure.


Firmware Catalog

File View Tools Help

View (53/269): All | Vault | Web repositories

☑ display only the latest version of a core

Search by all fields (53/53): C

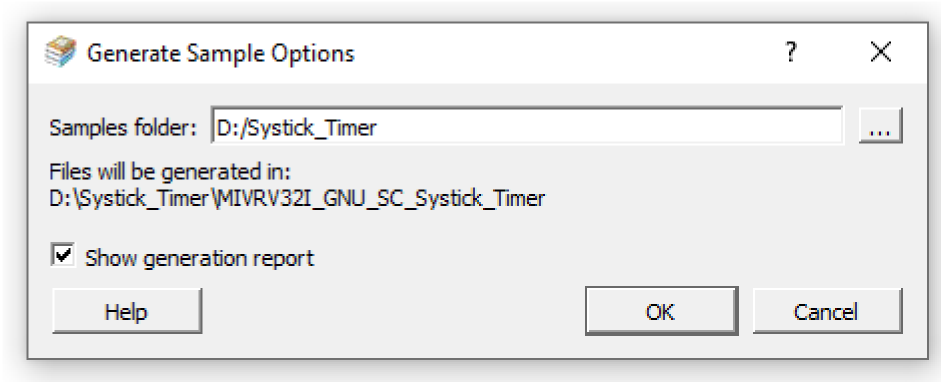
Name	Version
CoreLPC Driver	2.1.101
CoreMACFilter Driver	2.1.100
CoreMMC Driver	3.0.101
CoreMMC Driver	2.0.100
CorePWM Driver	2.4.100
CoreSDLC Driver	2.1.100
CoreSPI Driver	3.3.100
CoreSysServices_PF Driver	2.0.102
CoreTSE Driver	2.5.100
CoreTimer Driver	2.4.100
CoreUARTapb Driver	3.3.101
CoreWatchdog Driver	2.2.100
Cortex-M1 CMSIS Hardware Abstraction Layer	2.1.101
Hardware Abstraction Layer (HAL)	2.3.102
MIV_RV32 Hardware Abstraction Layer (HAL)	3.0.105
PolarFire PCIe Driver	2.1.100
PolarFire Transceiver Driver	2.0.106
PolarFire User Crypto Driver	2.2.102
SmartFusion CMSIS-PAL	2.4.102
SmartFusion MSS ACE Driver	2.3.105
SmartFusion MSS Ethernet MAC Driver	3.1.102
SmartFusion MSS GPIO Driver	2.1.100
SmartFusion MSS I2C Driver	3.1.101

Documentation:
MIV_RV32_HAL_UG.pdf
MIV_RV32_HAL_RN.pdf

Description: Hardware Abstraction Layer for the Mi-V soft processors.

Start-up code for Mi-V soft processors
Supports MIV_RV32, MIV_RV32IMC and legacy RV32IMA core variants
Support for the Integer multiplication and division(M), Floating point(F) and Atomic instruction (A) and Compressed (C)

2. In the **Generate Sample Options** dialog box, enter a folder location in which the project must be generated, as shown in the following figure.

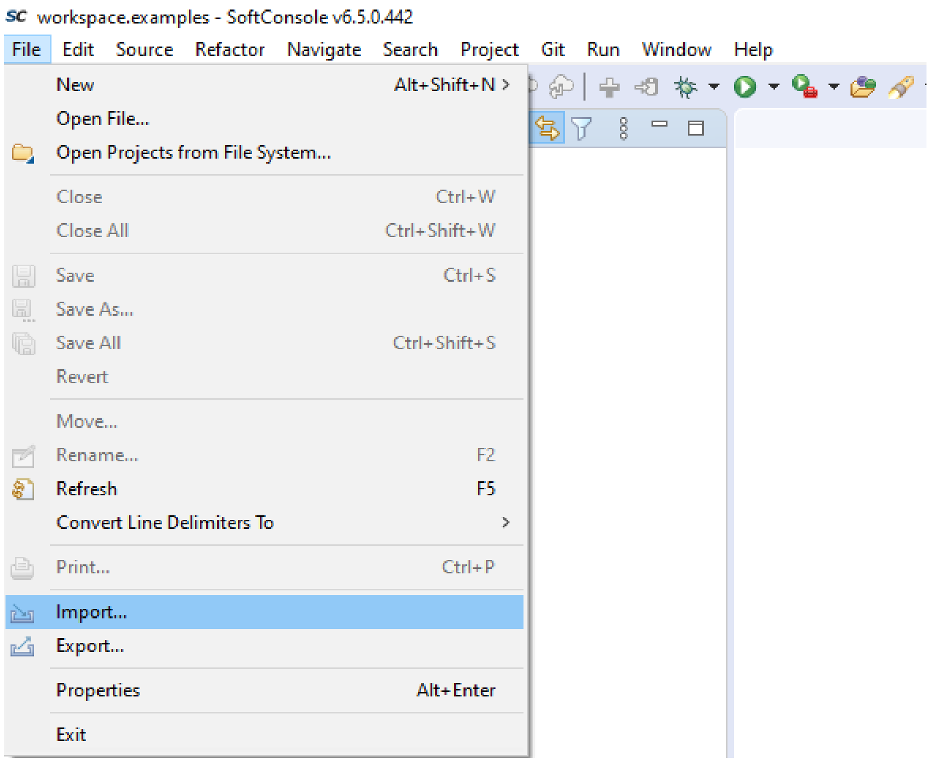

Generate Sample Options

Samples folder: D:/Systick_Timer

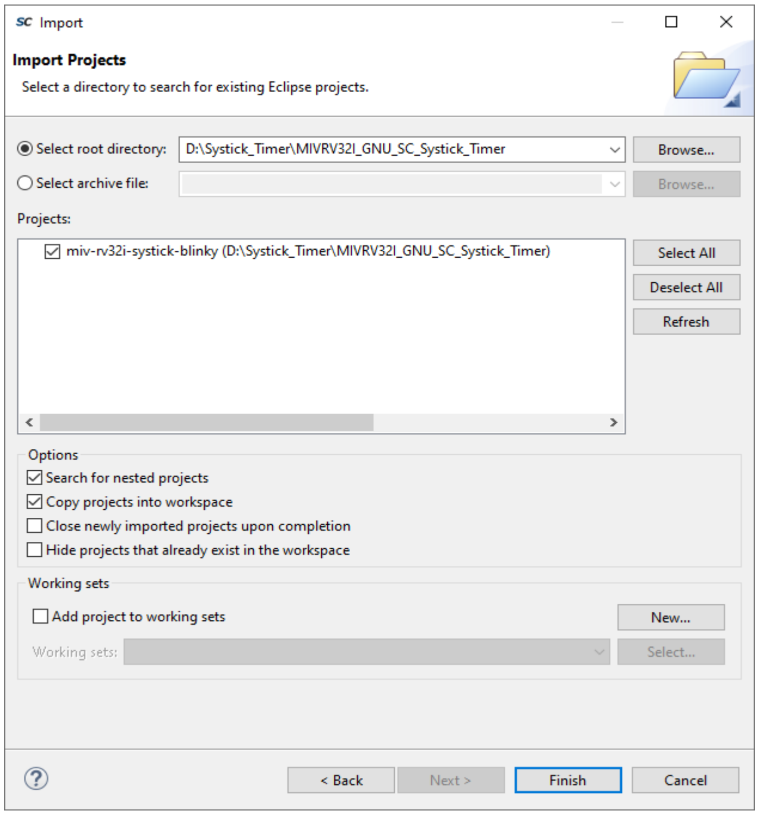
Files will be generated in:
D:\Systick_Timer\MIVRV32I_GNU_SC_Systick_Timer

☑ Show generation report

3. Open SoftConsole workspace and import the generated project using the option, as shown in the following figure.



3.1. Select **General** > **Existing Projects** into workspace and click **Next**.
3.2. Copy the root directory (the generated project path) or use **Browse** to navigate to the root directory.
3.3. Select the application in the directory to import and click **Finish**.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

(●) Select root directory: D:\Systick_Timer\MIVRV32I_GNU_SC_Systick_Timer [Browse...]

( ) Select archive file: [Browse...]

Projects:

[✓] miv-rv32i-systick-blinky (D:\Systick_Timer\MIVRV32I_GNU_SC_Systick_Timer)

Options

[✓] Search for nested projects

[✓] Copy projects into workspace

[ ] Close newly imported projects upon completion

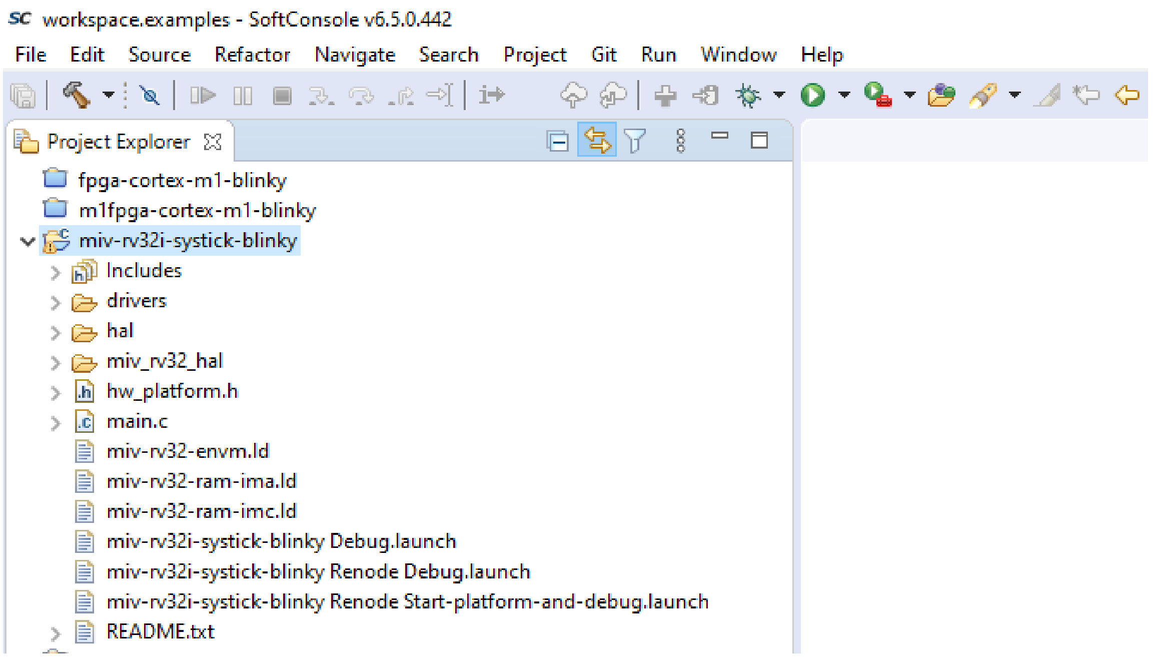
[ ] Hide projects that already exist in the workspace

Working sets

[ ] Add project to working sets

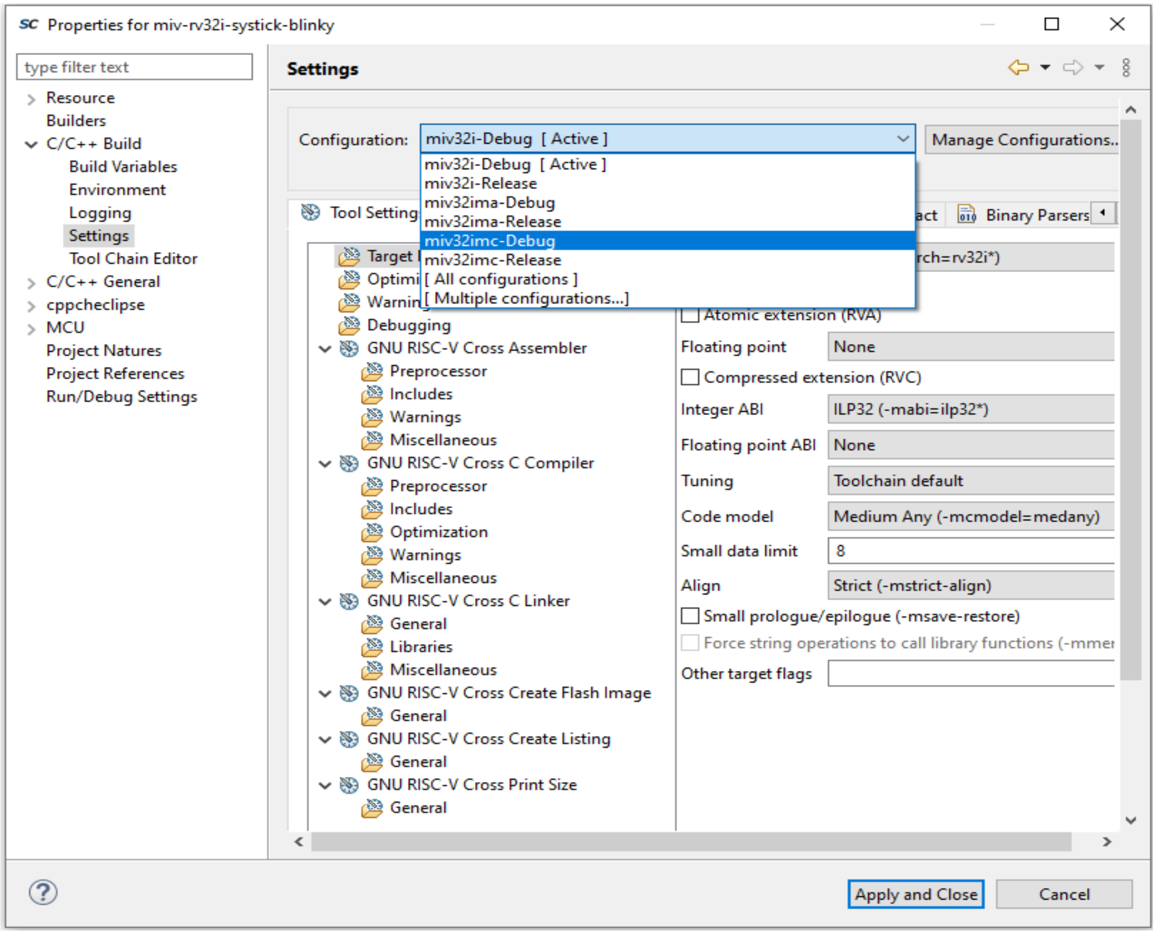
Working sets:

The following figure shows the imported SoftConsole project in the workspace.


SC workspace.examples - SoftConsole v6.5.0.442
File Edit Source Refactor Navigate Search Project Git Run Window Help
Project Explorer

```
fpga-cortex-m1-blinky
m1fpga-cortex-m1-blinky
miv-rv32i-systick-blinky
  Includes
  drivers
  hal
  miv_rv32_hal
  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.launch
  README.txt
```

4. As described in the Recommended migration process section, replace your application specific files in the example.
5. Open the `hw_platform.h` file and configure,
   5.1. The peripheral base addresses as per the memory map generated by Libero SoC Software design.
   5.2. The system clock frequency based on the Libero SoC Software design.
6. Right-click the project name and open the properties menu (last option in menu). The project settings offer six types of configurations like debug and release configurations for Mi-V I, IMA, and IMC cores.
   Note: The selected configuration must match with the processor core in the design.

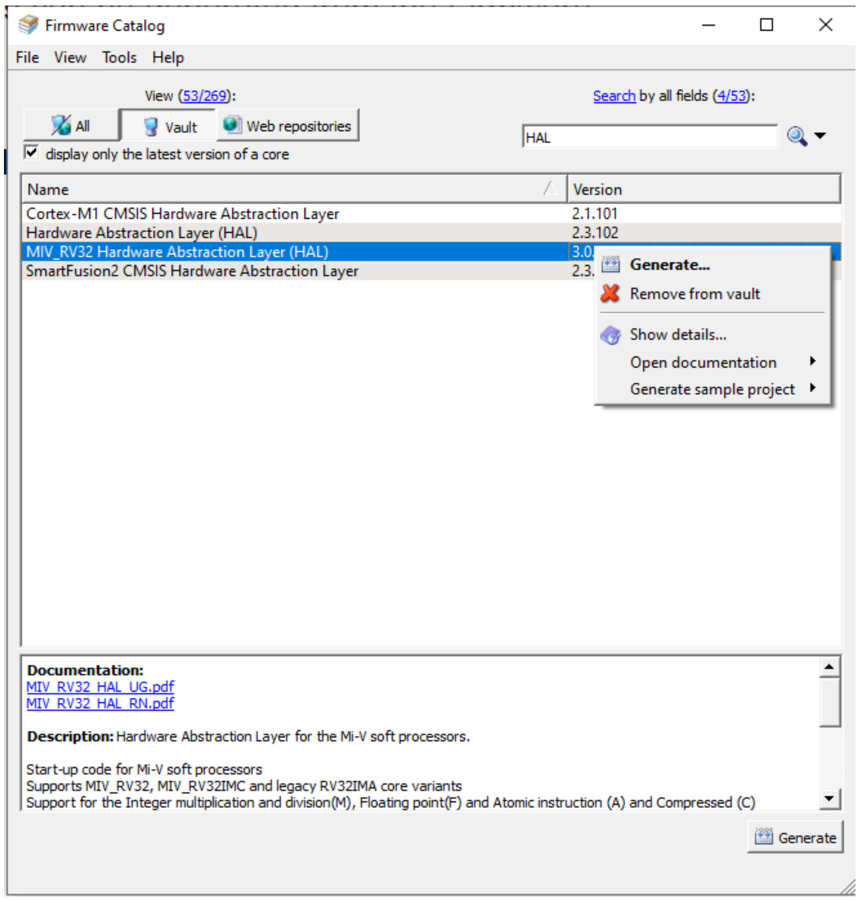


7. Select the configuration that matches your processor design. Make any application specific changes like pre-processor, include paths and so on.
8. Click **Apply and Close**. The same process must be followed to build the release target.
9. Build the debug or release target. Fix any build errors that arise in the process.
10. Use the default build configurations and look for any application specific settings.
11. Launch the application in debug mode to test the functionality.

3.4 Updating the MIV_RV32 HAL

The MIV_RV32 HAL can be updated to the latest version. The source files are generated from the Firmware Catalog, which is installed with Libero SoC Software.

1. Open the **Firmware Catalog** and search for **hal**.
2. Right-click **MIV_RV32 Hardware Abstraction Layer (HAL)** and click **Generate**. Select a location for the HAL update files.



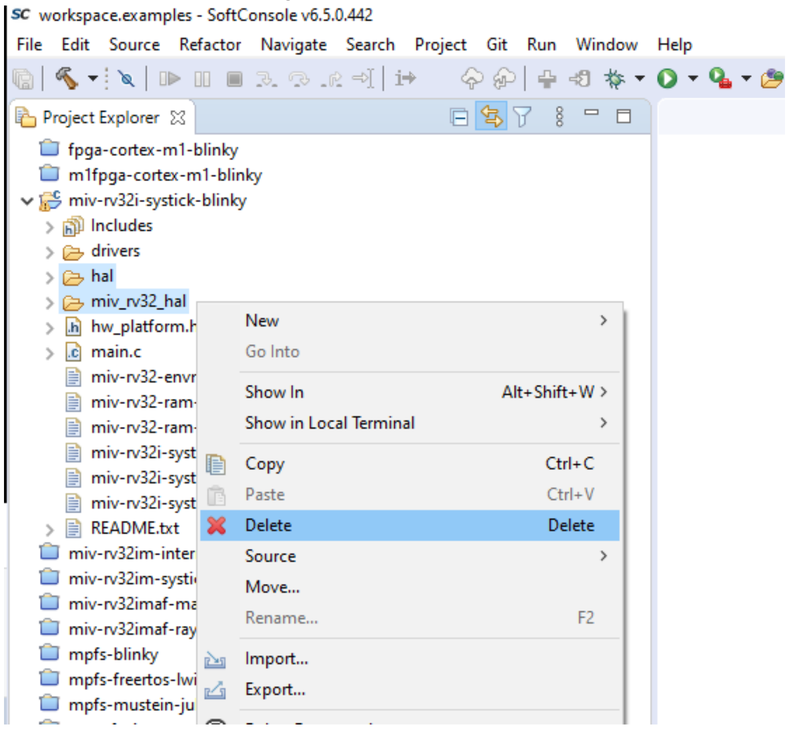
Name	Version
Cortex-M1 CMSIS Hardware Abstraction Layer	2.1.101
Hardware Abstraction Layer (HAL)	2.3.102
MIV_RV32 Hardware Abstraction Layer (HAL)	3.0
SmartFusion2 CMSIS Hardware Abstraction Layer	2.3.

Documentation:
MIV_RV32_HAL_UG.pdf
MIV_RV32_HAL_RN.pdf

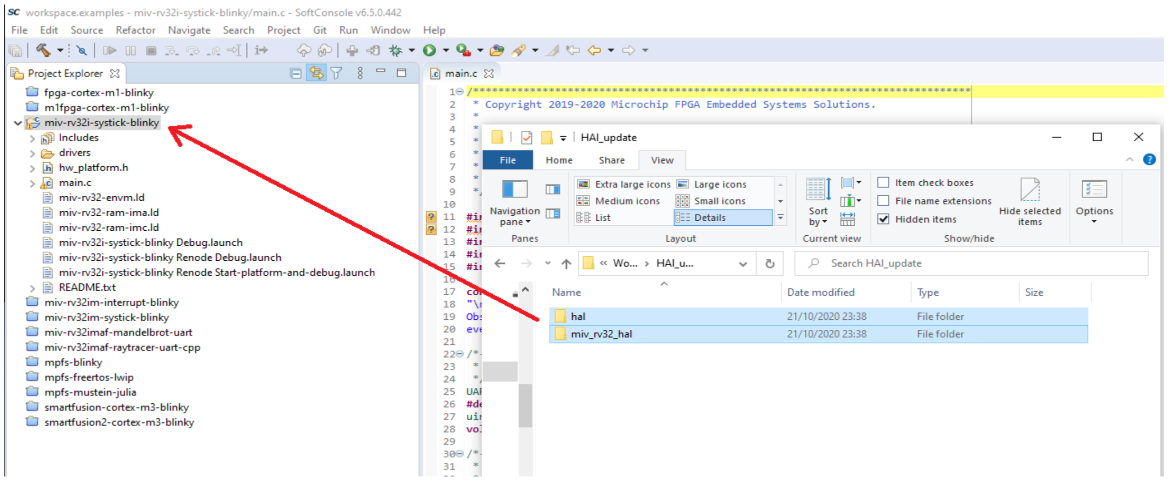
Description: Hardware Abstraction Layer for the Mi-V soft processors.

Start-up code for Mi-V soft processors
Supports MIV_RV32, MIV_RV32IMC and legacy RV32IMA core variants
Support for the Integer multiplication and division(M), Floating point(F) and Atomic instruction (A) and Compressed (C)

3. After generating the HAL update, the **hal** and **miv_rv32_hal** folders from the SoftConsole project must be updated. Note that the existing project specific linker script in the **miv_rv32_hal** folder will be over written. If required, it must be backed up before deleting the folder.

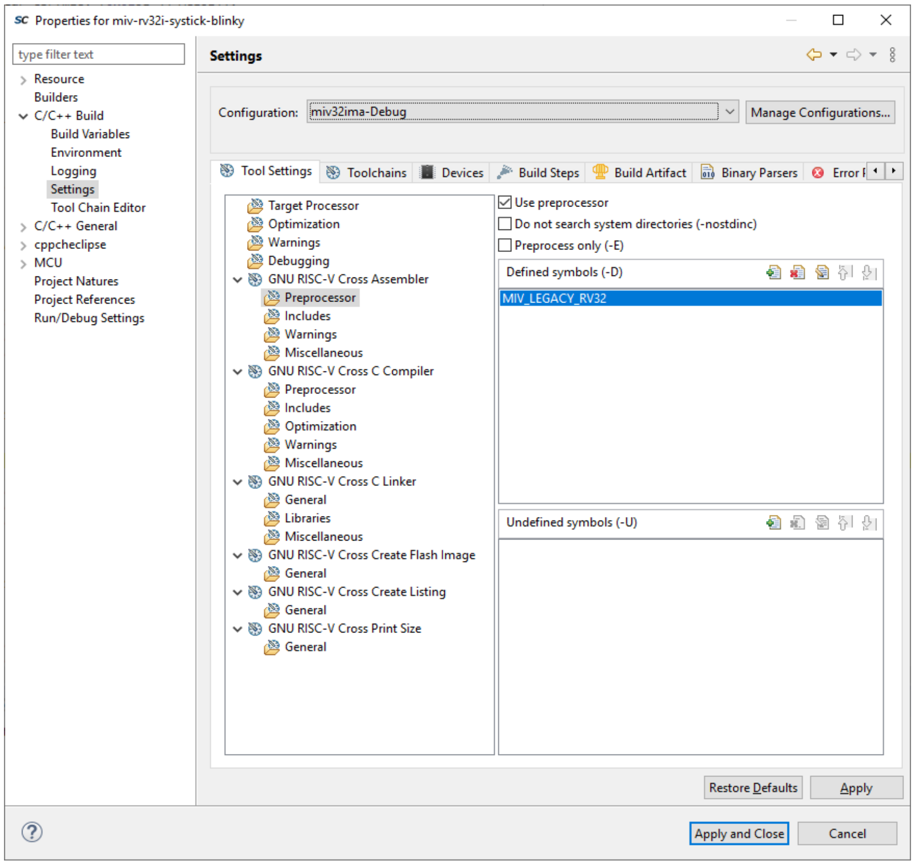


4. Copy the generated **hal** and **miv_rv32_hal** folders, generated by the Firmware Catalog into the SoftConsole project.


workspace.examples - miv-rv32i-systick-blinky/main.c - SoftConsole v6.5.0.442
File Edit Source Refactor Navigate Search Project Git Run Window Help
Project Explorer
fpga-cortex-m1-blinky
m1fpga-cortex-m1-blinky
miv-rv32i-systick-blinky
Includes
drivers
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.launch
README.txt
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
* Copyright 2019-2020 Microchip FPGA Embedded Systems Solutions.
HAL_update
File Home Share View
Navigation pane
Extra large icons Large icons Medium icons Small icons List Details
Sort by
Item check boxes File name extensions Hidden items
Hide selected items
Options
Panes Layout Current view Show/hide
« Wo... › HAL_u...
Search HAL_update
Name Date modified Type Size
hal 21/10/2020 23:38 File folder
miv_rv32_hal 21/10/2020 23:38 File folder

3.5 Defining the Core to the HAL

The `MIV_RV32` and `MIV_Legacy` cores handle traps and interrupts differently. They also have different methods of causing internal interrupts. Each one includes unique interrupts, for example, a PLIC or ECC errors. Due to this, the HAL must be configured for the core that is being used, by defining a symbol in the SoftConsole project properties. This symbol must be defined in the following preprocessor settings. In **C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross Assembler > Preprocessor**, the symbol `MIV_LEGACY_RV32` must be defined, if an `MIV_Legacy` core is used.



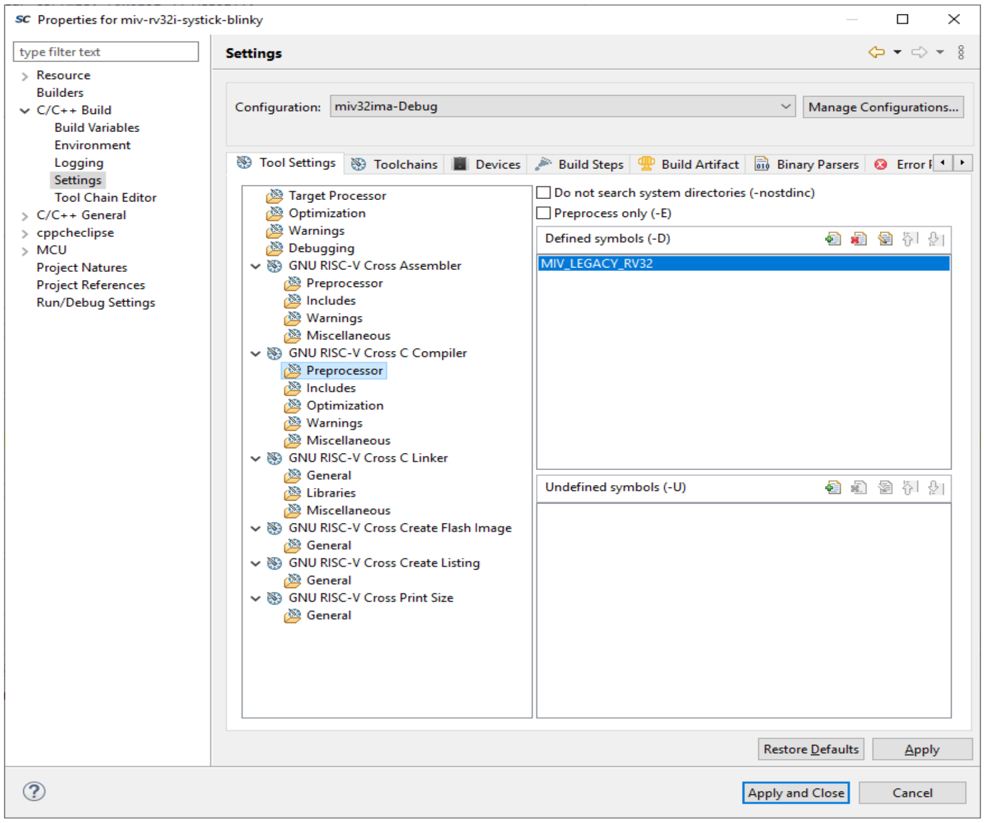
Configuration: miv32ima-Debug

☑ Use preprocessor
☐ Do not search system directories (-nostdinc)
☐ Preprocess only (-E)

Defined symbols (-D)
MIV_LEGACY_RV32

Undefined symbols (-U)

In **C/C++ Build** > **Settings** > **Tool Settings** > **GNU RISC-V Cross Compiler** > **Preprocessor**, the symbol **MIV_LEGACY_RV32** must be defined, if an `MIV_Legacy` core is used. If `MIV_RV32` or `MIV_RV32IMC` core is used, then no symbol needs to be defined.



Configuration: miv32ima-Debug

☐ Do not search system directories (-nostdinc)
☐ Preprocess only (-E)

Defined symbols (-D)
MIV_LEGACY_RV32

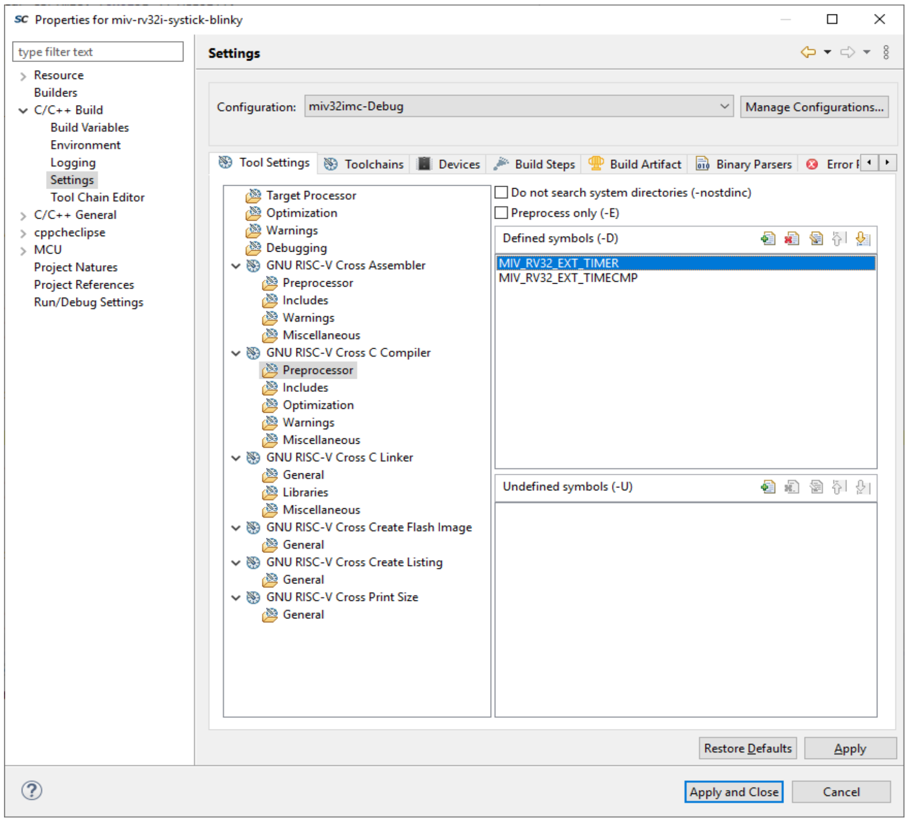
Undefined symbols (-U)

Two additional defines can be included, depending on the configuration of the `MIV_RV32` core to define the system timer configuration of the core:

- **MIV_RV32_EXT_TIMER**
- **MIV_RV32_EXT_TIMECMP**

These symbols must be defined only when the internal MTIME and internal MTIMECMP options are not selected in the IP Core configurations. These symbols must be defined in the following project settings path.

C/C++ Build > **Settings** > **Tool Settings** > **GNU RISC-V Cross Compiler** > **Preprocessor**.


Properties for miv-rv32i-systick-blinky

Configuration: miv32imc-Debug

Tool Settings > GNU RISC-V Cross C Compiler > Preprocessor

Do not search system directories (-nostdinc)

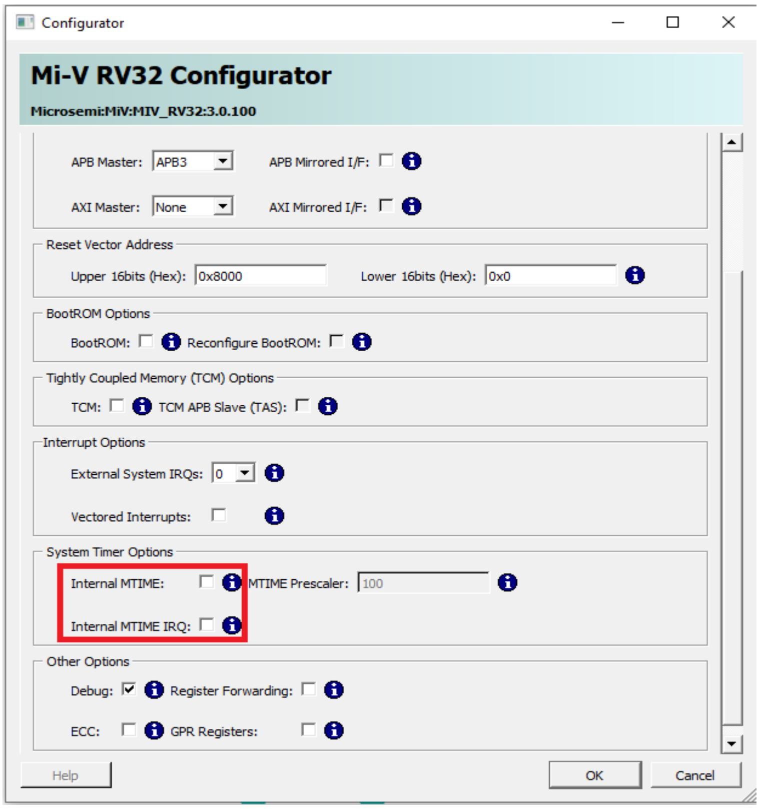
Preprocess only (-E)

Defined symbols (-D)

MIV_RV32_EXT_TIMER
MIV_RV32_EXT_TIMECMP

Undefined symbols (-U)

When the "Internal MTIME" and "Internal MTIMECMP" are enabled in the core (default configuration). The SoftCosole project uses the same default settings and it works without adding any symbols to the project settings.


Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

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: ☐

3.6 Interrupts

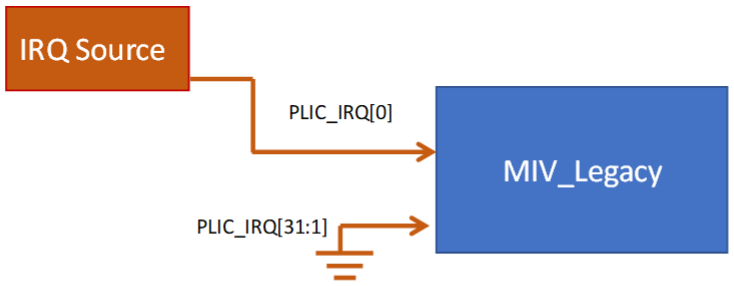
In MIV_RV32 HAL v3.0.x or greater handler, names for the standard RISC-V interrupts have not been changed, that is, external interrupt, software interrupt, and timer interrupt. If the `MIV_LEGACY_RV32` symbol is defined in the GCC pre-processor, the software is built to support the `MIV_Legacy` cores. If the symbol is not defined, then the software will be built to support the `MIV_RV32` core (as well as the `MIV_RV32IMC` core).

In the `MIV_Legacy` cores, PLIC interrupts cause the external interrupt to assert, and the core determines which interrupt in the PLIC has occurred and jump to its handler. In the `MIV_RV32` core, each interrupt has its own encoding and a PLIC does not need to be polled to determine which interrupt has occurred.

3.6.1 Sample Design 11 – Single Interrupt Source

3.6.1.1 MIV_Legacy Configuration

In this sample design, `MIV_Legacy` has one interrupt source with the remaining 31 PLIC interrupts tied low.

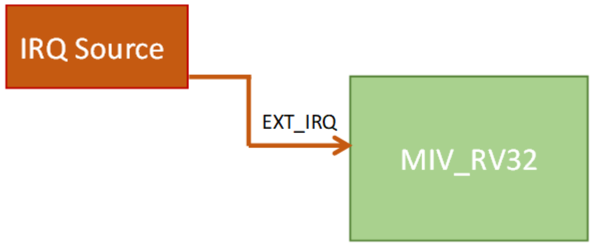


The following code block is a software implementation of the interrupt handling.

```
uint8_t External_1_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
```

3.6.1.2 MIV_RV32 Configuration

The interrupt source is connected to the `EXT_IRQ` input of `MIV_RV32`.



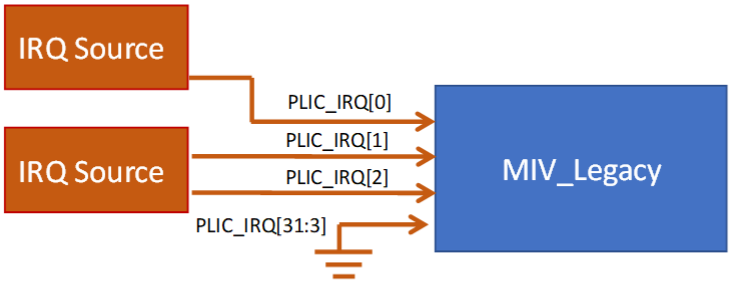
The following code block is a software implementation of the interrupt handling.

```
uint8_t External_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
```

3.6.2 Sample Design 12 – Multiple Interrupt Sources

3.6.2.1 MIV_Legacy Configuration

In this sample design, `MIV_Legacy` has an interrupt source generating an interrupt for PLIC_IRQ[0] and a second source generating interrupts for PLIC_IRQ[1] and PLIC_IRQ[2] with the remaining PLIC interrupts tied low.

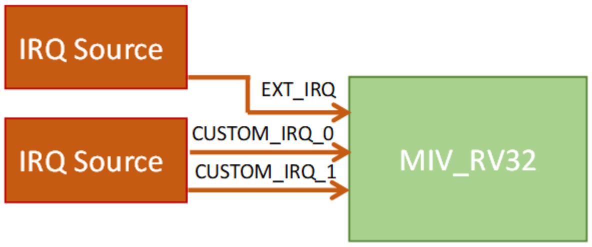


The following code block is a software implementation of the interrupt handling.

```
uint8_t External_1_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
uint8_t External_2_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
uint8_t External_3_IRQHandler()
{
    return(EXT_IRQ_KEEP_ENABLED);
}
```

3.6.2.2 MIV_RV32 Configuration

The interrupt source generating a single interrupt is connected to the EXT_IRQ core; the source generating the two second interrupts is connected to two of the custom external interrupts.



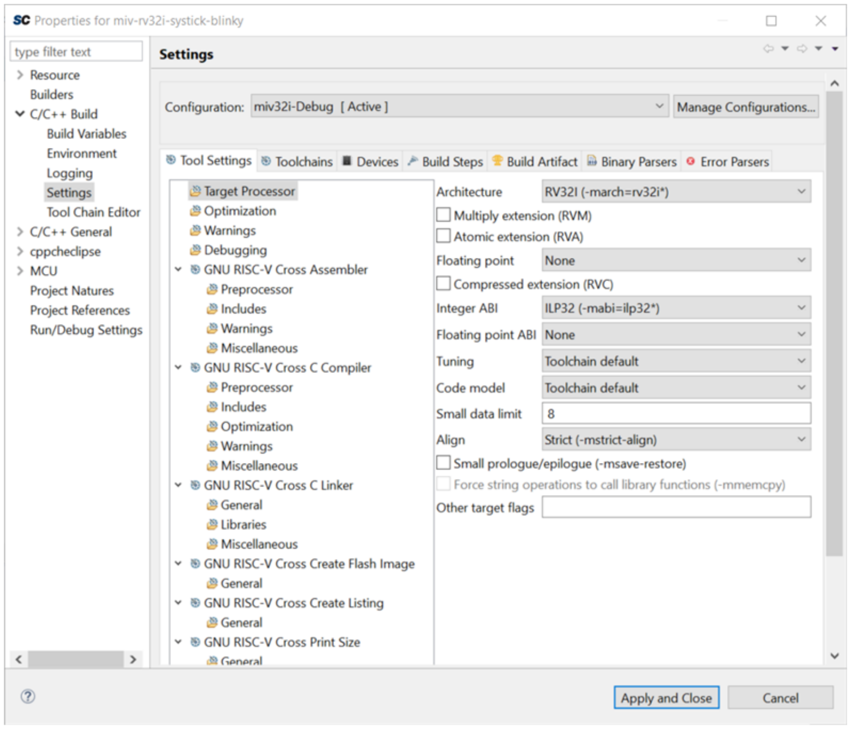
The core configuration to enable `CUSTOM_IRQ_0` and `CUSTOM_IRQ_1` is as follows:

The following code block is a software implementation of the interrupt handling.

```
void External_IRQHandler()
{

}
void MSYS_E10_IRQHandler(void)
{

}
void MSYS_E11_IRQHandler(void)
{

}
```

3.7 MIV_RV32 Extensions

As `MIV_RV32` supports any configuration of RV32I, RV32IM, RV32IC, or RV32IMC, the SoftConsole projects need to be configured appropriately. In the **Project Properties** > **C/C**, select the check box for the **Multiply extension (RVM)**, if the M Extension is included in the core, select the check box for the **Compressed extension (RVC)**, if the C Extension is included in the core.



Properties for miv-rv32i-systick-blinky

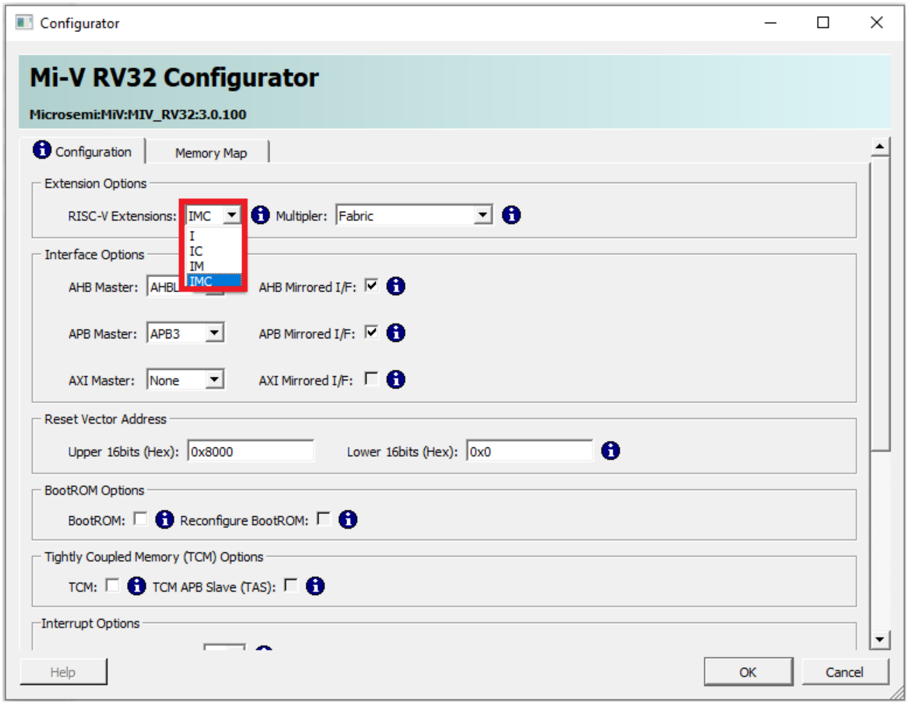
Settings

Configuration: miv32i-Debug [Active]

Tool Settings | Toolchains | Devices | Build Steps | Build Artifact | Binary Parsers | Error Parsers

Target Processor

- Architecture: RV32I (-march=rv32i*)
- Multiply extension (RVM): ☐
- Atomic extension (RVA): ☐
- Floating point: None
- Compressed extension (RVC): ☐
- Integer ABI: ILP32 (-mabi=ilp32*)
- Floating point ABI: None
- Tuning: Toolchain default
- Code model: Toolchain default
- Small data limit: 8
- Align: Strict (-mstrict-align)
- Small prologue/epilogue (-msave-restore): ☐
- Force string operations to call library functions (-mmemcpy): ☐
- Other target flags:



Configurator

Mi-V RV32 Configurator

Microsemi:MiV:MIV_RV32:3.0.100

Configuration | Memory Map

Extension Options

- RISC-V Extensions: IMC (I, IC, IM, IMC)
- Multiplier: Fabric

Interface Options

- AHB Master: AHBL
- 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: ☐

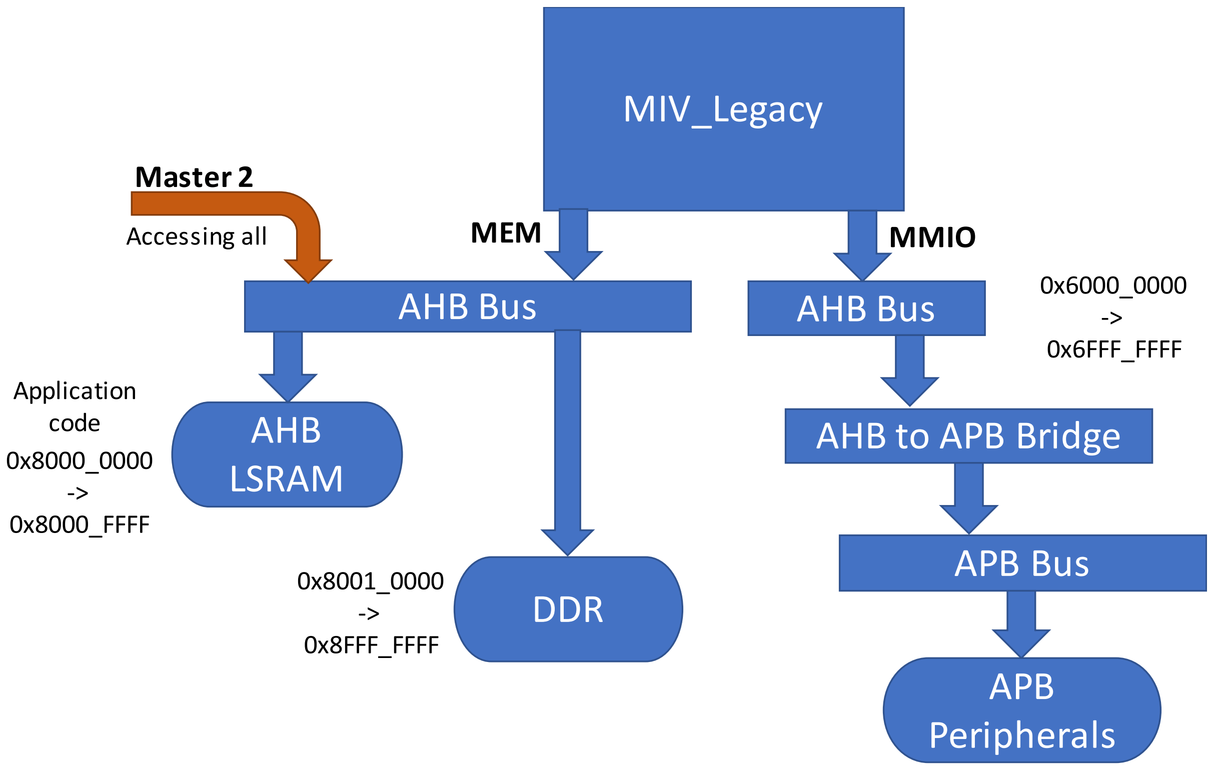
Tightly Coupled Memory (TCM) Options

- TCM: ☐
- TCM APB Slave (TAS): ☐

Interrupt Options

3.8 Maintaining Performance in Code Implementations From MIV_Legacy

In some use cases, certain code requirements are needed in software running on `MIV_Legacy`. The main requirement is that if memory is needed to appear consistent to another master accessing it, as shown in the following use case.



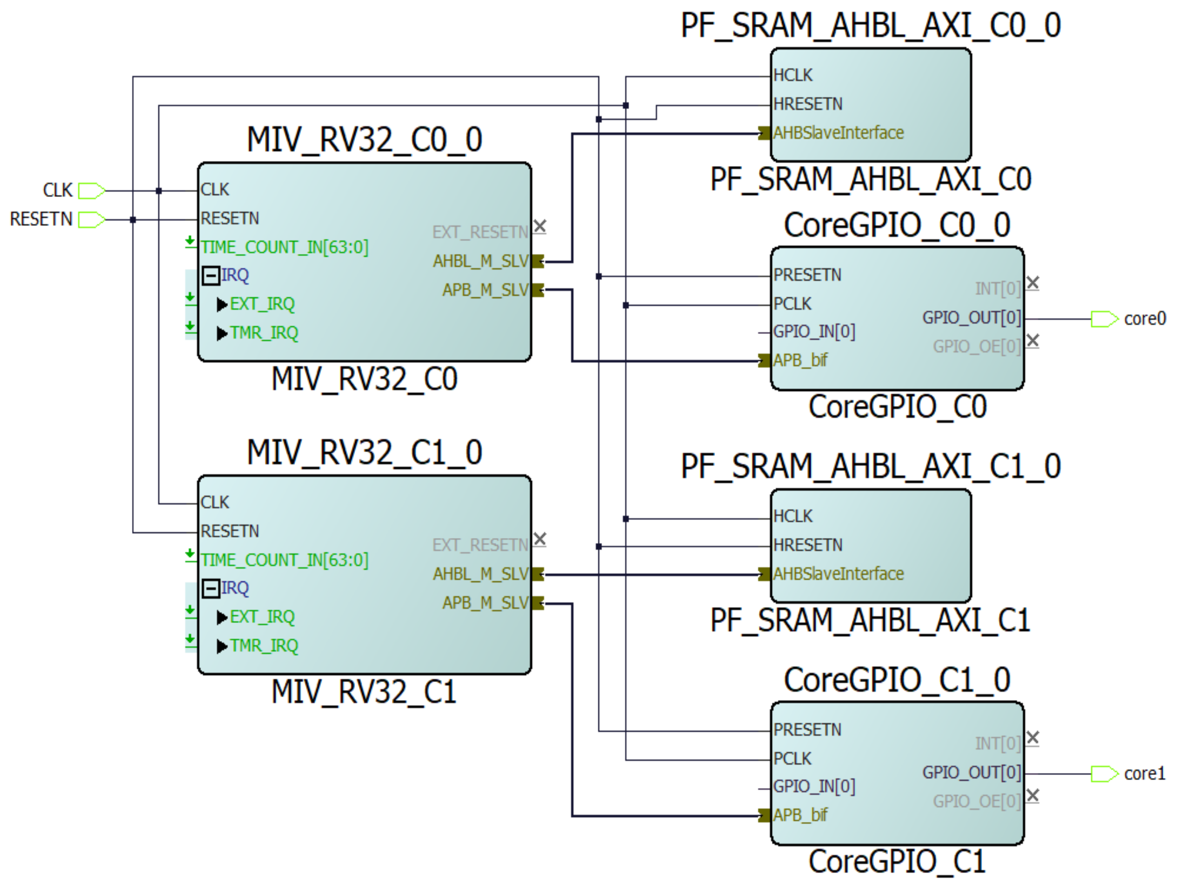
Due to the cache of `MIV_Legacy`, `fence` and `fence.i` instructions are required to make the memories in question appear consistently to the other masters. As there is no cache on `MIV_RV32`, there is no requirement to execute `fence` or `fence.i` instructions. Using the TAS to access, the TCM also appears consistently without `fence` or `fence.i` instructions.

Any ported code with `fence` or `fence.i` instructions still executes on the `MIV_RV32`. The instructions themselves have no effect when executed, but still need to be decoded, incurring a five-cycle delay.

Note: Any code that is ported from a `MIV_Legacy` configuration to a `MIV_RV32` configuration, should have any `fence` and `fence.i` instructions removed.

3.8.1 Latency of `fence` and `fence.i` Instructions

The following figure is an example system to test the delay added by `fence` and `fence.i` instructions.



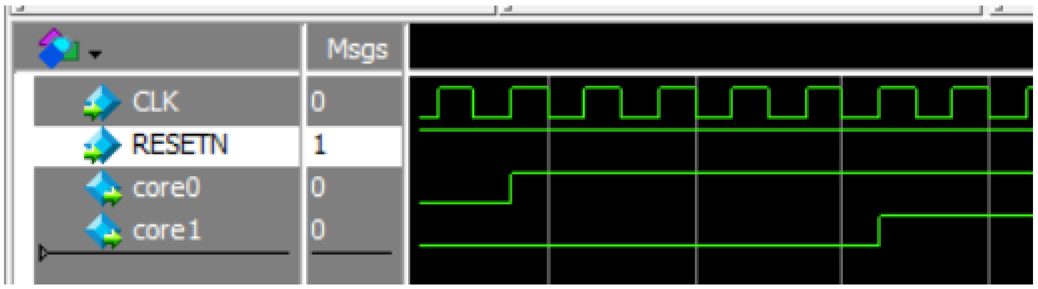
Core 0 boots and executes the following code in main.

```
int main(int argc, char **argv)
{
    GPIO_init(&g_gpio0, COREGPIO_IN_BASE_ADDR, GPIO_APB_32_BITS_BUS);
    GPIO_set_outputs(&g_gpio0, 1);
}
```

Core 1 boots and executes the following code in main.

```
int main(int argc, char **argv)
{
    asm volatile("fence.i");

    GPIO_init(&g_gpio0, COREGPIO_IN_BASE_ADDR, GPIO_APB_32_BITS_BUS);
    GPIO_set_outputs(&g_gpio0, 1);
}
```

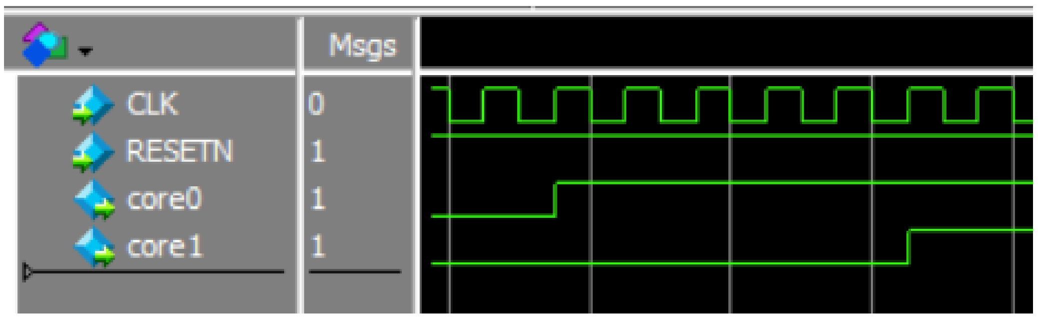
The time taken by each core to set its GPIO output indicates how long it takes to execute the code. The only difference between the code being executed on both cores is that the core 1 executes a `fence.i` instruction before initializing and setting its GPIO. The following figure shows that this result in core 1 settings is GPIO 5 cycles after core 0.



The code for core 1 is modified as in the following:

```
int main(int argc, char **argv)
{
    asm volatile("fence");

    GPIO_init(&g_gpio0, COREGPIO_IN_BASE_ADDR, GPIO_APB_32_BITS_BUS);
    GPIO_set_outputs(&g_gpio0, 1);
}
```

In this case, a fence instruction is executed instead of a `fence.i` instruction.



The same delay as seen with `fence.i` can been seen here. This delay is compounded every time and the instruction is executed.

4. Revision History

Revision	Date	Description
A	October 2020	Initial Revision

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

Legal Notice

Trademarks

ISBN: 978-1-5224-7048-9

Quality Management System

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

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
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	**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	**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	**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