

Partial Reconfiguration Controller v1.3

LogiCORE IP Product Guide

Vivado Design Suite

PG193 April 4, 2018

Table of Contents

IP Facts

Chapter 1: Overview

Feature Summary	7
Unsupported Features	10
Licensing and Ordering Information	10

Chapter 2: Product Specification

Overview	11
Performance	17
Resource Utilization	17
Port Descriptions	17
Register Space	21

Chapter 3: Designing with the Core

General Design Guidelines	32
Clocking	32
Resets	32
Virtual Socket Manager Control Interface	33
Protocol Description	36

Chapter 4: Design Flow Steps

Customizing and Generating the Core	51
Constraining the Core	63
Simulation	64
Synthesis and Implementation	64
Customizing the Core Post Implementation	64
Partial Bitstream Preparation	71

Appendix A: Migrating and Upgrading

Migrating to the Vivado Design Suite	74
Upgrading in the Vivado Design Suite	74

Appendix B: Debugging

Finding Help on Xilinx.com	75
Debug Tools	76
Hardware Debug	77

Appendix C: Additional Resources and Legal Notices

Xilinx Resources	79
References	79
Revision History	80
Please Read: Important Legal Notices	80

Introduction

The Xilinx® Partial Reconfiguration Controller core provides management functions for self-controlling partially reconfigurable designs. It is intended for enclosed systems where all of the Reconfigurable Modules are known to the controller. The optional AXI4-Lite register interface allows the core to be reconfigured at run time, so it can also be used in systems where the Reconfigurable Modules can change in the field.

The core can be customized for many Virtual Sockets, Reconfigurable Modules per Virtual Sockets, operations and interfaces.

Features

- Up to 32 Virtual Sockets
- Up to 128 Reconfigurable Modules per Virtual Socket
- Up to 512 remappable software and hardware triggers per Virtual Socket
- Optional hardware and software shutdown of Reconfigurable Modules (configurable per Reconfigurable Module)
- Optional software start-up of Reconfigurable Modules (configurable per Reconfigurable Module)
- Optional reset of Reconfigurable Modules after loading (configurable per Reconfigurable Module)
- Virtual Socket Managers can be shutdown and restarted by the user to allow external controllers to partially reconfigure the device
- User control of Virtual Socket Manager output signals is supported in the shutdown state

- All trigger and Reconfigurable Module information is configurable using the AXI4-Lite interface to allow for in-field upgrades
- Optional AXI4-Lite interface for control and status
- Optional AXI4-Stream status interface (per Virtual Socket)
- Optional AXI4-Stream control interface (per Virtual Socket)
- Optional bitstream decompression

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family ⁽¹⁾	UltraScale+™ UltraScale™, 7 Series
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	XDC
Simulation Model	Source HDL
Supported S/W Driver ⁽²⁾	Standalone
Tested Design Flows ⁽³⁾	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide .
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Standalone driver details can be found in the SDK directory (<install_directory>/SDK/<release>/data/embeddedsw/doc/xilinx_drivers.htm). Linux OS and driver support information is available from the [Xilinx Wiki page](#).
3. For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).

Overview

The Partial Reconfiguration Controller consists of one or more Virtual Socket Managers which connect to a single fetch path. A Virtual Socket (Figure 1-1) is a term that refers to a Reconfigurable Partition (RP) plus any logic that exists in the static logic to assist the RP with partial reconfiguration. For example, this logic could be used to isolate the static design from the Reconfigurable Partition while reconfiguration occurs, or to ensure Reconfigurable Modules are in a safe state before they are removed from the device. Some designs might not require this, in which case a Virtual Socket is equivalent to a Reconfigurable Partition.

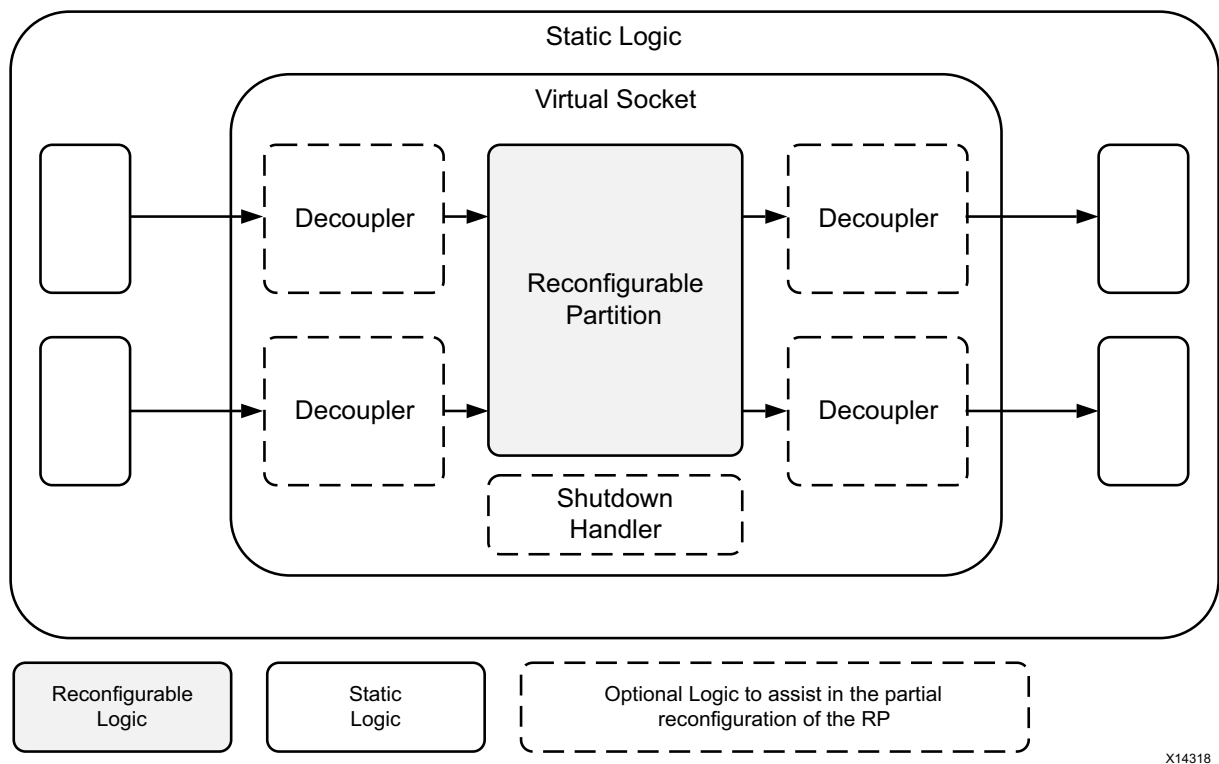


Figure 1-1: Virtual Socket

The fetch path fetches bitstreams from an external configuration library and sends them to the Internal Configuration Access Port (ICAP). The partial bitstreams are stored in a configuration library.

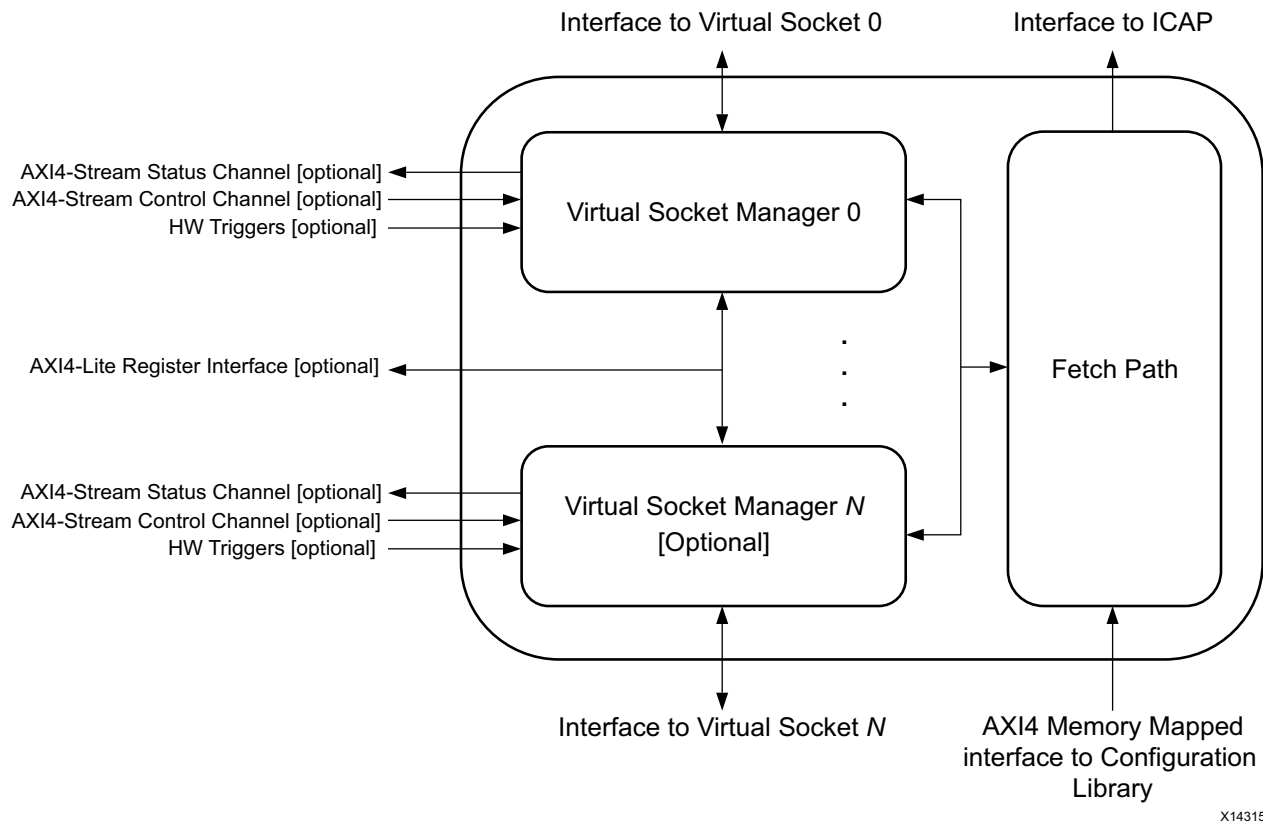


Figure 1-2: Architectural Block Diagram

Virtual Socket Managers operate in parallel watching for trigger events to occur. Triggers can be hardware based (signals) or software based (a register write). When a trigger is seen by a Virtual Socket Manager, the Virtual Socket Manager maps the trigger to a Reconfigurable Module and manages the reconfiguration of that Reconfigurable Module.

Each Virtual Socket Manager operates independently of the others, so while one Virtual Socket Manager is partway through the load of a Reconfigurable Module, another can start processing a trigger. Virtual Socket Managers have to queue for access to the fetch path. The actual reconfiguration of each module, however, remains sequential.

Virtual Socket Managers can be in one of two states:

- **Active state:** The Virtual Socket Manager is in control of the associated Virtual Socket. It reacts to triggers and loads Reconfigurable Modules.
- **Shutdown state:** Something else is in control of the associated Virtual Socket. The Virtual Socket Manager does not react to triggers and does not load Reconfigurable Modules.

These states are described in more detail in [Operational States](#).

Each Virtual Socket Manager can have its own AXI4-Stream Status and Control channels (independently optional) and can share a single AXI4-Lite register interface (also optional). These interfaces are not required for operation and can be omitted if the Partial Reconfiguration Controller can be fully independent in a particular system.

Feature Summary

Virtual Sockets and Reconfigurable Modules

The Partial Reconfiguration Controller supports up to 32 Virtual Sockets.

Each Virtual Socket can contain up to 128 Reconfigurable Modules, where each Reconfigurable Module is defined by one partial bitstream⁽¹⁾. Different Virtual Sockets can contain different numbers of Reconfigurable Modules. For example, *Virtual Socket 0* might have 32 Reconfigurable Modules, and *Virtual Socket 1* might only have 2 Reconfigurable Modules.

Remapable Software and Hardware Triggers

Reconfigurable Modules are loaded into a Virtual Socket in response to trigger activation. Each Virtual Socket can have hardware-based triggers and software-based triggers. The number of triggers per Virtual Socket is configurable, and the mapping from a particular trigger to a particular Reconfigurable Module is configurable during core configuration and at run time, if the AXI4-Lite interface is enabled. There can be more triggers than Reconfigurable Modules which allows for the addition of Reconfigurable Modules in the field, and for distributed control of partial reconfiguration. For more information, see [Hardware Triggers](#) and [SW_TRIGGER Register](#).

Reconfigurable Module Management

Loading a Reconfigurable Module is not always as simple as sending a partial bitstream to the ICAP. For example:

- An existing Reconfigurable Module might need to be deactivated to prevent system issues.
- The static logic might need to be protected from the signal values from the Virtual Socket during the reconfiguration interval.
- The new Reconfigurable Module might need to be integrated into the system and reset.

1. If the device being managed is an UltraScale device, each Reconfigurable Module also requires a Clearing bitstream.

The Partial Reconfiguration Controller provides support for all of these tasks and is configurable on a per-Reconfigurable Module basis.

Coexistence with Other Partial Reconfiguration Controllers

Partial reconfiguration occurs when a partial bitstream is loaded into one of several configuration ports (such as SelectMap, Serial, JTAG, ICAP, and PCAP in Zynq®-7000 AP SoC devices).



IMPORTANT: *It is vital that only one of these interfaces be used at a time, and that multiple controllers do not try to control the same Virtual Socket at the same time.*

The Partial Reconfiguration Controller IP core offers two mechanisms to support this:

1. A simple arbitration protocol is used to arbitrate the access to the configuration ports. You must supply an arbiter that is suitable for your system. If arbitration is not required, the arbitration signals can be tied to constant values. For more information, see [ICAP Sharing Protocol](#).
2. Each Virtual Socket Manager can be placed into a shutdown state to prevent it from trying to control the Virtual Socket. This allows other controllers (such as software or JTAG) to have exclusive control of a Virtual Socket. For more information, see [Shutdown State](#).

User Control of Virtual Socket Manager Outputs

When a Virtual Socket Manager is placed into the shutdown state, the signals that are used for Reconfigurable Module management are fully controllable from the AXI4-Lite interface and the AXI4-Stream Control interface. This allows the software, or another hardware component, to deactivate an existing Reconfigurable Module, protect the static logic during the reconfiguration interval, and integrate and reset the new Reconfigurable Module.

AXI4-Lite Interface for Control, Status, and Reprogramming

The Partial Reconfiguration Controller can be configured to have a single fully compliant AXI4-Lite interface for the Virtual Socket Managers. This interface can be used to:

- Access status information for each Virtual Socket Manager.
- Send commands to each Virtual Socket Manager.
- Reprogram the trigger and Reconfigurable Module information for each Virtual Socket Manager.

For more information, see [Register Space](#).

AXI4-Stream Channels for Status, and Control

The Partial Reconfiguration Controller can be configured to have fully compliant AXI4-Stream interfaces for each Virtual Socket Manager. These interfaces can be used to:

- Access status information for each Virtual Socket Manager.
- Send commands to each Virtual Socket Manager.

For more information, see [STATUS Register](#) and [CONTROL Register](#).

These channels can be configured individually for each Virtual Socket Manager. For example, the Virtual Socket Managers in an instance of the Partial Reconfiguration Controller could be configured as follows:

Table 1-1: Example Configuration of Partial Reconfiguration Controller

Virtual Socket Manager	Status Channel	Control Channel
0	No	No
1	No	Yes
2	Yes	No
3	Yes	Yes

Compatible with any Bitstream Storage Location

The Partial Reconfiguration Controller core fetches bitstream data from an AXI4 bus and, as a result, is not directly tied to any particular storage device. This allows the controller to access bitstreams no matter where they are stored, as long as a compatible AXI4 interface is available. The Vivado® IP catalog contains several such blocks of IP, such as the AXI External Memory Controller (`axi_emc`), and the Memory Interface Generator (MIG).

Note: The STARTUP primitive does not support loading of partial bitstreams. IP, such as AXI SPI or AXI EMC should not be configured to use the STARTUP primitive to clock or deliver partial bitstreams from external flash.

Bitstream Decompression

The Partial Reconfiguration Controller's API can be used to compress partial bitstreams, and the core configured to decompress them before passing them to the ICAP. This is useful if bitstream storage space is limited, or if the data path to the Partial Reconfiguration Controller is bandwidth limited.

If bitstream decompression is selected, then all bitstreams received by the core must be compressed. It is not possible to mix compressed and uncompressed bitstreams in the same core instance.

Note: This compression scheme differs from the built-in Multi-Frame-Write (MFW) scheme supported directly by `write_bitstream` and the configuration engine. Using `BITSTREAM.GENERAL.COMPRESS TRUE` (only) will result in an incorrect format for a partial bitstream for this decompression scheme (see [Partial Bitstream Preparation](#)). Both schemes can be used together, although using the PRC compression on bitstreams generated with `BITSTREAM.GENERAL.COMPRESS FALSE` *generally* results in smaller bitstreams.

Which scheme (or schemes) to use depends on your design goals. PRC compression reduces the amount of data that needs to be stored and transported to the PRC but it does not reduce the amount of data that needs to be passed through the ICAP. MFW compression reduces the amount of data that has to be passed through the ICAP but it does not compress the bitstreams by as much as the PRC compression scheme does. If your PR bottleneck is bitstream storage or transport over AXI, PRC compression should be used. If the bottleneck is the amount of time taken to pass data through the ICAP then MFW compression should be used. Both schemes can be used together to achieve both advantages.

Note: It is not possible to predict how much any particular partial bitstream will compress by as the amount of compression depends on the specifics of each partial bitstream. As a guide, we have measured between 30% and 70% compression on a suite of partial bitstreams that have 50% or more LUT and FF utilization. However, there is no guarantee that all partial bitstreams will fall within this range.

Unsupported Features

- The Partial Reconfiguration Controller cannot be configured using the Vivado `set_property` command. Instead, a custom `set_property` command is provided with the core. For more information, see [Configuring Tcl User Parameters](#).
- Encrypted bitstreams are not supported when a 7 series device is being controlled.
- Encrypted bitstreams can be used when an UltraScale or UltraScale+ device is being controlled. However, the PRC may be unable to fully recover if a Fetch error occurs during the load of an encrypted bitstream.

Licensing and Ordering Information



IMPORTANT: *In the Vivado Design Suite, the Partial Reconfiguration flow requires a license. Contact your local sales offices for pricing and ordering details [\[Ref 9\]](#).*

IP License Type

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the [Xilinx End User License](#). Information about this and other Xilinx LogiCORE IP modules is available at the [Xilinx Intellectual Property](#) page.

Product Specification

Overview

Operational States

Each Virtual Socket Manager can exist in two states:

- [Active State](#)
- [Shutdown State](#)

Each Virtual Socket Manager can be configured to start in either state after a reset, and commands can be used to move a Virtual Socket Manager between states. Additionally, a Virtual Socket Manager enters the shutdown state in the event of an error, unless it has been configured not to.

Active State

The active state is the main state of each Virtual Socket Manager, and is where partial reconfiguration is managed. Each Virtual Socket Manager follows a basic set of steps, as shown in [Figure 2-1](#). Note that dotted steps are optional.

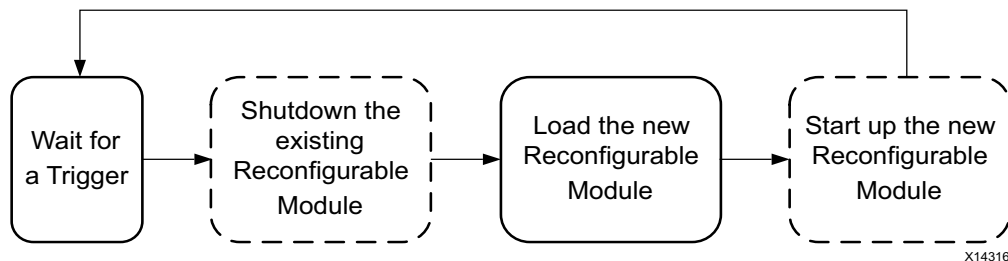


Figure 2-1: Basic Steps for Virtual Socket Manager in Active State

The Virtual Socket Manager starts by waiting for a trigger to arrive. When a trigger is seen, the Virtual Socket Manager starts to shutdown any Reconfigurable Module found in the Virtual Socket. This shutdown sequence can be configured on a per-Reconfigurable Module basis. The valid options are:

- **No shutdown is required.**
- **Hardware-only shutdown is required.** The Virtual Socket Manager informs the Reconfigurable Module that it will be removed, and waits until the Reconfigurable Module gives permission. This is intended for cases where arbitrarily removing a Reconfigurable Module could cause deadlock or other system corruption.
- **Shutdown of hardware and then software is required.** The Virtual Socket Manager performs hardware shutdown as described above, and then issues an interrupt informing software that the reconfigurable module will be removed. It then waits until the software gives permission. This is intended for cases where the system software may have to unload drivers or make other system changes.
- **Shutdown of software and then hardware is required.** As above, but with software shutdown performed first.

The protocol for hardware shutdown is as follows:

- The Virtual Socket Manager asserts a signal (`vsm_<name>_rm_shutdown_req`) High until the Reconfigurable Module gives permission to be removed.
- When the Reconfigurable Module is ready to be removed, it asserts `vsm_<name>_rm_shutdown_ack` High until the Reconfigurable Module is removed.

For software shutdown:

- The Virtual Socket Manager asserts `vsm_<name>_sw_shutdown_req` High until the **Proceed** command is received. For more information about this command, see the [Proceed Command](#).

See [Reconfigurable Module Hardware and Software Shutdown](#) for more information.

If only software shutdown is required for a particular Reconfigurable Module, the Reconfigurable Module should hardwire the `vsm_<name>_rm_shutdown_ack` signal to 1.

After the shutdown of any existing Reconfigurable Module is complete, the Virtual Socket Manager asserts `vsm_<name>_rm_decouple` High and starts to process the trigger⁽¹⁾. The signal `vsm_<name>_rm_decouple` remains asserted until the reconfigurable module is successfully loaded, and is intended for use with decoupling logic which can be required to isolate the Virtual Socket from the static logic while reconfiguration occurs.

1. When the device being managed is an UltraScale device, the clearing bitstream for the current Reconfigurable Module is loaded before the trigger is processed. `vsm_<name>_rm_decouple` is asserted High after the Reconfigurable Module is shutdown and before the clearing bitstream is loaded.

This decoupling logic is design specific and is not provided with the Partial Reconfiguration Controller core.

The Virtual Socket Manager then requests access to the fetch path⁽¹⁾. When it gains access to the fetch path, it configures the fetch path to load the correct bitstream for the new Reconfigurable Module. If this completes with no errors, the Virtual Socket Manager will start up the new Reconfigurable Module. There are two phases to this, each of which are optional and can be configured on a per-Reconfigurable Module basis:

- **Software startup:** If enabled, the Virtual Socket Manager issues an interrupt informing the software that the Reconfigurable Module has been loaded (if decoupling is implemented, then the Reconfigurable Module is still decoupled at this stage, and might not be operational), and waits until the software responds. This is intended for cases where the system software might have to load drivers or make other system changes. For more information, see [Reconfigurable Module Software Startup](#).
- **Reconfigurable Module Reset:** If enabled, the Virtual Socket Manager asserts a reset signal to the Reconfigurable Module to a configurable level for a configurable number of clock cycles.

The Virtual Socket Manager deasserts `vsm_<name>_rm_decouple` on entry to the Reconfigurable Module reset state.

At this stage, the Virtual Socket Manager starts searching for new triggers to process.

Shutdown State

The shutdown state is where the Virtual Socket Manager does not respond to triggers and does not load Reconfigurable Modules. There are several reasons why a Virtual Socket Manager should be shutdown:

- There has been an error loading a Reconfigurable Module. In this case, the Virtual Socket Manager shuts itself down, unless it has been configured not to.
- Something else, such as the Vivado® Design Suite Hardware Debugger or PCIe, needs to load bitstreams into a Virtual Socket.
- The Virtual Socket Manager needs to be reprogrammed to change the triggers and the Reconfigurable Modules.

The shutdown state can be entered by sending the **Shutdown** command to the Virtual Socket Manager's CONTROL register. This command cannot be canceled. For more information, see [Shutdown Command](#).

1. When the device being managed is an UltraScale device, the Virtual Socket Manager has previously requested access to the fetch path in order to load the clearing bitstream. A second request is needed to load the partial bitstream that loads the new Reconfigurable Module. Other Virtual Socket Managers might have used the fetch path in the interim to load their own bitstreams.

Exiting the Shutdown State

There are two commands available to exit the shutdown state:

- **Restart with no Status:** This command is used when you exit the shutdown state without having made any changes to the Virtual Socket. The Virtual Socket Manager will resume with the information it had before shutdown. For more information, see [Restart with no Status Command](#).
- **Restart with Status:** This command is used when the user exits the shutdown state after having made changes to the Virtual Socket. Specifically, this command must be used if a Reconfigurable Module is loaded into the Virtual Socket when the Virtual Socket Manager was in shutdown. The Virtual Socket Manager will resume with the information supplied by you as part of the command. For more information, see [Restart with Status Command](#).

User Control of Virtual Socket Manager Outputs

When a Virtual Socket is in the shutdown state, the following signals can be controlled using the **User Control** command:

- `vsm_<name>_rm_shutdown_req`
- `vsm_<name>_rm_decouple`
- `vsm_<name>_rm_reset`
- `vsm_<name>_sw_shutdown_req`
- `vsm_<name>_sw_startup_req`

This feature allows the system to take control of the Virtual Socket reconfiguration while still being able to manage hardware and software shutdown, decoupling, software start-up and Reconfigurable Module reset. The status of the `vsm_<name>_rm_shutdown_ack` signal can be retrieved from the STATUS register using the AXI4-Lite or AXI4-Stream status channel interfaces. See [User Control Command](#) and [STATUS Register](#).

Error Handling

The Partial Reconfiguration Controller detects and handles the following types of errors:

- **Fetch Errors:** These errors occur in the fetch path when a bitstream is being read from the configuration library interface. This type of error occurs if the AXI4 Memory Mapped bus connected to the configuration library returned an AXI response error.
- **Bitstream Errors:** These errors occur inside the FPGAs configuration interface, and typically indicate that the bitstream is corrupt.

- **Bad Configuration Errors:** These errors occur if a bitstream is configured as 0 bytes long. The fetch path detects and rejects requests for 0 byte bitstreams.
- **Lost Errors:** Lost errors occur when partial bitstreams are sent to the FPGA through one of the higher priority configuration ports (such as MCAP or JTAG) while the core is sending a partial bitstream to the ICAP. These errors can only be detected when the device to be managed is an UltraScale or UltraScale+ device.
- **Decompression Errors:**
 - **Bad Format Errors:** These errors occur when a compressed bitstream contains invalid information.
 - **Bad Size Errors:** These errors occur when a compressed bitstream ends in an invalid place in the decompression algorithm.

If a fetch error, bitstream error, lost error, bad format error, or bad size error occurs, the Partial Reconfiguration Controller responds as follows:

- The fetch path continues fetching the bitstream to maintain data path integrity.
- The ICAP interface continues to consume data to ensure the fetch path is drained, but it does not pass any more data to the ICAP.
- If the error is a fetch error, and it occurred on the first word of the bitstream, the ICAP interface does not pass anything to the ICAP.
- If the error is a fetch error, and it occurred on the second word of the bitstream, or beyond, the ICAP interface does not pass anything else to the ICAP. It drains the fetch path as described above, and then injects a DESYNC sequence into the ICAP.
- The Virtual Socket Manager that was attempting to load the failing bitstream will enter the shutdown state, and does not process any more triggers until the Virtual Socket Manager is restarted. This behavior is configurable when the core is generated, in which case the Virtual Socket Manager will not enter the shutdown state.
- The error will be reported on the status interfaces.

If the error is a bad configuration error, the Partial Reconfiguration Controller responds as follows:

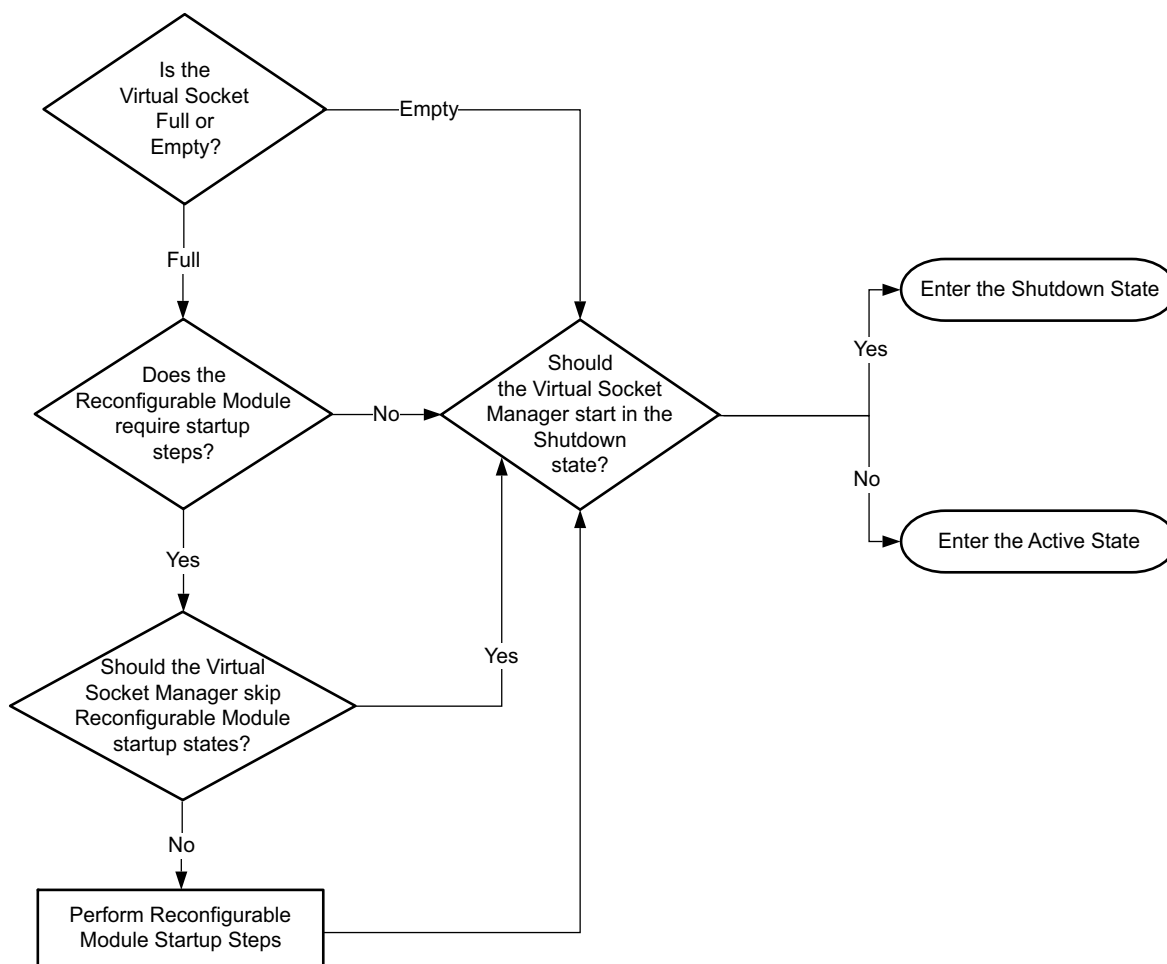
- The fetch path rejects the request and issues an error.
- The Virtual Socket Manager that was trying to load the failing bitstream enters the shutdown state, and does not process any more triggers until the Virtual Socket Manager is restarted. This behavior is configurable when the core is generated, so the Virtual Socket Manager can be configured not to enter the shutdown state. In this case, it will start monitoring for new triggers.
- The error will be reported on the status interfaces.

Post Reset Behavior

The behavior of a Virtual Socket Manager leaving reset (either a hard power-on reset or a soft reset) depends on many factors:

- If the Virtual Socket Manager is full or empty. A full Virtual Socket is one that contains a Reconfigurable Module. An empty Virtual Socket does not contain a Reconfigurable Module.
- If the Virtual Socket Manager is configured to start in the shutdown state.
- If the Reconfigurable Module in the Virtual Socket is configured to have a start-up phase (software start-up and/or reset).
- If the Virtual Socket Manager is configured to skip Reconfigurable Module start-up after reset.

Figure 2-2 shows the behavior of the Virtual Socket Manager after leaving reset.



X14317

Figure 2-2: Virtual Socket Manager Behavior After Leaving Reset

Performance

For details about performance, visit [Performance and Resource Utilization](#).

Resource Utilization

For details about resource utilization, visit [Performance and Resource Utilization](#).

Port Descriptions

In the following table, <name> is a user-defined name. For example, vsm_<name>_hw_triggers could be:

- vsm_shift_hw_triggers, where <name> = shift.
- vsm_count_hw_triggers, where <name> = count.

Table 2-1: Port Descriptions

Name	Direction	Description
clk	Input	Rising-edge clock.
reset	Input	Synchronous reset. The active level is configurable. Reset has to be asserted for at least three clock cycles.
vsm_<name>_hw_triggers	Input	Hardware trigger input for Virtual Socket Manager with name <name>. The width is configurable. Only present if the number of HW Triggers is greater than zero. A 0 to 1 transition on bit <i>N</i> activates trigger <i>N</i> . Triggers cannot be canceled.
vsm_<name>_rm_shutdown_req	Output	Active-High signal from the IP to the Reconfigurable Module in Virtual Socket <name>. Set to 1 by the IP when the Reconfigurable Module is to be removed. This functionality can be disabled on a per-Reconfigurable Module basis.
vsm_<name>_rm_shutdown_ack	Input	Active-High signal from the Reconfigurable Module in Virtual Socket <name> to the IP. Set to 1 by the Reconfigurable Module when the Reconfigurable Module can be removed. This functionality can be disabled on a per-Reconfigurable Module basis.
vsm_<name>_rm_decouple	Output	Active-High signal from the IP to any decoupling logic separating Virtual Socket <name> from the static logic. Set to 1 by the IP when a Reconfigurable Module is being loaded. This signal should be used to control any Virtual Socket decoupling logic.

Table 2-1: Port Descriptions (Cont'd)

Name	Direction	Description
vsm_<name>_rm_reset	Output	Reset signal from the IP to the Reconfigurable Module in Virtual Socket <name>. The use of this signal, its active level and duration are configurable on a per-Reconfigurable Module basis.
vsm_<name>_sw_shutdown_req	Output	Active-High signal intended as a CPU interrupt. Set to 1 by Virtual Socket <name> when the active Reconfigurable Module is to be removed. Set to 0 when the Proceed command is written to the Virtual Socket Manager's CONTROL register. This functionality can be configured on a per-Reconfigurable Module basis.
vsm_<name>_sw_startup_req	Output	Active-High signal intended as a CPU interrupt. Set to 1 by Virtual Socket <name> when the new Reconfigurable Module has been loaded. Set to 0 when the Proceed command is written to the Virtual Socket Manager's CONTROL register. This functionality can be enabled or disabled on a per-Reconfigurable Module basis.
vsm_<name>_m_axis_status_tvalid	Output	TVALID signal for the AXI4-Stream Status channel of the Virtual Socket Manager <name>. This signal is always set to 1 as status is always available. This channel can be enabled or disabled on a per-Virtual Socket Manager basis.
vsm_<name>_m_axis_status_tdata	Output	32-bit wide tdata signal for the AXI4-Stream Status channel of the Virtual Socket Manager <name>. This has the same format as the STATUS register (see STATUS Register for details). This channel can be enabled or disabled on a per-Virtual Socket Manager basis.
vsm_<name>_s_axis_ctrl_tvalid	Input	tvalid signal for the AXI4-Stream Control channel of the Virtual Socket Manager <name>. This channel can be enabled or disabled on a per-Virtual Socket Manager basis.
vsm_<name>_s_axis_ctrl_tready	Output	tready signal for the AXI4-Stream Control channel of the Virtual Socket Manager <name>. This channel can be enabled or disabled on a per-Virtual Socket Manager basis.
vsm_<name>_s_axis_ctrl_tdata	Input	32-bit wide tdata signal for the AXI4-Stream Control channel of the Virtual Socket Manager <name>. This has the same format as the CONTROL Register (see CONTROL Register for details). This channel can be enabled or disabled on a per-Virtual Socket Manager basis.
vsm_<name>_event_error	Output	Asserted for a single clock cycle when an error occurs in the Virtual Socket Manager.
icap_clk	Input	Rising-edge clock. This must be the same clock that is attached to the ICAP primitive.

Table 2-1: Port Descriptions (Cont'd)

Name	Direction	Description
icap_reset	Input	Synchronous reset. Active level is configurable. This reset is used to reset the ICAP interface logic, and needs to be synchronous to the <code>icap_clk</code> .
icap_i	Input	Status data returning from the ICAP primitive. Connect to the ICAP's <code>o</code> port.
icap_o	Output	The data to the ICAP primitive. Connect to the ICAP's <code>i</code> port.
icap_csib	Output	The CSIB signal to the ICAP primitive. Connect to the ICAP's <code>csib</code> port.
icap_rdwrb	Output	The RDWRB signal to the ICAP primitive. Connect to the ICAP's <code>rdwrb</code> port.
icap_avail	Input	Connect to the ICAP AVAIL port. This port is only available when the device to be managed is an UltraScale or UltraScale+ device.
icap_prdone	Input	Connect to the ICAP PRDONE port. This port is only available when the device to be managed is an UltraScale or UltraScale+ device.
icap_prerror	Input	Connect to the ICAP PRERROR port. This port is only available when the device to be managed is an UltraScale or UltraScale+ device.
cap_req	Output	This signal is not present if <code>C_ARBITRATION_PROTOCOL</code> is set to 0. For use with an ICAP arbiter. This signal is asserted by the IP on every clock cycle where it has data to transfer to the ICAP.
cap_gnt	Input	This signal is not present if <code>C_ARBITRATION_PROTOCOL</code> is set to 0. For use with an ICAP arbiter. This signal should be asserted by the arbiter on every clock cycle where the IP has access to the ICAP. When set to 1, this signal should remain at 1 until <code>cap_req</code> returns to 0.
cap_rel	Input	This signal is not present if <code>C_ARBITRATION_PROTOCOL</code> is set to 0. For use with an ICAP arbiter. This signal should be asserted by the arbiter on every clock cycle where something else is requesting access to the ICAP. When set to 1, this signal should remain at 1 until <code>cap_req</code> returns to 0. This signal indicates to the IP that it should relinquish control of the ICAP at the earliest safe opportunity.
s_axi_reg_awaddr	Input	Standard 32-bit wide AXI4-Lite signal for the optional register interface.
s_axi_reg_awvalid	Input	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_awready	Output	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_wdata	Input	Standard 32-bit wide AXI4-Lite signal for the optional register interface.
s_axi_reg_wvalid	Input	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_wready	Output	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_bresp	Output	Standard 2-bit wide AXI4-Lite signal for the optional register interface.

Table 2-1: Port Descriptions (Cont'd)

Name	Direction	Description
s_axi_reg_bvalid	Output	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_bready	Input	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_araddr	Input	Standard 32-bit wide AXI4-Lite signal for the optional register interface.
s_axi_reg_arvalid	Input	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_arready	Output	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_rdata	Output	Standard 32-bit wide AXI4-Lite signal for the optional register interface.
s_axi_reg_rresp	Output	Standard 2-bit wide AXI4-Lite signal for the optional register interface.
s_axi_reg_rvalid	Output	Standard AXI4-Lite signal for the optional register interface.
s_axi_reg_rready	Input	Standard AXI4-Lite signal for the optional register interface.
m_axi_mem_araddr	Output	Standard 32-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_arlen	Output	Standard 8-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_arsize	Output	Standard 3-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_arburst	Output	Standard 2-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_arprot	Output	Standard 3-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_arcache	Output	Standard 4-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_aruser	Output	Standard 4-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_arvalid	Output	Standard AXI4 signal for the configuration library interface.
m_axi_mem_arready	Input	Standard AXI4 signal for the configuration library interface.
m_axi_mem_rdata	Input	Standard 32-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_rresp	Input	Standard 2-bit wide AXI4 signal for the configuration library interface.
m_axi_mem_rlast	Input	Standard AXI4 signal for the configuration library interface.
m_axi_mem_rvalid	Input	Standard AXI4 signal for the configuration library interface.
m_axi_mem_rready	Output	Standard AXI4 signal for the configuration library interface.

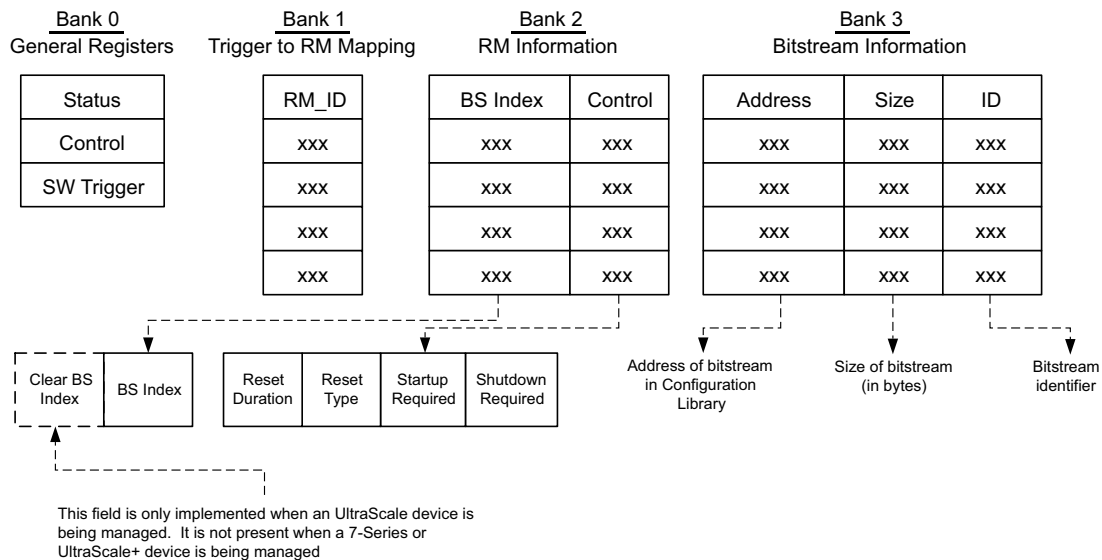
Register Space

Overview

Each Virtual Socket Manager has a set of registers that can be accessed through the optional AXI4-Lite interface. Each register is 32-bits wide, although some bits might not be used in some cases. The registers in each Virtual Socket Manager are split into four banks, where each bank has its own unique structure:

- Bank 0: General Registers
- Bank 1: Trigger to Reconfigurable Module Registers
- Bank 2: Reconfigurable Module Information Registers
- Bank 3: Bitstream Information Registers

Figure 2-3 provides an overview of the registers.



X14313

Figure 2-3: Register Banks within a Virtual Socket Manager

The *General registers* are used to control the Virtual Socket Manager, and retrieve its status information. The remaining banks are used to store information required to load a Reconfigurable Module.

The *Trigger to Reconfigurable Module registers*, the *Reconfigurable Module registers* and the *Bitstream Information registers* operate as follows:

- A trigger identifier⁽¹⁾ is decoded and used to select a row in the Trigger to Reconfigurable Module register bank (Bank 1). The selected register holds the identifier of the Reconfigurable Module to be loaded by that trigger (RM_ID).
- The RM_ID is used to select a row of registers in the Reconfigurable Module Information register bank (Bank 2).
- The lower register in the selected row (CONTROL) provides information about how the Reconfigurable Module is to be shut down and started.
- The upper register in the selected row (BS Index) provides the row number in the Bitstream Information register bank to access to get information about the bitstream required to load the Reconfigurable Module. This information is the address in external memory at which the bitstream is stored, its size in bytes, and its identifier.

The address of each register is encoded for as follows:

```
[Virtual Socket Manager Select] [Bank Select] [Register Select][00]
```

where the most significant bit is on the left.

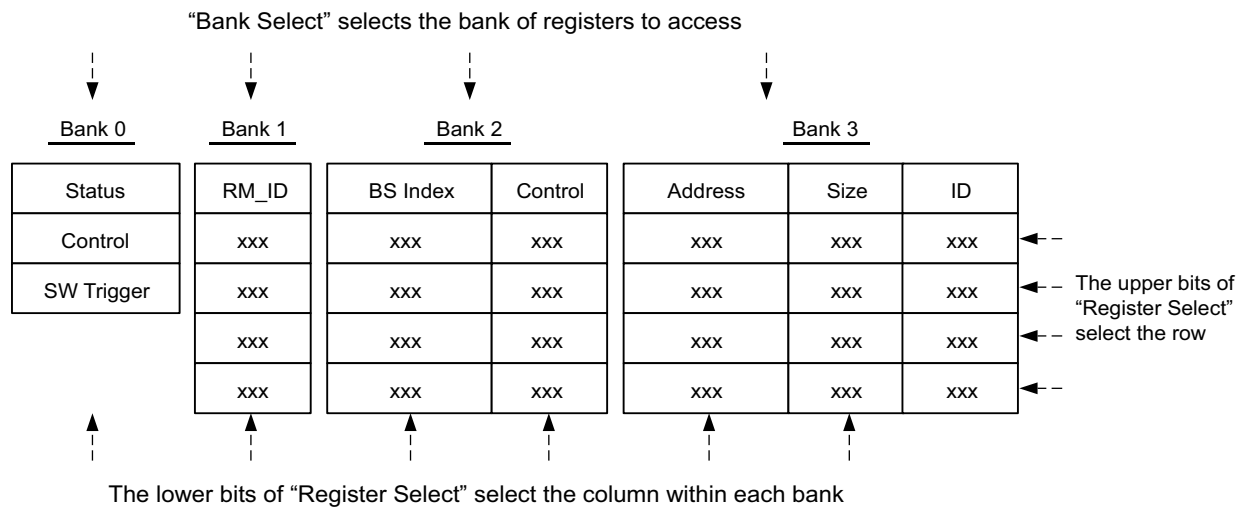
Each address segment is defined in [Table 2-2](#).

Table 2-2: Address of Each Register

Segment of Address	Description
Virtual Socket Manager Select	These bits contain the identifier of the Virtual Socket Manager. These identifiers are reported in the Configuration Information text file produced when the IP is generated.
Bank Select	An identifier representing the bank of registers to access. 0 = The General Register bank. 1 = The Trigger to Reconfigurable Module register bank. 2 = The Reconfigurable Module Information register bank. 3 = The Bitstream Information register bank.
Register Select	This segment of the address has different interpretations depending on the bank being accessed. The upper bits give the row of the bank to select, and the lower bits give the column within the bank to select. If a bank has one column, zero bits are required to select the column. If a bank has two columns, one bit is required to select the column, etc. See Figure 2-4 .

All slice ranges are identical across all Virtual Socket Managers. These, and each register address, are available in the configuration information text file that is produced when the IP is generated. See [Output Generation](#) for more details.

1. Only hardware triggers need to be decoded. Software triggers directly specify the row in the Trigger to Reconfigurable Module register bank



X14314

Figure 2-4: Mapping Address Fields to Registers

Bank 0: General Registers

The general registers are defined in [Table 2-3](#).

Table 2-3: Bank 0 - General Registers

Register Select Value	Register Name	Access Type	Description
0	STATUS	Read Only	Read from this register to get the Virtual Socket Manager's status.
0	CONTROL	Write Only	Write to this register to shutdown the Virtual Socket Manager or perform other Virtual Socket Manager commands.
1	SW_TRIGGER	Read/Write	Write to this register to send a trigger to the Virtual Socket Manager. Read the register to determine if there is a software trigger pending.

STATUS Register

The STATUS register is read only and is mapped to the same address as the CONTROL register.

Table 2-4: STATUS Register Format

Bits	Meaning	Details
31:24	BS_ID/RESERVED	These bits are reserved when the device to be managed is a 7 Series or UltraScale+ device. When the device to be managed is an UltraScale device, these bits contain the identifier of the bitstream to which the status applies.
23:8	RM_ID	The identifier of the Reconfigurable Module to which the status applies.
7	SHUTDOWN	<ul style="list-style-type: none"> 1 = The Virtual Socket Manager is in the shutdown state. 0 = The Virtual Socket Manager is not in the shutdown state.

Table 2-4: STATUS Register Format (Cont'd)

Bits	Meaning	Details
6:3	ERROR	<p>The following error codes are defined:</p> <ul style="list-style-type: none"> • 1111 = An unknown error occurred. • 1000 = (BAD FORMAT ERROR) A compressed bitstream was received in the incorrect format • 0111 = (BAD SIZE ERROR) A compressed bitstream ended at an invalid place in the decompression algorithm • 0110 = (LOST + FETCH errors) Access to the ICAPE3 was removed during a bitstream transfer, and there was an error fetching the bitstream from the configuration library. This error is only possible when the device to be managed is an UltraScale or UltraScale+ device. • 0101 = (BS + FETCH errors) The ICAP returned an error code while loading the bitstream and there was an error fetching the bitstream from the configuration library. • 0100 = (FETCH ERROR) There was an error fetching the bitstream from the configuration library. • 0011 = (LOST ERROR) Access to the ICAPE3 was removed during a bitstream transfer. This error is only possible when the device to be managed is an UltraScale or UltraScale+ device. • 0010 = (BS ERROR) The ICAP returned an error code while loading the bitstream. • 0001 = (BAD CONFIG ERROR) The fetch path was asked to load a 0 byte bitstream. • 0000 = No Error. <p>All other values are RESERVED.</p>
2:0	STATE	<p>The following states are defined when the Virtual Socket Manager is in the active state:</p> <ul style="list-style-type: none"> • 111 = The Virtual Socket is full. That is, a Reconfigurable Module has been successfully loaded. • 110 = The Virtual Socket Manager is executing the Reconfigurable Module reset step. • 101 = The Virtual Socket Manager is executing the software start-up step. • 100 = The Virtual Socket Manager is loading the new Reconfigurable Module. • 011 = The Virtual Socket Manager is loading the Clearing Bitstream for the currently loaded Reconfigurable Module. (Not used if the device to be managed is a 7 Series or UltraScale+ device.) • 010 = The Virtual Socket Manager is executing the software shutdown step. • 001 = The Virtual Socket Manager is executing the hardware shutdown step. • 000 = The Virtual Socket is empty. That is, there is no Reconfigurable Module loaded. <p>The following states are defined when the Virtual Socket Manager is in the shutdown state:</p> <ul style="list-style-type: none"> • 001 = RM_SHUTDOWN_ACK is 1. • 000 = RM_SHUTDOWN_ACK is 0.

CONTROL Register

The CONTROL register is write only, and is mapped to the same address as the STATUS register.

Table 2-5: CONTROL Register Format

Bits	Meaning	Details
31:16	HALFWORD FIELD	A 16-bit field containing extra information for the selected command. See the command descriptions for more information.
15:8	BYTE FIELD	An 8-bit field containing extra information for the selected command. See the command descriptions for more information.
7:0	CMD	<p>The following commands are defined:</p> <ul style="list-style-type: none"> • 000 = Shutdown • 001 = Restart with no Status • 010 = Restart with Status • 011 = Proceed • 100 = User Control <p>All other values are reserved.</p> <p>These commands are described in Virtual Socket Manager Control Interface.</p>

SW_TRIGGER Register

The SW_TRIGGER register can be written and read.

Table 2-6: Software Trigger Register Format

Bits	Meaning	Details
31	Trigger Pending	<p>Ignored on write.</p> <p>On read, returns:</p> <ul style="list-style-type: none"> • 1 if there is a software trigger pending. • 0 if there is no software trigger pending.
30:W	Reserved	<p>Ignored on write.</p> <p>Returns 0 on read.</p>
W-1:0	Trigger	<p>The Trigger Identifier. The value written to this register is a positive integer that directly specifies the row in the Trigger to Reconfigurable Module register bank that holds the identifier of the Reconfigurable Module to be loaded by this trigger. Writing this while a trigger is pending overwrites the pending trigger.</p> <p>The width of this field (W) is:</p> $\left\lceil \log_2 \left(\text{Number of Triggers Allocated for this Virtual Socket Manager} \right) \right\rceil$

Bank 1: Trigger to Reconfigurable Module Registers

The Trigger to Reconfigurable Module registers are defined in [Table 2-7](#).

Table 2-7: Bank 1 - Trigger to Reconfigurable Module Registers

Register Select Value	Register Name	Access Type	Description
0	TRIGGER0	Write/Read	This register contains the identifier of the Reconfigurable Module (RM_ID) that will be loaded if trigger 0 is asserted.
1	TRIGGER1	Write/Read	This register contains the identifier of the Reconfigurable Module (RM_ID) that will be loaded if trigger 1 is asserted.
2	TRIGGER1	Write/Read	This register contains the identifier of the Reconfigurable Module (RM_ID) that will be loaded if trigger 2 is asserted.
⋮	⋮	⋮	⋮
<i>N</i>	TRIGGER <i>N</i>	Write/Read	This register contains the identifier of the Reconfigurable Module (RM_ID) that will be loaded if trigger <i>N</i> is asserted.

The Trigger to Reconfigurable Module registers contain the mapping between the Triggers and the Reconfigurable Modules to load. There can be more triggers than Reconfigurable Modules, allowing for the in-field addition of Reconfigurable Modules and/or easier triggering of the same Reconfigurable Module from multiple sources.

Each register is 32-bits wide, but only the lower *X* bits are used, where $X = \lceil \log_2(\text{Number of Triggers Allocated for this Virtual Socket Manager}) \rceil$

Unused bits are ignored on writes, and return 0 on reads. The Trigger to Reconfigurable Module registers can only be accessed when the Virtual Socket Manager is in the shutdown state. If the Virtual Socket Manager is not in the shutdown state, reads return 0 and writes are ignored.

Bank 2: Reconfigurable Module Information Registers

The Reconfigurable Module Information registers are defined in [Table 2-8](#).

There are two registers per row in this bank, and the LSB of *Register Select* is used to select between them.

- **BS Index Register:** LSB = 0
- **Control Register:** LSB = 1

All registers in this bank are readable and writable when the Virtual Socket Manager is in the shutdown state. If the Virtual Socket Manager is not in the shutdown state, reads return 0, and writes are ignored.

Table 2-8: Bank 2 - Reconfigurable Module Information Registers

Register Select MSBs	Register Select LSB	Register Name	Description
0	0	RM_BS_INDEX0	This register contains the row number in the Bitstream Information register bank that holds information about the bitstream for Reconfigurable Module 0.
0	1	RM_CONTROL0	This register contains the control information for Reconfigurable Module 0.
1	0	RM_BS_INDEX1	This register contains the row number in the Bitstream Information register bank that holds information about the bitstream for Reconfigurable Module 1.
1	1	RM_CONTROL1	This register contains the control information for Reconfigurable Module 1.
2	0	RM_BS_INDEX2	This register contains the row number in the Bitstream Information register bank that holds information about the bitstream for Reconfigurable Module 2.
2	1	RM_CONTROL2	This register contains the control information for Reconfigurable Module 2.
⋮		⋮	⋮
<i>N</i>	0	RM_BS_INDEX <i>N</i>	This register contains the row number in the Bitstream Information register bank that holds information about the bitstream for Reconfigurable Module <i>N</i> .
<i>N</i>	1	RM_CONTROL <i>N</i>	This register contains the control information for Reconfigurable Module <i>N</i> .

RM_BS_INDEX Register

Table 2-9: RM_BS_INDEX Register Format

Bits	Meaning	Details
31:16	CLEAR_BS_INDEX/ Reserved	Reserved when the device to be managed is 7 Series or UltraScale+. When the device to be managed is an UltraScale device, these bits contain an address into the Bitstream Information register bank, and they link a Reconfigurable Module to its clearing bitstream.
15:0	BS_INDEX	These bits contain an address into the Bitstream Information register bank, and they link a Reconfigurable Module to the bitstream required to load it.

RM_CONTROL Register

Table 2-10: RM_CONTROL Register Format

Bits	Meaning	Details
31:13	Reserved	Reads return 0, writes are ignored
12:5	Reset Duration	<p>The number of clock cycles -1 to assert the Reconfigurable Module reset for</p> <ul style="list-style-type: none"> • 0: 1 clock cycle • 1: 2 clock cycles • etc. <p>The maximum reset duration is 256 clock cycles.</p>
4:3	Reset Required	<ul style="list-style-type: none"> • 00: No Reconfigurable Module Reset is required. • 01: RESERVED. • 10: Active-Low Reconfigurable Module reset is required. • 11: Active-High Reconfigurable Module reset is required.
2	Startup Required	<ul style="list-style-type: none"> • 0: No start-up is required. • 1: Software start-up is required.
1:0	Shutdown Required	<ul style="list-style-type: none"> • 00: No Reconfigurable Module shutdown is required. • 01: Hardware Reconfigurable Module shutdown is required. • 10: Hardware then software shutdown is required. • 11: Software then hardware shutdown is required.

Bank 3: Bitstream Information Registers

The Bitstream Information registers are defined in [Table 2-11](#).

There are three registers per row in this bank, and the two LSBs of *Register Select* are used to select between them.

- **ID Register:** LSBs = 0
- **Address Register:** LSBs = 1
- **Size Register:** LSBs = 2

When the device to be managed is an UltraScale device, all registers in this bank are readable and writable when the Virtual Socket Manager is in the shutdown state. Reconfigurable Modules for this type of device require two bitstreams, so all bitstream identifiers are 0 or 1.

When the device to be managed is a 7 series or UltraScale+ device, the BS_ADDRESS and BS_SIZE registers in this bank are readable and writable when the Virtual Socket Manager is in the shutdown state. Reconfigurable Modules for this type of device only require one bitstream, so all bitstream identifiers are 0. Writes to the BS_ID registers are ignored and reads always return 0.

For all managed device types, reads from all registers in this bank return 0 and writes to all registers in this bank are ignored if the Virtual Socket Manager is not in the shutdown state.

Table 2-11: Bank 3 - Bitstream Information Registers

Register Select MSBs	Register Select LSB	Register Name	Description
0	0	BS_ID0	This register contains the identifier of this bitstream.
0	1	BS_ADDRESS0	This register contains the byte address of Bitstream 0 in the configuration library interface.
0	2	BS_SIZE0	This register contains the size in bytes of Bitstream 0.
1	0	BS_ID1	This register contains the identifier of this bitstream.
1	1	BS_ADDRESS1	This register contains the byte address of Bitstream 1 in the configuration library interface.
1	2	BS_SIZE1	This register contains the size in bytes of Bitstream 1.
2	0	BS_ID2	This register contains the identifier of this bitstream.
2	1	BS_ADDRESS2	This register contains the byte address of Bitstream 2 in the configuration library interface.
2	2	BS_SIZE2	This register contains the size in bytes of Bitstream 2.
⋮		⋮	⋮
<i>N</i>	0	BS_ID <i>N</i>	This register contains the identifier of this bitstream.
<i>N</i>	1	BS_ADDRESS <i>N</i>	This register contains the byte address of Bitstream <i>N</i> in the configuration library interface.
<i>N</i>	2	BS_SIZE <i>N</i>	This register contains the size in bytes of Bitstream <i>N</i> .

BS_ID Register

Table 2-12: BS_ID Register Format

Bits	Meaning	Details
31:1	Reserved	Reserved
0	ID	The identifier of the bitstream with regards to the Reconfigurable Module. The first bitstream for a Reconfigurable Module has the ID 0, and the second has the ID 1.

BS_ADDRESS Register

Table 2-13: BS_ADDRESS Register Format

Bits	Meaning	Details
31:0	ADDRESS	The address of the bitstream in the configuration library interface.

BS_SIZE Register**Table 2-14: BS_SIZE Register format**

Bits	Meaning	Details
31:0	Size	The bitstream size in bytes

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

Preparing for In-Field Upgrades

Some designs use Partial Reconfiguration to allow in-field updates of parts of the design. Specifically, Reconfigurable Modules can be added, removed, or changed after the design is in the field. The Partial Reconfiguration Controller supports this by allowing the trigger, Reconfigurable Module and bitstream information to be reprogrammed using the AXI4-Lite interface. However, new Reconfigurable Modules can only be added if there is space for them and a trigger is available to trigger their activation.

If in-field upgrades are planned for a design, the extra space for these items should be allocated at core generation time.

Clocking

There are two clock ports: `icap_clk` for the logic driving the ICAP interface, and `clk` for everything else.

The number of synchronization stages used when passing signals between domains is configurable.

Resets

There are two reset ports: `icap_reset` for the logic driving the ICAP interface, and `reset` for everything else. The exact timing of these resets relative to each other is not important. However, if one is asserted, both must be asserted. For example, resetting the ICAP interface without resetting the rest of the Partial Reconfiguration Controller could result in

lockup, and vice versa. Additionally, no triggers should be generated before both resets have been asserted and released.



RECOMMENDED: *It is highly recommended that all Virtual Socket Managers are placed into the shutdown state (and checked to see that they are in shutdown) before applying a reset.*

If this is not possible, then the following conditions must be met before the core is reset:

1. The ICAP interface must not be transferring a bitstream to the ICAP. Interrupting a bitstream transfer could put the device into an unstable state and could require a full chip reconfiguration.
2. Virtual Socket Managers that are in the active state must have no pending triggers. Resetting a Virtual Socket Manager that has started to process a trigger can corrupt the state of the Virtual Socket Manager. For Virtual Socket Managers that control UltraScale devices, this could make it impossible to load a new Reconfigurable Module in that Virtual Socket.

Note: The active reset level is configurable.

Virtual Socket Manager Control Interface

Each Virtual Socket Manager can be controlled independently using the optional AXI4-Stream control channel, or the AXI4-Lite register interface using the CONTROL register. The command word format is identical no matter which interface is used.

Shutdown Command

The **Shutdown** command instructs the Virtual Socket Manager to enter the shutdown state at the earliest safe opportunity. There can be a long delay (indeterminate) between the request and when the Virtual Socket Manager enters the shutdown state. You cannot cancel the **Shutdown** command after it has been sent.

This command can only be used if the Virtual Socket Manager is not in the shutdown state.

The BYTE and HALFWORD fields of the control word are not used with this command.

Restart with no Status Command

The **Restart with no Status** command instructs the Virtual Socket Manager to exit the shutdown state. The Virtual Socket Manager's Empty/Full status, Reconfigurable Module identifier, and error status remain as they were before the Virtual Socket Manager entered the shutdown state.

This command should be used to restart a Virtual Socket Manager in shutdown if the Virtual Socket has not been modified during shutdown.

This command can only be used if the Virtual Socket Manager is in the shutdown state.

The BYTE and HALFWORD fields of the control word are not used with this command.

Restart with Status Command

The **Restart with Status** command instructs the Virtual Socket Manager to exit the shutdown state. The Virtual Socket Manager's Empty/Full status, and Reconfigurable Module identifier are specified as part of the command.

This command should be used to restart a shutdown Virtual Socket Manager if the Virtual Socket has been modified during shutdown (that is, a Reconfigurable Module is loaded into the Virtual Socket by something other than the Virtual Socket Manager).

This command must only be used with a full status if the loaded Reconfigurable Module is known to the Virtual Socket Manager. If a Reconfigurable Module is loaded that is unknown to the Virtual Socket Manager, the Virtual Socket must be left in an empty state before the Virtual Socket Manager is restarted. An *empty* state means that either there is no Reconfigurable Module in the Virtual Socket, or that the loaded Reconfigurable Module does not need any shutdown steps. When the Virtual Socket is on an UltraScale device, there is an additional requirement that the loaded Reconfigurable Module is unmasked and does not need its clearing bitstream loaded.

This command can only be used if the Virtual Socket Manager is in the shutdown state.

The BYTE and HALFWORD fields of the control word are used as follows:

- BYTE[0]: The Empty/Full status value to set on restart.
 - 0: The Virtual Socket is empty.
 - 1: The Virtual Socket is full.
- HALFWORD: The identifier of the Reconfigurable Module that is loaded while the Virtual Socket Manager is in a shutdown state. This must be a Reconfigurable Module that is known to the Virtual Socket Manager.

Proceed Command

The **Proceed** command instructs the Virtual Socket Manager to proceed with processing the Reconfigurable Module. If the Virtual Socket Manager has asserted the `vsm_<name>_sw_shutdown_req` or `vsm_<name>_sw_startup_req` interrupt signals, it will stall until this command occurs. This command is ignored if neither of these signals is asserted.

This command can only be used if the Virtual Socket Manager is not in the shutdown state.

The BYTE and HALFWORD fields of the control word are not used with this command.

User Control Command

The **User Control** command is used to set the values of the following signals:

- `vsm_<name>_rm_shutdown_req`
- `vsm_<name>_rm_decouple`
- `vsm_<name>_rm_reset`
- `vsm_<name>_sw_shutdown_req`
- `vsm_<name>_sw_startup_req`

When set, the values remain set until another **User Control** command is sent, or one of the **Restart** commands are sent.

This command can only be used if the Virtual Socket Manager is in the shutdown state.

The BYTE and HALFWORD fields of the control word are used as follows:

- BYTE[0]: Controls `rm_shutdown_req`.
 - 0: Sets `vsm_<name>_rm_shutdown_req` to 0.
 - 1: Sets `vsm_<name>_rm_shutdown_req` to 1.
- BYTE[1]: Controls `rm_decouple`.
 - 0: Sets `vsm_<name>_rm_decouple` to 0.
 - 1: Sets `vsm_<name>_rm_decouple` to 1.
- BYTE[2]: Controls `sw_shutdown_req`.
 - 0: Sets `vsm_<name>_sw_shutdown_req` to 0.
 - 1: Sets `vsm_<name>_sw_shutdown_req` to 1.
- BYTE[3]: Controls `sw_startup_req`.
 - 0: Sets `vsm_<name>_sw_startup_req` to 0.
 - 1: Sets `vsm_<name>_sw_startup_req` to 1.
- BYTE[4]: Controls `rm_reset`.
 - 0: Sets `vsm_<name>_rm_reset` to 0.
 - 1: Sets `vsm_<name>_rm_reset` to 1.
- HALFWORD: Not used.

Protocol Description

Hardware Triggers

If a Virtual Socket Manager is configured to have hardware triggers, it has a vector input signal, `vsm_<name>_hw_triggers`. Each bit in this signal is processed independently, and maps directly to a trigger identifier of the same number. For example:

- `vsm_<name>_hw_triggers[0]` maps to trigger 0
- `vsm_<name>_hw_triggers[1]` maps to trigger 1
- etc.

Trigger *N* occurs when `vsm_<name>_hw_triggers[N]` synchronously transitions from 0 to 1. Triggers cannot be canceled after they occur. After one clock cycle, `vsm_<name>_hw_triggers[N]` can return to 0 and trigger *N* will still be pending in the Virtual Socket Manager.

When trigger *N* occurs, the Virtual Socket Manager records that fact and the trigger becomes available for processing. The trigger is ignored if it occurs again before the previous activation is selected for processing. Only one instance of a trigger activation is stored.

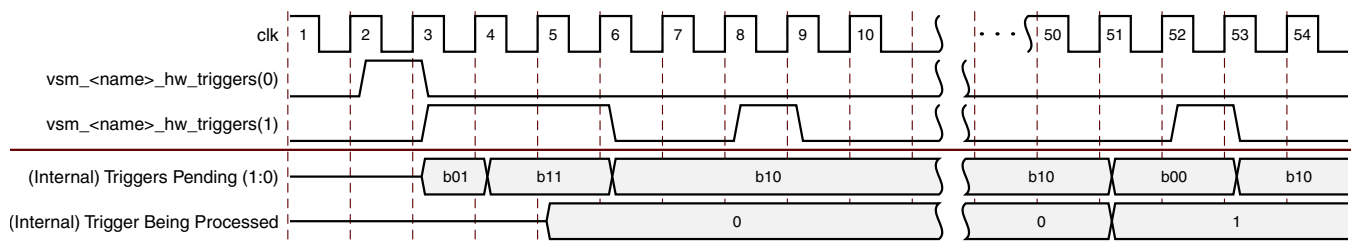


Figure 3-1: Hardware Trigger Example

Figure 3-1 shows an example of the hardware triggers in operation. Note that the clock cycle numbers are just for illustration and do not relate to any fixed latency.

- At clock 3, a transition from 0 to 1 on `vsm_<name>_hw_triggers[0]` is seen and this is recorded internally on `Triggers Pending[0]`.
- At clock 4, a transition from 0 to 1 on `vsm_<name>_hw_triggers[1]` is seen and this is recorded internally on `Triggers Pending[1]`.
- At clock 6, the Virtual Socket Manager starts processing trigger 0. `Triggers Pending[0]` changes to 0 which means a new trigger 0 could be captured.

- At clock 9, a transition from 0 to 1 on `vsm_<name>_hw_triggers[1]` is seen. However, `Triggers Pending[1]` is already 1, so the new trigger 1 is ignored.
- At clock 51, the Virtual Socket Manager stops processing trigger 0 and starts on trigger 1 which has been pending since clock cycle 5. `Triggers Pending[1]` changes to 0 which means a new trigger 1 could be captured.
- This trigger arrives at clock cycle 53, where `vsm_<name>_hw_triggers[1]` transitions from 0 to 1. Because trigger 1 is not pending, it is captured again.

At this stage, trigger 1 is both pending and being processed. When the Reconfigurable Module mapped to trigger 1 loads, the Virtual Socket Manager immediately starts to remove it and process the pending trigger, which will load the same Reconfigurable Module. Loading an already loaded Reconfigurable Module might be useful in certain circumstances so the Partial Reconfiguration Controller allows it. Proper trigger management can avoid this situation if it is not desired.

Reconfigurable Module Hardware and Software Shutdown

Some Reconfigurable Modules might need to be shut down before they are removed from the device. This would be required if arbitrarily removing a Reconfigurable Module could cause deadlock or other system issues. The exact nature of the shutdown required is specific to each Reconfigurable Module. The Partial Reconfiguration Controller provides a handshaking protocol to manage the process, but the actual shutdown mechanism must be provided by the system designer.

Each Virtual Socket Manager has an active-high output signal called `vsm_<name>_rm_shutdown`. This signal is provided to control user supplied shutdown logic, such as the PR AXI Shutdown IP core [Ref 17], and is asserted under the following conditions:

1. When the Virtual Socket is empty.
2. When a full Virtual Socket Manager is reset, and `shutdown_required` is set to HW, HW/SW or SW/HW.
3. When the Virtual Socket Manager starts to load a new Reconfigurable Module and the existing Reconfigurable Module needs to be shut down before it can be removed. Once asserted, `vsm_<name>_rm_shutdown_ack` will remain asserted until the new Software Startup and RM Reset steps are complete. These steps are optional. If they are disabled, `vsm_<name>_rm_shutdown_req` is deasserted at the points where they would have completed had they been enabled.

It is deasserted at all other times. `vsm_<name>_rm_shutdown_req` may change value when the PRC is reset. The following rules apply:

1. If the Virtual Socket was empty before reset, `vsm_<name>_rm_shutdown_req` will assert during reset. See Figure 3-8.

Note: It would normally be asserted before the reset, resulting in no visible change. However, its value may have been changed by the user using the register interface.

2. If the Virtual Socket was full before reset, and the Reconfigurable Module was configured to have `shutdown_required = No` then `vsm_<name>_rm_shutdown_req` will be deasserted. See Figure 3-10.
3. If the Virtual Socket was full before reset, and the Reconfigurable Module was configured to have `shutdown_required = HW, HW/SW or SW/HW`, then `vsm_<name>_rm_shutdown_req` will be asserted. See Figure 3-9.

When `vsm_<name>_rm_shutdown_ack` becomes asserted in case 3, the Virtual Socket Manager will stall until the user logic responds.

Figure 3-2 shows the protocol. `vsm_<name>_rm_shutdown_req` is set to 1 by the Partial Reconfiguration Controller when the Reconfigurable Module is scheduled for removal. The Reconfigurable Module should put itself into a safe state as soon as possible. There is no time limit for this to occur. When the Reconfigurable Module is ready to be removed, it must set `vsm_<name>_rm_shutdown_ack` to 1.

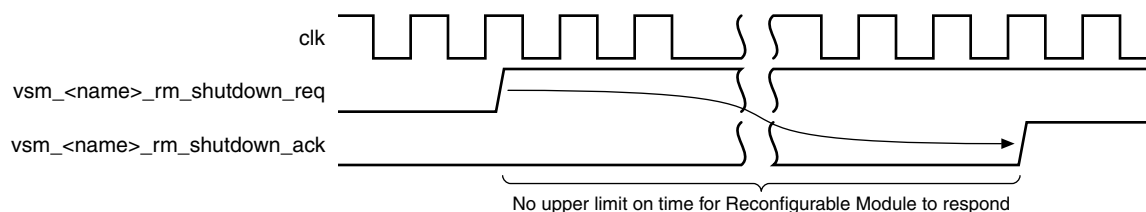


Figure 3-2: Hardware Shutdown of a Reconfiguration Module

The following waveforms show the behavior of `vsm_<name>_rm_shutdown_req`.

To reduce the size of the waveforms:

- `vsm_<name>_` has been omitted from signal names.
- The number of clock cycles spent in each state has been kept small and made constant. In reality, each state can last an indeterminate amount of time.
- `Command` represents a command sent to the VSM using the AXI4-Lite interface or the VSM's AXI4-Stream Control channel.
- The `Load RM` state includes loading the Clearing Bitstream if the device being controlled is an UltraScale device.

Figure 3-3 shows the behavior of `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load when the existing Reconfigurable Module is configured as `SHUTDOWN_REQUIRED = no`.

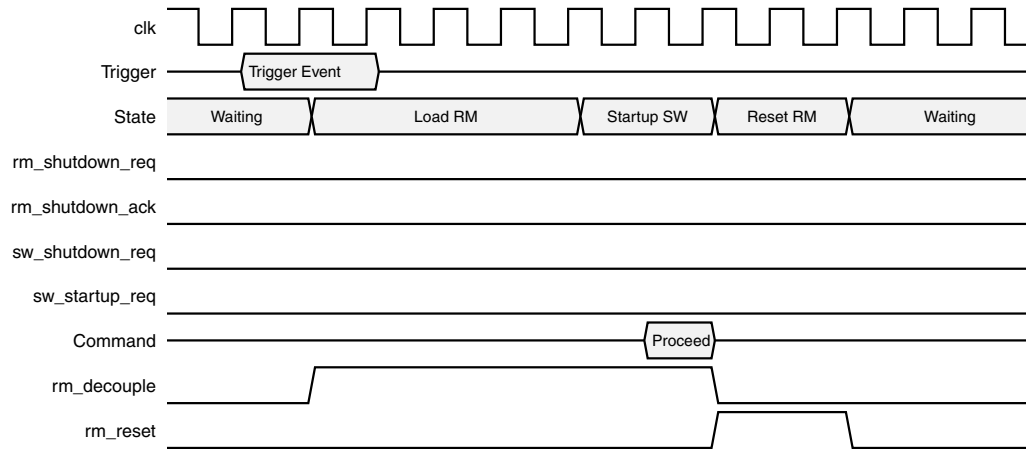


Figure 3-3: `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load operation when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = no`

Figure 3-4 shows the behavior of `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = hw`.

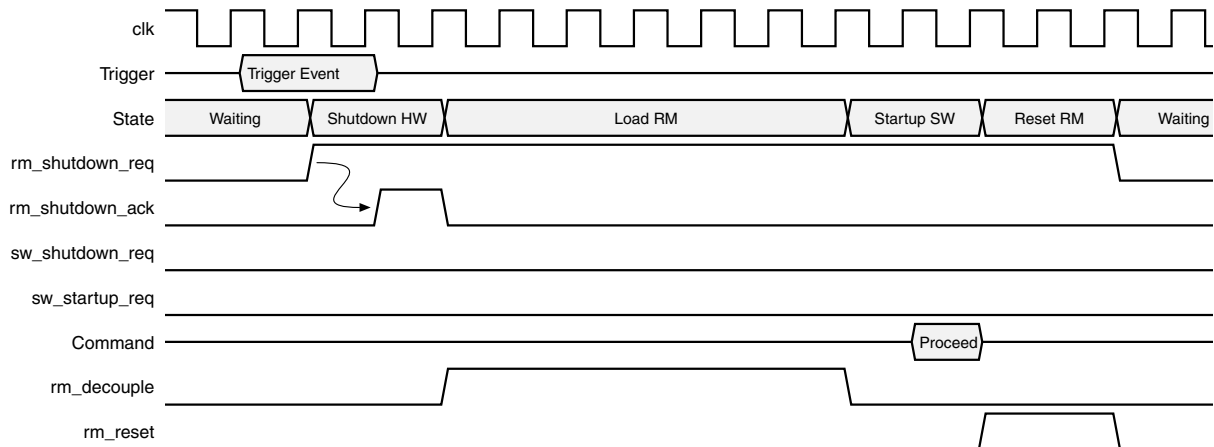


Figure 3-4: `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load operation when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = hw`

Figure 3-5 shows the behavior of `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = hw/sw`.

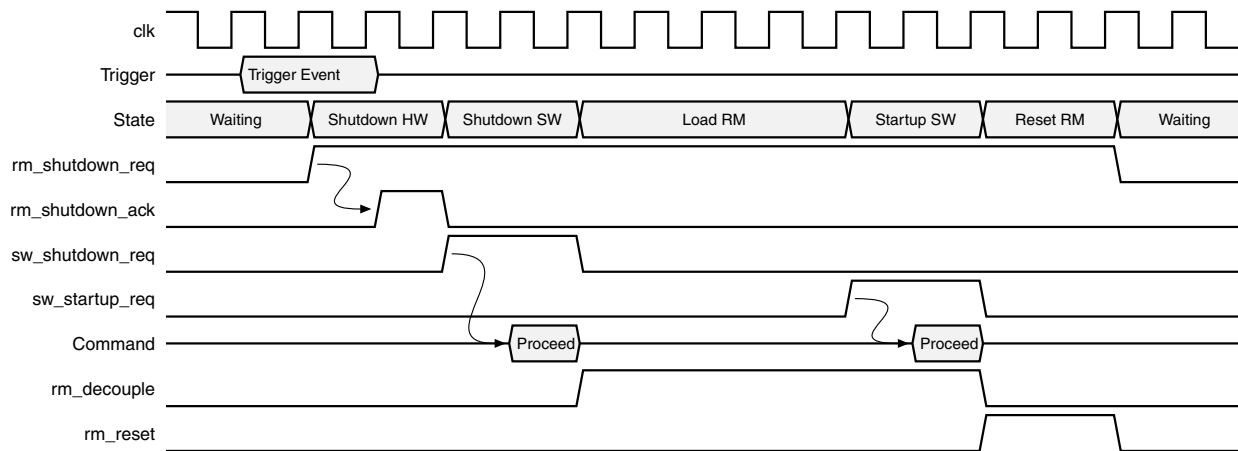


Figure 3-5: `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load operation when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = hw/sw`

Figure 3-6 shows the behavior of `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = sw/hw`.

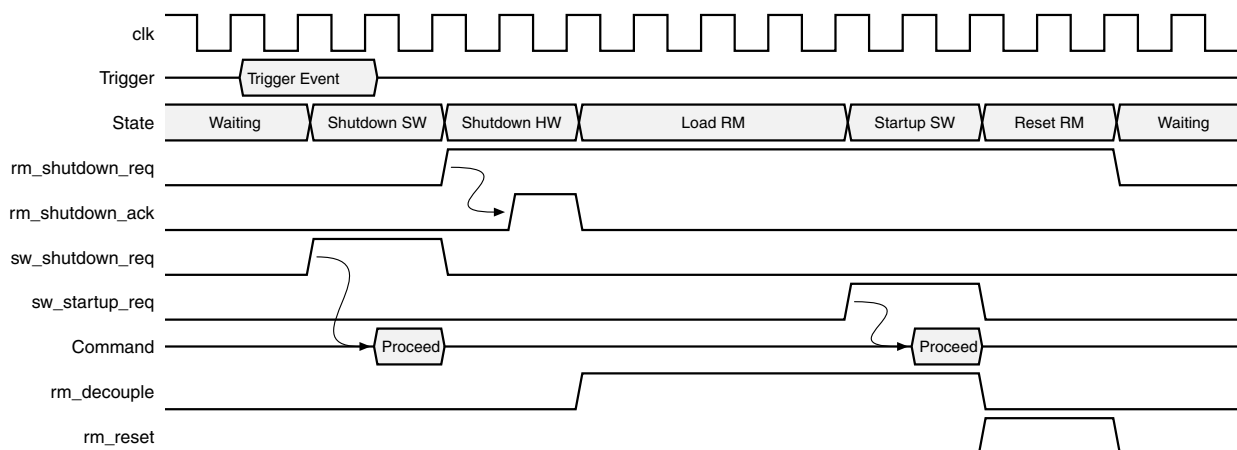


Figure 3-6: `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load operation when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = sw/hw`

Note: `vsm_<name>_rm_shutdown_req` stays asserted while software for the new Reconfigurable Module is being set up, and while the new Reconfigurable Module is being reset. This is to prevent the system trying to interact with the new Reconfigurable Module until it is ready to operate.

Figure 3-7 shows the behavior of `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load when all optional startup steps are disabled.

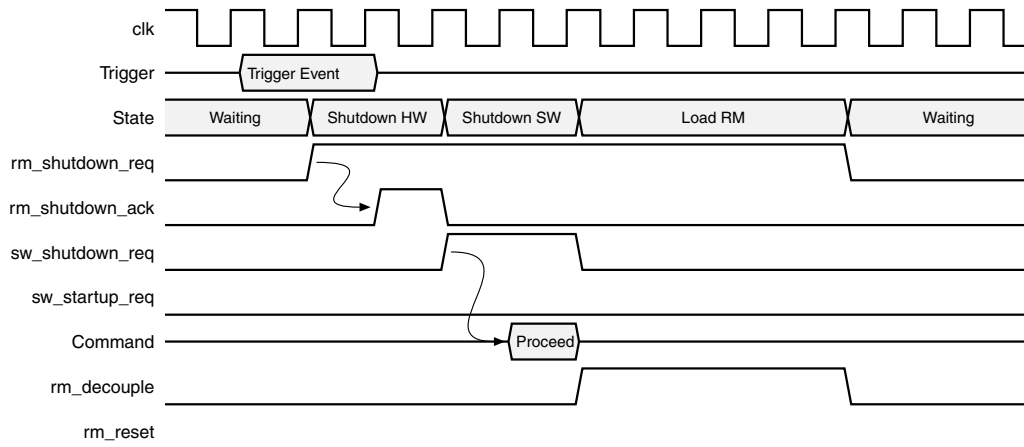


Figure 3-7: `vsm_<name>_rm_shutdown_req` during a Reconfigurable Module load operation when all optional startup steps are disabled

Figure 3-8, Figure 3-9 and Figure 3-10 show how `vsm_<name>_rm_shutdown_req` responds to a reset.

Figure 3-8 shows an empty Virtual Socket Manager during a core reset.

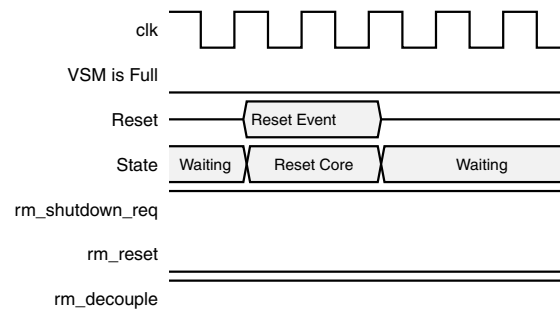


Figure 3-8: `vsm_<name>_rm_shutdown_req` during a core reset while the Virtual Socket Manager is empty

Figure 3-9 shows a full Virtual Socket Manager during a core reset when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = HW, SW/HW or HW/SW`.

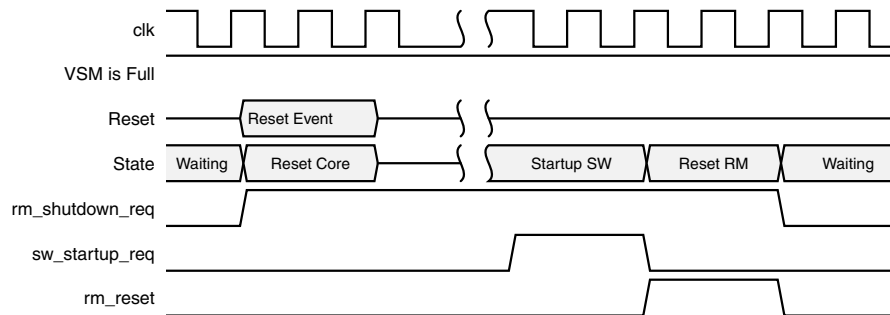


Figure 3-9: `vsm_<name>_rm_shutdown_req` during a core reset while the Virtual Socket Manager is full. The existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = HW, SW/HW or HW/SW` (Optional Reconfigurable Module startup steps are enabled).

Figure 3-10 shows a full Virtual Socket Manager during a core reset when the existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = No`.

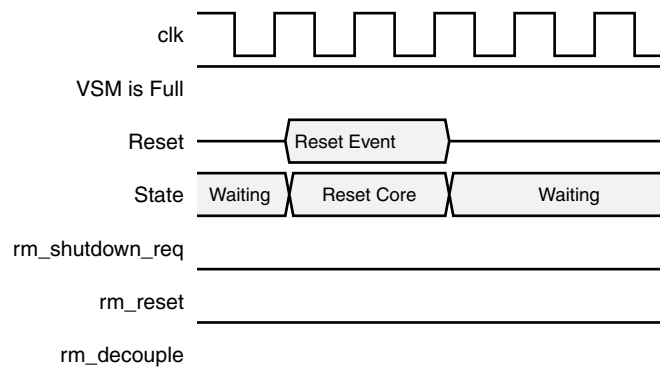


Figure 3-10: `vsm_<name>_rm_shutdown_req` during a core reset while the Virtual Socket Manager is full. The existing Reconfigurable Module is configured to have `SHUTDOWN_REQUIRED = No` (Optional Reconfigurable Module startup steps are disabled.)

If a Virtual Socket Manager is put into the shutdown state, `vsm_<name>_rm_shutdown_req` maintains its value. However, it becomes controllable through the control register at this point and can be changed using the control interface. This means it could be deasserted even if the Virtual Socket Manager is empty, or asserted even if the Virtual Socket Manager is full. When the Virtual Socket Manager re-enters the active state, `vsm_<name>_rm_shutdown_req` will assert if the Virtual Socket Manager is empty, and deassert if the Virtual Socket Manager is full, regardless of what it was set to by the control interface during shutdown.

Additionally, software might also need to shut down before a Reconfigurable Module is removed. For example, a device driver might need to be unloaded. The Partial Reconfiguration Controller provides the `vsm_<name>_sw_shutdown_req` signal to aid this

process. This can be attached to an interrupt controller, or any other scheme the design has for communicating with software. [Figure 3-11](#) shows the protocol. When the Reconfigurable Module is being removed, `vsm_<name>_sw_shutdown_req` is set to 1. The software should put itself into a safe state as soon as possible. There is no time limit for this to occur. When the software is ready for the Reconfigurable Module to be removed, it must send the **Proceed** command to the Virtual Socket Manager.

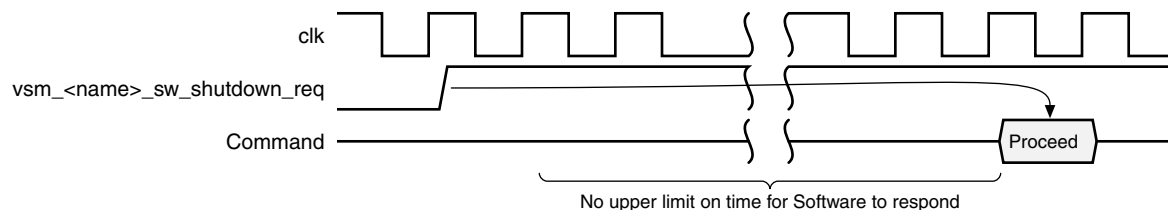


Figure 3-11: Software Shutdown of a Reconfiguration Module

The shutdown behavior is configurable on a per Reconfigurable Module basis.

Reconfigurable Module Software Startup

After a Reconfigurable Module is loaded, but before it is reset (optional) and before decoupling is released, software might need to load device drivers. The Partial Reconfiguration Controller provides the `vsm_<name>_sw_startup_req` signal to aid this process. This signal can be attached to an interrupt controller, or any other scheme the design has for communicating with software. [Figure 3-12](#) shows the protocol. When the Reconfigurable Module is loaded, `vsm_<name>_sw_startup_req` is set to 1. The software should do what is needed to be ready for the Reconfigurable Module becoming operational. There is no time limit on this. When the software is ready for the Reconfigurable Module to be started, it must send the **Proceed** command to the Virtual Socket Manager CONTROL register using the AXI4-Lite interface. For more information, see [Proceed Command](#).

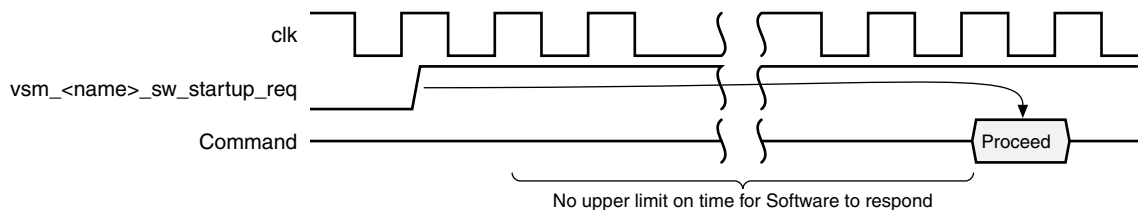


Figure 3-12: Software Startup of a Reconfiguration Module

The start-up behavior is configurable on a per Reconfigurable Module basis.

ICAP Sharing Protocol

The Partial Reconfiguration Controller uses a simple protocol to arbitrate for access to the ICAP port. This can be controlled using the `C_ARBITRATION_PROTOCOL` user parameter. The following values can be set:

- 0: This turns off arbitration and removes the arbitration protocol ports from the core's boundary.
- 1: This turns arbitration on for the case where there is no external latency added to the arbitration signals.
- 2: This turns arbitration on for the case where there is external latency added to the arbitration signals.

Protocol versions 1 and 2 are similar so are described together with the minor differences explained at the end of this section.

The IP sets `cap_req` to 1 on every clock cycle where it has data to transfer to the ICAP. Data transfer occurs only when the arbiter sets `cap_gnt` to 1 as well. The arbiter must keep `cap_gnt` as 1 until `cap_req` becomes 0.

The arbiter can set `cap_rel` to 1 to signal that something else requires access to the ICAP. If the Partial Reconfiguration Controller sees `cap_rel` equal to 1, it completes any bitstream load that is in progress and then sets `cap_req` to 0. `cap_rel` should be asserted by the arbiter on every clock cycle where something else is requesting access to the ICAP. When set to 1, this signal should remain at 1 until `cap_req` returns to 0.

Protocols 1 and 2 differ at the end of a transfer. In the case where there is no latency added to the arbitration signals (protocol 1), the arbiter can keep `cap_gnt` asserted and the Partial Reconfiguration Controller can immediately request a new transfer.

In the case where there is latency added to the arbitration signals (protocol 2), the arbiter must set `cap_gnt` to 0. When this occurs, the Partial Reconfiguration Controller will set `cap_req` to 0.

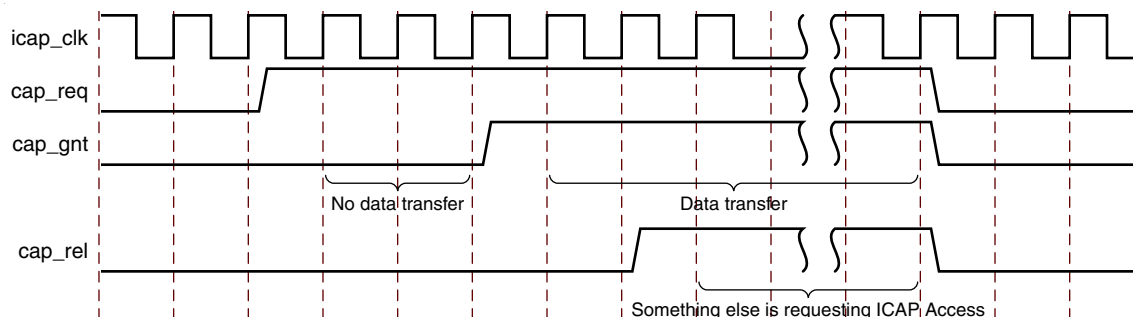


Figure 3-13: ICAP Arbitration Protocol

Decoupling

Each Virtual Socket Manager has an active-high output signal called `vsm_<name>_rm_decouple`. This signal is provided to control user supplied decoupling logic. It is asserted high under the following conditions:

1. When the Virtual Socket is empty.
2. When the Virtual socket Manager is loading a new Reconfigurable Module. This starts once the Hardware and Software shutdown steps are complete and continues until the Software Startup step is complete. These steps are optional. If they are disabled, `vsm_<name>_rm_decouple` is asserted and deasserted at the points where they would have completed had they been enabled.

It is deasserted at all other times. `vsm_<name>_rm_decouple` may change value when the PRC is reset. The following rules apply:

1. If the Virtual Socket was empty before reset, `vsm_<name>_rm_decouple` will assert during reset. See [Figure 3-17](#).
2. If the Virtual Socket was full before reset then `vsm_<name>_rm_shutdown_req` will be deasserted. See [Figure 3-15](#) and [Figure 3-18](#).

Note: `vsm_<name>_rm_decouple` would normally be asserted if the VSM is empty, and deasserted when full, so a reset would normally result in no visible change. However, the value of `vsm_<name>_rm_decouple` may have been changed by the user using the register interface.

The following waveforms show the behavior of `vsm_<name>_rm_decouple`.

To reduce the size of the waveforms:

- **vsm_<name>_** has been omitted from signal names.
- The number of clock cycles spent in each state has been kept small and made constant. In reality, each state can last an indeterminate amount of time.
- **Command** represents a command sent to the VSM using the AXI4-Lite interface or the VSM's AXI4-Stream Control channel.
- The **Load RM** state includes loading the Clearing Bitstream if the device being controlled is an UltraScale device.

Figure 3-14 Shows the behavior of `vsm_<name>_rm_decouple` during a Reconfigurable Module load when all optional shutdown and startup steps are enabled.

Note: `vsm_<name>_rm_decouple` stays asserted while software for the new Reconfigurable Module is being set up. This is to stop the Reconfigurable Module operating before the system is ready.

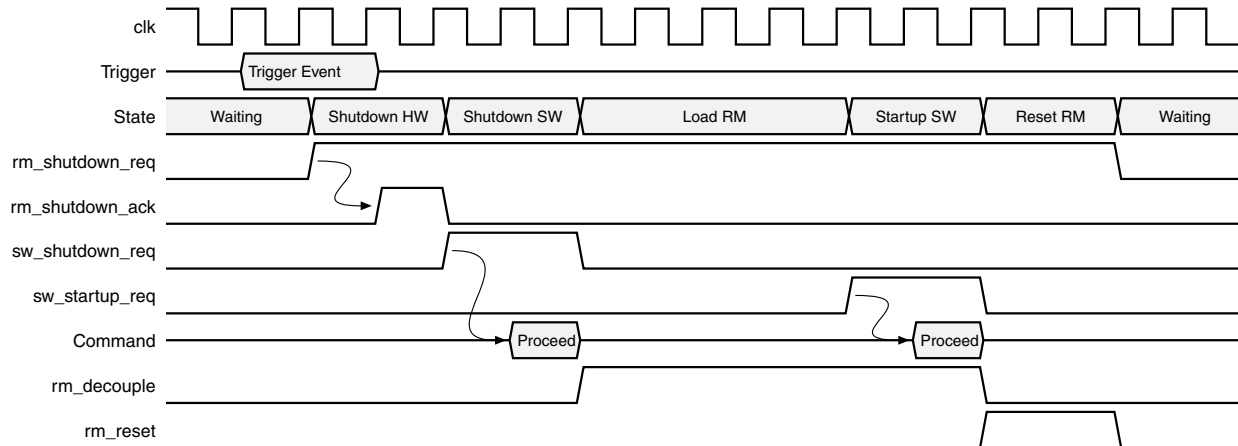


Figure 3-14: `vsm_<name>_rm_decouple` during a Reconfigurable Module load operation (all optional steps enabled)

As shown in **Figure 3-15**, if the startup software step (Startup SW) occurs after a core reset, `vsm_<name>_rm_decouple` will remain deasserted during this step.

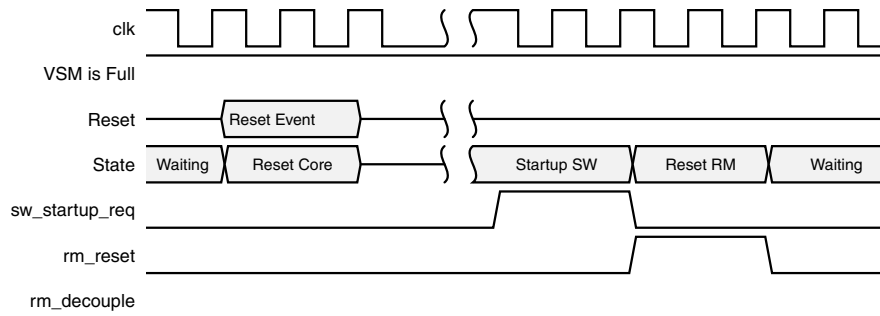


Figure 3-15: `vsm_<name>_rm_decouple` during a core reset while the Virtual Socket Manager is full. Optional Reconfigurable Module startup steps are enabled.

Figure 3-16 shows the behavior of `vsm_<name>_rm_decouple` during a Reconfigurable Module load when all optional shutdown and startup steps are disabled.

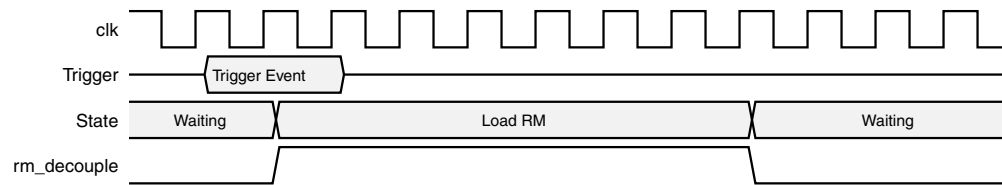


Figure 3-16: **`vsm_<name>_rm_decouple` during a Reconfigurable Module load operation when all optional steps are disabled**

Figure 3-15, Figure 3-17 and Figure 3-18 show how `vsm_<name>_rm_decouple` responds to a reset.

Figure 3-17 shows an empty Virtual Socket Manager (`vsm_<name>_rm_decouple = 1`) during a core reset.

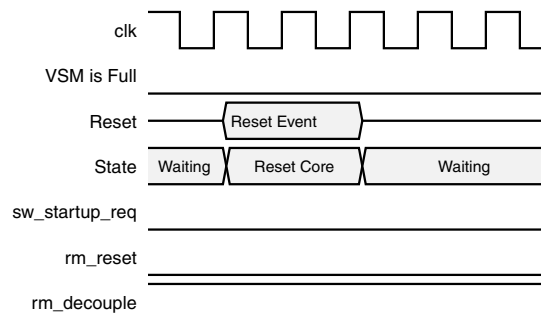


Figure 3-17: **`vsm_<name>_rm_decouple` during a core reset while the Virtual Socket Manager is empty**

Figure 3-18 shows a full Virtual Socket Manager (`vsm_<name>_rm_decouple = 0`) during a core reset.

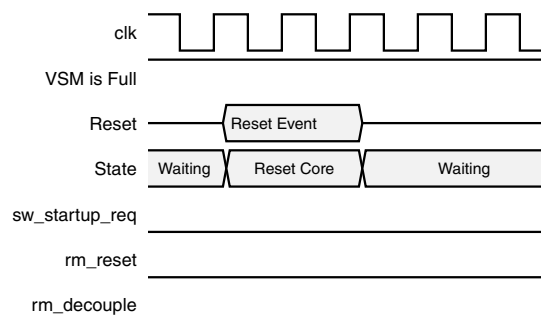


Figure 3-18: **`vsm_<name>_rm_decouple` during a core reset while the Virtual Socket Manager is full. (Optional Reconfigurable Module startup steps are disabled.)**

If a Virtual Socket Manager is put into the shutdown state, `vsm_<name>_rm_decouple` maintains its value. However, it becomes controllable through the control register at this point and can be changed using the control interface. This means it could be deasserted even if the Virtual Socket Manager is empty, or asserted even if the Virtual Socket Manager is full. When the Virtual Socket Manager re-enters the active state, `vsm_<name>_rm_decouple` will assert if the Virtual Socket Manager is empty, and deassert if the Virtual Socket Manager is full, regardless of what it was set to by the control interface during shutdown.

RM_RESET

Each Virtual Socket Manager has an output signal called `vsm_<name>_rm_reset`. This signal is provided to reset Reconfigurable Modules just after they are loaded. Its enablement, active value and duration are configurable for each Reconfigurable Module. It is only asserted under three conditions:

1. Just after a Reconfigurable Module is loaded, if reset is enabled for that Reconfigurable Module.
2. When the core has been reset, the Virtual Socket Manager is full, reset is enabled for that Reconfigurable Module, and Skip Startup After Reset is disabled.
3. When the Virtual Socket Manager is in the shutdown state and `vsm_<name>_rm_reset` is directly manipulated through one of the control interfaces.

If software startup is enabled then this occurs before the Reconfigurable Module is reset.

`vsm_<name>_rm_reset` is deasserted at all other times. It does not change when the Partial Reconfiguration Controller is reset.

The following waveforms show the behavior of `vsm_<name>_rm_reset`.

To reduce the size of the waveforms:

- **vsm_<name>_** has been omitted from signal names.
- The number of clock cycles spent in each state has been kept small and made constant. In reality, each state can last an indeterminate amount of time.
- **Command** represents a command sent to the VSM using the AXI4-Lite interface or the VSM's AXI4-Stream Control channel.
- The **Load RM** state includes loading the Clearing Bitstream if the device being controlled is an UltraScale device.

Figure 3-19 Shows the behavior of `vsm_<name>_rm_reset` during a Reconfigurable Module load when all optional shutdown and startup steps are enabled.

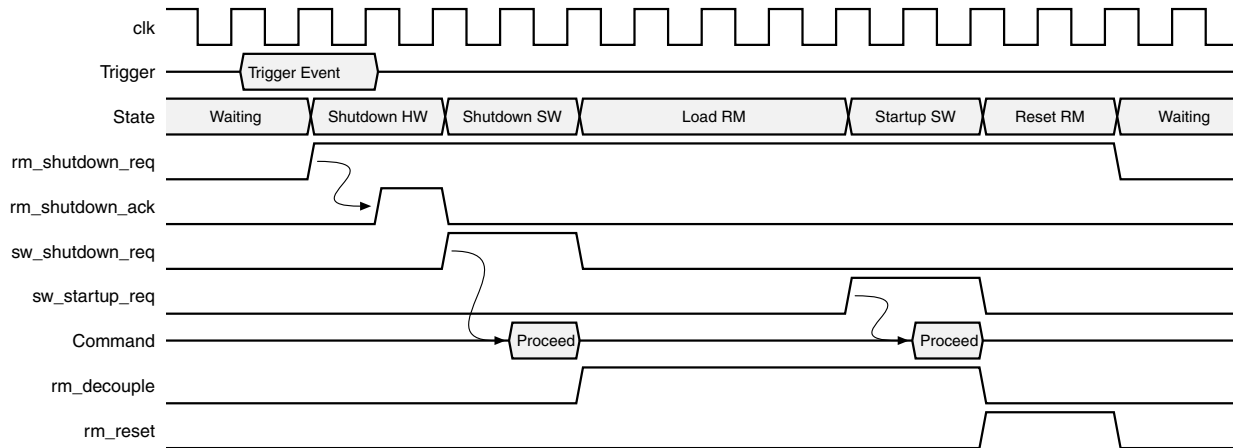


Figure 3-19: `vsm_<name>_rm_reset` (configured as active high) during a Reconfigurable Module load operation when all optional steps are enabled.

Note: `vsm_<name>_rm_reset` asserts immediately after `rm_decouple` deasserts. If the Reconfigurable Module requires an edge sensitive reset rather than a level sensitive reset then additional work may be required to ensure the Reconfigurable Module sees the edge. One option is to ensure that the Reconfigurable Module's clock and reset lines are not decoupled. Another option is to delay the `vsm_<name>_rm_reset` by a clock cycle in the system to ensure that the Reconfigurable Module sees `vsm_<name>_rm_reset` deasserted for at least 1 clock cycle before it asserts.

Figure 3-20 shows `vsm_<name>_rm_reset` being asserted during the startup steps after a core reset. These steps are optional. If the Virtual Socket Manager is configured to skip these steps after reset, `vsm_<name>_rm_reset` is not asserted.

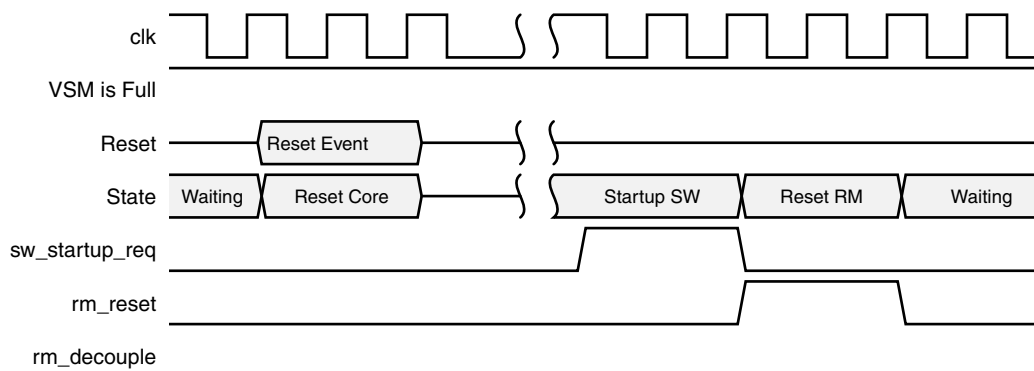


Figure 3-20: `vsm_<name>_rm_reset` (configured as active high) during a core reset while the Virtual Socket Manager is full. Optional Reconfigurable Module startup steps are enabled.

`vsm_<name>_rm_reset` is unaffected by a Partial Reconfiguration Controller reset, and will take on a deasserted level at power-on if the Virtual Socket is full. There is one exception to this rule which may cause `vsm_<name>_rm_reset` to be asserted erroneously after the initial power-on reset.

The `vsm_<name>_rm_reset` signal is deasserted in the power-on configuration based on the Reset Type value programmed using the core's **Customize IP** dialog box, or the core's **set_property** command. The Reset Type value for each Reconfigurable Module can also be programmed directly in the routed netlist using the steps described in [Customizing the Core Post Implementation](#). Programming the netlist directly changes the values stored in the Virtual Socket Manager's registers for this Reconfigurable Module but it does not change the initialization values used by the Virtual Socket Manager. These do not change until the Virtual Socket Manager retrieves the information for the loaded Reconfigurable Module from memory, which occurs approximately 6 clock cycles after reset is released. If the Reset Type of the power-on Reconfigurable Module is programmed directly into the netlist with a different value from that used to originally configure the core, the Virtual Socket Manager continues to use the original value until the updated values are retrieved after reset.

For example:

1. The core's **Customize IP** dialog box is used to inform the Virtual Socket Manager that the power-on Reconfigurable Module has an active high reset. The Virtual Socket Manager will be generated to set `vsm_<name>_rm_reset` to 0 (deasserted).
2. The core's API is used to change the Virtual Socket Manager directly in the netlist to tell it that the power-on Reconfigurable Module has an active low reset.
3. When the Virtual Socket Manager exits power-on reset, `vsm_<name>_rm_reset` will remain at 0 for approximately 6 clock cycles because it was originally configured to have a Reconfigurable Module with an active high reset. The Reconfigurable Module will be inadvertently reset during this time.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [\[Ref 1\]](#)
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 2\]](#)
- *Vivado Design Suite User Guide: Getting Started* (UG910) [\[Ref 5\]](#)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [\[Ref 6\]](#)

Customizing and Generating the Core

This section includes information about using Xilinx® tools to customize and generate the core in the Vivado® Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [\[Ref 1\]](#) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 2\]](#) and the *Vivado Design Suite User Guide: Getting Started* (UG910) [\[Ref 5\]](#).

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

Validation Tab

This tab is available on the left of the Customize IP dialog box and shares space with the IP Symbol tab, the Address Map tab, and the Trigger Mapping tab. This tab is displayed by default. The Validation tab lists any errors that remain with the current configuration. The core cannot be generated until all listed errors are fixed. When the configuration contains no errors, the text "There are no errors" displays in the tab.

Address Map Tab

This tab is available on the left of the Customize IP dialog box and shares space with the IP Symbol tab, the Trigger Mapping tab, and the Validation tab. The Address Map tab provides information about the structure of the AXI Lite address, and provides the name and address of each register in the Partial Reconfiguration Controller

Trigger Mapping Tab

This tab is available on the left of the Customize IP dialog box and shares space with the IP Symbol tab, the Address Map tab, and the Validation tab. The Trigger Mapping tab provides an overview of the trigger mapping that is configured for the selected Virtual Socket.

Global Options

This tab is available on the right of the Customize IP dialog box and shares space with the Virtual Socket Options tab. The Global Options tab is used to configure the parts of the core that do not depend on the number of Virtual Socket Managers, or their configuration. The following options are available:

- **Enable the AXI Lite Interface:** Enables or disables the AXI4-Lite register interface.
- **Reset Active Level:** Configures the core to respond to an active-High or an active-Low reset.
- **Managed Device Type:** Select the type of device to be managed by the core. This is the device where the Virtual Sockets and connected ICAP primitive reside.
- **CAP arbitration protocol:** Select the type of CAP arbitration required.
- **Bitstream Compression:** Select whether the decompression block is required. If enabled, all partial bitstreams must be compressed (see [Partial Bitstream Preparation](#)).
- **FIFO Depth:** The depth of the FIFO in the fetch path. Valid values are 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, and 131072. The value 16 is only available when the number of **CDC Stages** is set to 2 or 3.
- **FIFO Implementation:** Configures the FIFO in the fetch path to use Block RAMs or Distributed RAMs.

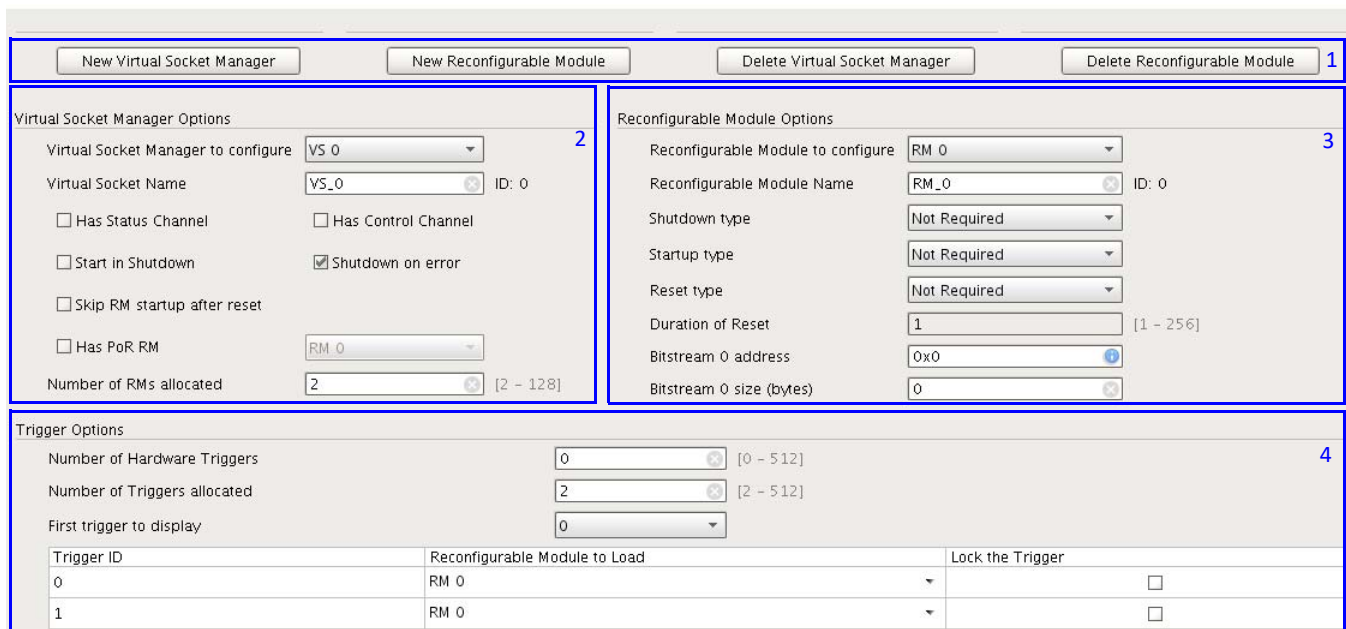
- **CDC Stages:** The number of synchronization stages to use when crossing between clock domains. Valid values are 2, 3, 4, 5, and 6. Using 4, 5, or 6 requires a minimum FIFO depth of 32 entries.

Virtual Socket Manager Options

This tab is available on the right of the Customize IP dialog box and shares space with the Global Options tab. The Virtual Socket Manager Options tab is used to configure a single Virtual Socket Manager, along with a single Reconfigurable Module within the Virtual Socket Manager. The Virtual Socket Manager and Reconfigurable Module to be configured can be selected using drop-down controls.

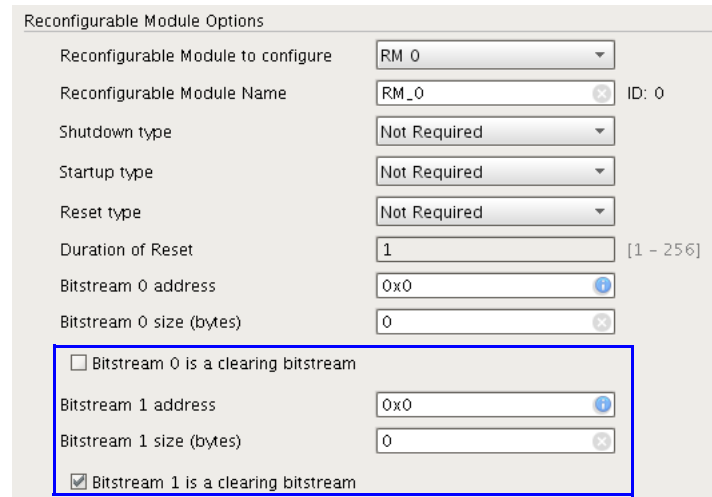
Figure 4-1 and Figure 4-2 shows the Virtual Socket Manager Options. The tab is split into four areas:

1. Control Buttons
2. Virtual Socket Manager Options
3. Reconfigurable Module Options
4. Trigger Options



Trigger ID	Reconfigurable Module to Load	Lock the Trigger
0	RM 0	<input type="checkbox"/>
1	RM 0	<input type="checkbox"/>

Figure 4-1: Virtual Socket Manager Options When Managing a 7 Series or UltraScale+ Device



Reconfigurable Module Options

Reconfigurable Module to configure: RM 0

Reconfigurable Module Name: RM_0 ID: 0

Shutdown type: Not Required

Startup type: Not Required

Reset type: Not Required

Duration of Reset: 1 [1 - 256]

Bitstream 0 address: 0x0

Bitstream 0 size (bytes): 0

☐ Bitstream 0 is a clearing bitstream

Bitstream 1 address: 0x0

Bitstream 1 size (bytes): 0

☒ Bitstream 1 is a clearing bitstream

Figure 4-2: Additional Virtual Socket Manager Options When Managing an UltraScale Device

Control Buttons

There are four control buttons available, three of which are automatically hidden when not required:

- **New Virtual Socket Manager:** Click this button to add a new Virtual Socket Manager to the core.
- **New Reconfigurable Module:** Click this button to add a new Reconfigurable Module to the currently selected Virtual Socket Manager. This button is only available when a Virtual Socket Manager is selected.
- **Delete Virtual Socket Manager:** Click this button to delete the currently selected Virtual Socket Manager. This button is only available when a Virtual Socket Manager is selected.
- **Delete Reconfigurable Module:** Click this button to delete the currently selected Reconfigurable Module from the currently selected Virtual Socket Manager. This button is only available when a Reconfigurable Module is selected.

Virtual Socket Manager Options

- **Virtual Socket Manager to Configure:** Use this drop-down list to select the Virtual Socket Manager to configure.
- **Virtual Socket Name:** [Optional] Enter a new name for the Virtual Socket Manager here. The name must satisfy the following rules:
 - Contains only letters, numbers, or "_" (underscore).
 - Does not start or end with "_" (underscore).

- Does not contain "__" (double underscore).

Note: The name change only takes effect when you click another control in the GUI, or press the **Tab** key.

- **Has Status Channel:** Enables or disables the AXI4-Stream status channel for the selected Virtual Socket Manager.
- **Has Control Channel:** Enables or disables the AXI4-Stream control channel for the selected Virtual Socket Manager.
- **Start in Shutdown:** Select this option if the selected Virtual Socket Manager should start in the shutdown state.
- **Shutdown on Error:** Select this option if the selected Virtual Socket Manager should enter the shutdown state if an error is detected.
- **Skip RM Startup After Reset:** When the Virtual Socket Manager exits reset, it executes the Reconfigurable Module start-up steps (reset and/or software start-up) if a Reconfigurable Module is loaded. Select this option to skip those steps after a reset.
- **Has PoR RM:** Select this option and select the appropriate Reconfigurable Module if the initial configuration bitstream for the device contains a Reconfigurable Module in this Virtual Socket.

Note: This option is mandatory⁽¹⁾ when the device being managed in an UltraScale device. The value is automatically set to true, and Customize IP parameter is disabled in this case.

- **Number of RMs allocated:** Specify how much space should be allocated to store information about Reconfigurable Modules in the Virtual Socket Manager. This is required if you intend to add more Reconfigurable Modules in the field using the AXI4-Lite interface. This value is automatically rounded up to a power of two.

Reconfigurable Module Options

- **Reconfigurable Module to Configure:** Use this drop-down list to select the Reconfigurable Modules in the selected Virtual Socket Manager to configure.
- **Reconfigurable Module Name:** [Optional] Enter a new name for the Reconfigurable Module here. The name must satisfy the following rules:
 - Contains only letters, numbers or "_" (underscore).
 - Does not start or end with "_" (underscore).

1. A Reconfigurable Partition in a static bitstream can be implemented as a black box or a pseudo black box (See *Vivado Design Suite User Guide: Partial Reconfiguration (UG909)* [Ref 3]). When the device being managed is a 7 series or UltraScale+ device, this option can be ignored by the Partial Reconfiguration Controller core if the system has no need to ever revert the Reconfigurable Partition back to being a black box. However, when the device being managed is an UltraScale device, the black box's clearing bitstream must be loaded before another Reconfigurable Module can be loaded, so the black box must be defined as a Reconfigurable Module in the core. If there is no need to ever revert the Reconfigurable Partition back to being a black box, the address and size information for this Reconfigurable Module's partial bitstream can be left at 0. In this case, it should not be mapped to a trigger.

- Does not contain "__" (double underscore).

Note: The name change only takes effect when you click another control in the GUI.

- **Shutdown Type:**

- **Not Required:** The Reconfigurable Module does not need to be informed that it will be removed.
- **Hardware Only:** The Reconfigurable Module needs to be informed that it will be removed, but there is no software that needs to be informed.
- **HW then SW:** The Reconfigurable Module needs to be informed that it will be removed, and the software needs to be informed as well. The Reconfigurable Module should be informed first.
- **SW then HW:** The Reconfigurable Module needs to be informed that it will be removed, and software needs to be informed as well. The software should be informed first.

For more information, see [Reconfigurable Module Hardware and Software Shutdown, page 37](#).

- **Startup Type:**

- **Not Required:** The system software does not need to be informed that the Reconfigurable Module has been loaded.
- **Software Only:** The system software needs to be informed that the Reconfigurable Module has been loaded.

For more information, see [Reconfigurable Module Software Startup, page 43](#).

- **Reset Type:**

- **Not Required:** The Reconfigurable Module does not need to be reset after it has been loaded.
- **Active High:** The Reconfigurable Module needs to be reset with an active-High signal after it has been loaded.
- **Active Low:** The Reconfigurable Module needs to be reset with an active-Low signal after it has been loaded.
- **Duration of Reset:** This control is only enabled if the Reconfigurable Module requires an active-High or active-Low reset. It is used to specify how many clock cycles the Reconfigurable Modules reset should be asserted. The maximum reset duration is 256 clock cycles.
- **Bitstream 0 Address:** The address of this Reconfigurable Module bitstream in the configuration library interface. This must be aligned to a 32-bit word boundary (bottom two bits must be 0).

- **Bitstream 0 Size:** The size (in bytes) of this Reconfigurable Module bitstream in the configuration library interface. This must be a multiple of 32 bits (bottom two bits of the size must be 0).
- **Bitstream 0 is a Clearing Bitstream:** Click this option if this bitstream is a clearing Bitstream. This option is only available when the device to be managed is an UltraScale device.
- **Bitstream 1 Address:** The address of this Reconfigurable Module bitstream in the configuration library interface. This must be aligned to a 32-bit word boundary (bottom two bits must be 0). This option is only available when the device to be managed is an UltraScale device.
- **Bitstream 1 Size:** The size (in bytes) of this Reconfigurable Module bitstream in the configuration library interface. This must be a multiple of 32 bits (bottom two bits of the size must be 0). This option is only available when the device to be managed is an UltraScale device.
- **Bitstream 1 is a Clearing Bitstream:** Click this option if this bitstream is a clearing Bitstream. This option is only available when the device to be managed is an UltraScale device.

Trigger Options

- **Number of Hardware Triggers:** Specify the number of triggers in the Virtual Socket Manager that can be activated from dedicated hardware signals. For more information, see [Hardware Triggers, page 36](#).
- **Number of Triggers Allocated:** Specify the number of triggers that this Virtual Socket Manager has.
- **First Trigger to display:** A Virtual Socket Manager can contain many more triggers than it is efficient to display in the Customize IP dialog box. A maximum of four triggers are shown at any one time, with the first trigger shown selected by this drop-down list.
- **Trigger ID:** [Read only] The identifier of the trigger in each row of the table.
- **Reconfigurable Module to Load:** The Reconfigurable Module to load when the **Trigger ID** trigger is seen for this Virtual Socket Manager.
- **Lock the Trigger:** Used to control whether the trigger is locked to the specified Reconfigurable Module, or whether it will change when Reconfigurable Modules are added or removed.

By default, trigger T loads the Reconfigurable Module with identifier T , modulus the number of Reconfigurable Modules in the Virtual Socket Manager. This means that, by default, every trigger maps to a defined Reconfigurable Module, and all Reconfigurable Modules can be triggered. Adding a new Reconfigurable Module causes the triggers to remap without any intervention.

To override this for a trigger, lock it to the value selected by the **Reconfigurable Module to Load** option. When a trigger is locked, it will not change if new Reconfigurable Modules are added. If a trigger's Reconfigurable Module is subsequently deleted, the trigger will unlock (if locked) and revert to using its default value.

User Parameters

Table 4-1 shows the relationship between the fields in the Vivado IDE and the User Parameters (which can be viewed in the Tcl console).

Table 4-1: Customization Parameters to User Parameter Relationship

Vivado IDE Parameter/Value	User Parameters/Value	Default Value
Enable the AXI Lite Interface	HAS_AXI_LITE_IF	1
Reset Active Level	RESET_ACTIVE_LEVEL	0
CAP arbitration protocol	CP_ARBITRATION_PROTOCOL	0
	0: No Arbitration Required	
	1: Latency has not been added to the arbiter signals	
	2: Latency has been added to the arbiter signals	
Specify if partial bitstreams are compressed	CP_COMPRESSION	0
	0: Partial bitstreams are not compressed	
	1: Partial bitstreams are compressed	
FIFO Depth	CP_FIFO_DEPTH	32
FIFO Implementation	CP_FIFO_TYPE	lutram
Block RAM	blockram	
Distributed RAM	lutram	
Managed Device Type	CP_FAMILY	By default, this uses the device type specified in the project settings.
	7series: If a 7 series device is being managed.	
	ultrascale: If an UltraScale device is being managed.	
	ultrascale_plus: If an UltraScale+ device is being managed.	
CDC Stages	CDC_STAGES	6
Virtual Socket to Configure	No equivalent User Parameter.	
Virtual Socket Name	No equivalent User Parameter The name of each Virtual Socket Manager is specified as part of the user parameter name (shown as <vsname> below)	

Table 4-1: Customization Parameters to User Parameter Relationship (Cont'd)

Vivado IDE Parameter/Value	User Parameters/Value	Default Value
Has Status Channel	VS.<vsname>.HAS_AXIS_STATUS	0
Has Control Channel	VS.<vsname>.HAS_AXIS_CONTROL	0
Start in Shutdown	VS.<vsname>.START_IN_SHUTDOWN	0
Shutdown on Error	VS.<vsname>.SHUTDOWN_ON_ERROR	1
Skip RM Startup After Reset	VS.<vsname>.SKIP_RM_STARTUP_AFTER_RESET	0
Has PoR RM	VS.<vsname>.HAS_POR_RM VS.<vsname>.POR_RM	<ul style="list-style-type: none"> 0: If the Virtual Socket contains no RM in the Power On configuration. 1: If the Virtual Socket contains an RM in the Power On configuration. <p>Note: This must be set to 1 when an UltraScale device is being managed.</p> RM_0
Number of RMs allocated	VS.<vsname>.NUM_RMS_ALLOCATED	2
Reconfigurable Module to Configure	No equivalent User Parameter	
Reconfigurable Module Name	No equivalent User Parameter The name of each Reconfigurable Module is specified as part of the user parameter name (shown as <rmname> below)	
Shutdown Type	VS.<vsname>.RM.<rmname>.SHUTDOWN_REQUIRED	no
Not Required	no	
Hardware Only	hw	
HW then SW	hw/sw	
SW then HW	sw/hw	
Startup Type	VS.<vsname>.RM.<rmname>.STARTUP_REQUIRED	no
Not Required	no	
Software Only	sw	
Reset Type	VS.<vsname>.RM.<rmname>.RESET_REQUIRED	no
Not Required	no	
Active High	high	
Active Low	low	
Duration of Reset	VS.<vsname>.RM.<rmname>.RESET_DURATION	1

Table 4-1: Customization Parameters to User Parameter Relationship (Cont'd)

Vivado IDE Parameter/Value	User Parameters/Value	Default Value
Bitstream Address	VS.<vsname>.RM.<rmname>.BS.<bsid>.ADDRESS ⁽¹⁾	0
Bitstream Size	VS.<vsname>.RM.<rmname>.BS.<bsid>.SIZE ⁽¹⁾	0
Bitstream is Clearing	VS.<vsname>.RM.<rmname>.BS.<bsid>.CLEAR ⁽¹⁾	0
Number of Hardware Triggers	VS.<vsname>.NUM_HW_TRIGGERS	0
Number of Triggers Allocated	VS.<vsname>.NUM_TRIGGERS_ALLOCATED	2
First Trigger to Display	No equivalent User Parameter	
Trigger ID	No equivalent User Parameter See "Reconfigurable Module to Load"	
Reconfigurable Module to Load	VS.<vsname>.TRIGGER<trigger_id>_TO_RM	trigger_id mod Number of Reconfigurable Modules defined in the Virtual Socket Manager
Lock the Trigger	No equivalent User Parameter Only triggers that have non-default values need to be specified, which has the effect of locking that trigger	

Notes:

1. <bsid> can only be 0 when the device to be managed is a 7 series or an UltraScale+ device. <bsid> can be 0 or 1 when the device being managed is an UltraScale device.

Configuring Tcl User Parameters

The Partial Reconfiguration Controller can be configured from the Tcl command line by setting properties directly. A custom `set_property` command is required. The following command has to be executed in the Vivado Tcl command line to access `prc_v1_3::set_property`:

```
source [get_property REPOSITORY \
      [get_ipdefs *prc:1.3]]/xilinx/prc_v1_3/tcl/api.tcl -notrace
```

Example:

```
create_ip -name prc -vendor xilinx.com -library ip -module_name dut
prc_v1_3::set_property -dict [list \
    CONFIG.HAS_AXI_LITE_IF 0 \
    CONFIG.RESET_ACTIVE_LEVEL 1 \
    CONFIG.CP_FIFO_DEPTH 16 \
    CONFIG.CP_ARBITRATION_PROTOCOL 0 \
    CONFIG.CP_COMPRESSION 0 \
    CONFIG.CP_FIFO_TYPE lutram \
    CONFIG.VS0.HAS_AXIS_STATUS 0 \
    CONFIG.VS0.HAS_AXIS_CONTROL 0 \
    CONFIG.VS0.NUM_TRIGGERS_ALLOCATED 4 \
    CONFIG.VS0.NUM_HW_TRIGGERS 4 \
    CONFIG.VS0.NUM_RMS_ALLOCATED 2 \
    CONFIG.VS0.POR_RM 0 \
    CONFIG.VS0.SKIP_RM_STARTUP_AFTER_RESET 0 \
    CONFIG.VS0.START_IN_SHUTDOWN 0 \
    CONFIG.VS0.SHUTDOWN_ON_ERROR 0 \
    CONFIG.VS0.RM0.SHUTDOWN_REQUIRED no \
    CONFIG.VS0.RM0.STARTUP_REQUIRED no \
    CONFIG.VS0.RM0.RESET_REQUIRED high \
    CONFIG.VS0.RM0.RESET_DURATION 3 \
    CONFIG.VS0.RM0.BS.0.ADDRESS 0xAE9DF4 \
    CONFIG.VS0.RM0.BS.0.SIZE 375300 \
    CONFIG.VS0.RM1.SHUTDOWN_REQUIRED sw/hw \
    CONFIG.VS0.RM1.STARTUP_REQUIRED sw \
    CONFIG.VS0.RM1.RESET_REQUIRED high \
    CONFIG.VS0.RM1.RESET_DURATION 10 \
    CONFIG.VS0.RM1.BS0.ADDRESS 0xB45840 \
    CONFIG.VS0.RM1.BS0.SIZE 375300 \
    CONFIG.VS1.HAS_AXIS_STATUS 0 \
    CONFIG.VS1.HAS_AXIS_CONTROL 0 \
    CONFIG.VS1.NUM_TRIGGERS_ALLOCATED 4 \
    CONFIG.VS1.NUM_HW_TRIGGERS 4 \
    CONFIG.VS1.NUM_RMS_ALLOCATED 2 \
    CONFIG.VS1.POR_RM 1 \
    CONFIG.VS1.SKIP_RM_STARTUP_AFTER_RESET 0 \
    CONFIG.VS1.START_IN_SHUTDOWN 0 \
    CONFIG.VS1.SHUTDOWN_ON_ERROR 0 \
    CONFIG.VS1.RM0.SHUTDOWN_REQUIRED hw/sw \
    CONFIG.VS1.RM0.STARTUP_REQUIRED no \
    CONFIG.VS1.RM0.RESET_REQUIRED high \
    CONFIG.VS1.RM0.RESET_DURATION 16 \
    CONFIG.VS1.RM0.BS0.ADDRESS 0xBA128C \
    CONFIG.VS1.RM0.BS0.SIZE 404792 \
    CONFIG.VS1.RM1.SHUTDOWN_REQUIRED no \
    CONFIG.VS1.RM1.STARTUP_REQUIRED no \
    CONFIG.VS1.RM1.RESET_REQUIRED high \
    CONFIG.VS1.RM1.RESET_DURATION 32 \
    CONFIG.VS1.RM1.BS0.ADDRESS 0xC0400C \
    CONFIG.VS1.RM1.BS0.SIZE 404792 \
] [get_ips dut]
generate_target {all} [get_ips dut]
```



RECOMMENDED: *If `prc_v1_3::set_property` is called from within a script, Xilinx recommends that you source the script using the `-notrace` option. Large core configurations can take a substantial amount of time to complete when `-notrace` is not used.*

Output Generation

The Partial Reconfiguration Controller delivers standard synthesis and simulation models. For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 2].

In addition, a configuration information text file is delivered with the core, and located in:

```
<ip source dir>/documentation/configuration_information.txt
```

This text file contains the following information:

- The property values used to configure the core.
- The integer identifiers assigned to the Virtual Socket Manager and the Reconfigurable Modules.
 - The Virtual Socket Manager identifiers are required to access the registers in the Virtual Socket Managers.
 - The Reconfigurable Module identifiers are required to access the correct registers in the Reconfigurable Module information register bank. They are also required if the triggers are reprogrammed at run-time.
- The MSBs and LSBs of the following address fields:
 - Virtual Socket Manager Select
 - Bank Select
 - Register Select
- The address of each register in the generated core.

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

This section is not application for this IP core.

Device, Package, and Speed Grade Selections

The Partial Reconfiguration Controller works with 7 series, UltraScale and UltraScale+ devices.

Clock Frequencies

The ICAP clock has to be constrained to the maximum frequency of the ICAP port, or less. For the maximum frequency for your device family, see the applicable DC and AC switching characteristics data sheet (refer to [References in Appendix C](#)).

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 6].



IMPORTANT: For cores targeting 7 series or Zynq®-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 2].

Customizing the Core Post Implementation

The registers in the following Virtual Socket Manager register banks can be configured directly in the static netlist used to configure the device:

- [Bank 1: Trigger to Reconfigurable Module Registers](#)
- [Bank 2: Reconfigurable Module Information Registers](#)
- [Bank 3: Bitstream Information Registers](#)

Customizing the core post implementation can be useful when the partial bitstream sizes are not known when the Partial Reconfiguration Controller is initially configured, or when the final set of Reconfigurable Modules have not been decided when the Partial Reconfiguration Controller is initially configured.



IMPORTANT: Changes made to the netlist are not reflected in the original core configuration. Updating the original core configuration could trigger a new implementation run of the entire design.

To customize the core post implementation, type the following commands:

1. Load the Partial Reconfiguration Controller Tcl API.

```
source [get_property REPOSITORY [get_ipdefs *prc:1.3]]/xilinx/prc_v1_3/tcl/api.tcl -
notrace
```

2. Load the Partial Reconfiguration Controller IP's configuration.

- If the Partial Reconfiguration Controller IP definition and the netlist are already open, type this command:

```
set config [get_property CONFIG.ALL_PARAMS [get_ips <IP NAME>]]
```

- Otherwise, type these commands instead:

```
open_project <path to xpr file>
set config [get_property CONFIG.ALL_PARAMS [get_ips <IP NAME>]]
close_project
```

3. Create a descriptor for the Partial Reconfiguration Controller instance in the netlist you want to modify.

```
set dscr [prc_v1_3::netlist::get_descriptor $config <ip path>]
```

Note: <ip path> is the path to the Partial Reconfiguration Controller instance in the netlist. For example, i_prc/U0.

See [Mandatory Commands](#) for more information.

4. Modify the descriptor as required using the [High Level Commands](#) or [Low Level Commands](#). For example,

```
prc_v1_3::netlist::set_trigger          dscr vs_shift 0 rm_shift_left
prc_v1_3::netlist::set_rm_bs_address    dscr vs_shift rm_shift_left      0xAE9DF4
prc_v1_3::netlist::set_rm_bs_size       dscr vs_shift rm_shift_left      375300
```

5. Apply the changes in the descriptor to the Partial Reconfiguration Controller instance in the netlist.

```
prc_v1_3::netlist::apply_descriptor    dscr
```

6. Save the netlist in a checkpoint

```
write_checkpoint Implement/Config_shift_right_count_up/top_route_design.dcp -force
```



TIP: An example is provided in [Example Usage](#).

Note: The command descriptions below use the following terms:

<descriptor>	A data structure returned by <code>get_descriptor</code> .
<vsm_name>	The name of the Virtual Socket Manager you want to change.
<rm_name>	The name of the Reconfigurable Module you want to change, or in the case of trigger modification, the Reconfigurable Module you want to select.
<table_type>	The table holding the Register Bank you want to modify. The valid values are:
trigger_table	Trigger to Reconfigurable Module Registers. For details, see Table 2-7 .
rm_bs_index_table	BS Index Registers in the Reconfigurable Module Information Registers. For details, see Table 2-9 .
rm_ctrl_table	Control Registers in the Reconfigurable Module Information Registers. For details, see Table 2-10 .
bs_id_table	ID Registers in the Bitstream Information Registers. For details, see Table 2-12 .

bs_address_table	Address Registers in the Bitstream Information Registers. For details, see Table 2-13 .
bs_size_table	Size Registers in the Bitstream Information Registers. For details, see Table 2-14 .

Mandatory Commands

These commands must be run when customizing the core post implementation.

Table 4-2: Mandatory Commands

Command	Description
get_descriptor <configuration> <path to instance in netlist>	Returns a data structure that is needed by all other commands. <configuration> is the value of the IP CONFIG.ALL_PARAMS parameter. You can access this value using <code>get_property CONFIG.ALL_PARAMS [get_ips <ip name>]</code>
apply_descriptor <descriptor>	Write the descriptor information back into the netlist.

Low Level Commands

These commands let you access and modify the tables that implement the register banks directly.

Table 4-3: Low Level Commands

Command	Description
get_table_entry <descriptor> <vsm_name> <table_type> <row>	Get the value of the row of the named table in the Virtual Socket Manager <vsm_name>.
set_table_entry <descriptor> <vsm_name> <table_type> <row> <value>	Set the value of the row of the named table in the Virtual Socket Manager <vsm_name> to <value>. No error checking is performed on this value.
print_table_entry <descriptor> <vsm_name> <table_type> <row>	Print the value of the row of the named table in the Virtual Socket Manager <vsm_name>.
print_table <descriptor> <vsm_name> <table_type> <row>	Print the entire named table in the Virtual Socket Manager <vsm_name>. Note that this prints the entire memory even if only a subset of the addresses are used. For example, if the design has two triggers allocated, the trigger table will be implemented in a 32 element deep Distributed RAM. All 32 rows will be printed.

High Level Commands

These commands get and set values through an abstraction layer that contains error checking.

Table 4-4: High Level Commands

Command	Description
get_trigger <descriptor> <vsm_name> <trigger_id>	Get the Reconfigurable Module that is selected by the trigger <trigger_id> in the Virtual Socket Manager <vsm_name>.
set_trigger <descriptor> <vsm_name> <trigger_id> <rm_name>	Map the trigger <trigger_id> to the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_shutdown_required <descriptor> <vsm_name> <rm_name>	Get the shutdown required value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_shutdown_required <descriptor> <vsm_name> <rm_name> <val>	Set the shutdown required value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_startup_required <descriptor> <vsm_name> <rm_name>	Get the startup required value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_startup_required <descriptor> <vsm_name> <rm_name> <val>	Set the startup required value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_reset_required <descriptor> <vsm_name> <rm_name>	Get the reset required value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_reset_required <descriptor> <vsm_name> <rm_name> <val>	Set the reset required value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_reset_duration <descriptor> <vsm_name> <rm_name>	Get the reset duration value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_reset_duration <descriptor> <vsm_name> <rm_name> <val>	Set the reset duration value for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_bs_address <descriptor> <vsm_name> <rm_name>	Get the address in memory of the partial bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_bs_address <descriptor> <vsm_name> <rm_name> <address>	Set the address in memory of the partial bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_bs_size <descriptor> <vsm_name> <rm_name>	Get the size in memory of the partial bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.

Table 4-4: High Level Commands (Cont'd)

Command	Description
set_rm_bs_size <descriptor> <vsm_name> <rm_name> <size>	Set the size in memory of the partial bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_clearing_bs_address <descriptor> <vsm_name> <rm_name>	Get the address in memory of the clearing bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_clearing_bs_address <descriptor> <vsm_name> <rm_name> <address>	Set the address in memory of the clearing bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_clearing_bs_size <descriptor> <vsm_name> <rm_name>	Get the size in memory of the clearing bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_clearing_bs_size <descriptor> <vsm_name> <rm_name> <size>	Set the size in memory of the clearing bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_bs_id <descriptor> <vsm_name> <rm_name>	Get the identifier of the partial bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_bs_id <descriptor> <vsm_name> <rm_name> <id>	Set the identifier of the partial bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_clearing_bs_id <descriptor> <vsm_name> <rm_name>	Get the identifier of the clearing bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_clearing_bs_id <descriptor> <vsm_name> <rm_name> <id>	Set the identifier of the clearing bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_bs_index <descriptor> <vsm_name> <rm_name>	Get the BS_INDEX field from the RM_BS_INDEX register for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
set_rm_bs_index <descriptor> <vsm_name> <rm_name> <val>	[ADVANCED] This command is not usually required because the BS_INDEX is automatically set during core configuration and when <code>create_rm</code> is used. Set the BS_INDEX field in the RM_BS_INDEX register to specify which row of the Bitstream Information tables holds the partial bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.
get_rm_clearing_bs_index <descriptor> <vsm_name> <rm_name>	Get the CLEAR_BS_INDEX field from the RM_BS_INDEX register for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.

Table 4-4: High Level Commands (Cont'd)

Command	Description
set_rm_clearing_bs_index <descriptor> <vsm_name> <rm_name> <val>	<p>[ADVANCED] This command is not usually required because the CLEAR_BS_INDEX is automatically set during core configuration and when <code>create_rm</code> is used.</p> <p>Set the CLEAR_BS_INDEX field in the RM_BS_INDEX register to specify which row of the BS Information tables holds the clearing bitstream for the Reconfigurable Module <rm_name> in the Virtual Socket Manager <vsm_name>.</p>
create_rm <descriptor> <vsm_name> <rm_name> [partial_index] [clearing_index]	<p>Add a new Reconfigurable Module to the Virtual Socket Manager <vsm_name>. This maps a Reconfigurable Module with the name <rm_name> to the next available Reconfigurable Module identifier (which is the first free row in the Reconfigurable Module Information table).</p> <p>This new Reconfigurable Module's identifier is returned by the function.</p> <p>The index values for the partial bitstream and the clearing bitstream (if required) can be optionally specified. If not specified, the first free rows in the Bitstream Information tables are used.</p>

Example Usage

The following example can be used as a starting point for your own customization.

```
# Store a copy of the Partial Reconfiguration Controller's original configuration.
# If the netlist to be modified is in an open project along with the
# Partial Reconfiguration Controller IP, the open/close project commands will
# not be needed
#
open_project ./Sources/generated/prc.xpr
set config [get_property CONFIG.ALL_PARAMS [get_ips <IP NAME>]]
close_project

source [get_property REPOSITORY [get_ipdefs *prc:1.3]]/xilinx/prc_v1_3/tcl/api.tcl -
notrace

# Change this to point at the core instance in the netlist
set dscr [prc_v1_3::netlist::get_descriptor $config "i_prc/U0" ]

prc_v1_3::netlist::set_trigger          dscr vs_shift 0 rm_shift_left
prc_v1_3::netlist::set_trigger          dscr vs_shift 1 rm_shift_right
prc_v1_3::netlist::set_rm_shutdown_required dscr vs_shift rm_shift_left      hw
prc_v1_3::netlist::set_rm_startup_required dscr vs_shift rm_shift_left      sw
prc_v1_3::netlist::set_rm_reset_required  dscr vs_shift rm_shift_left      low
prc_v1_3::netlist::set_rm_reset_duration  dscr vs_shift rm_shift_left      24
prc_v1_3::netlist::set_rm_bs_address      dscr vs_shift rm_shift_left      0xAE9DF4
prc_v1_3::netlist::set_rm_bs_size         dscr vs_shift rm_shift_left      375300
prc_v1_3::netlist::set_rm_bs_address      dscr vs_shift rm_shift_right     0xB45840
prc_v1_3::netlist::set_rm_bs_size         dscr vs_shift rm_shift_right     375300
prc_v1_3::netlist::set_rm_bs_address      dscr vs_count rm_count_up        0xBA128C
prc_v1_3::netlist::set_rm_bs_size         dscr vs_count rm_count_up        404792
prc_v1_3::netlist::set_rm_bs_address      dscr vs_count rm_count_down     0xC0400C
prc_v1_3::netlist::set_rm_bs_size         dscr vs_count rm_count_down     404792
prc_v1_3::netlist::apply_descriptor      dscr

write_checkpoint Implement/Config_shift_right_count_up/top_route_design.dcp -force
```

Important Notes

- Changes made to the netlist directly are not reflected in the original IP's configuration. Doing so could trigger a new implementation run of the entire design
- These commands use the IP's configuration as their starting point. Any changes made in the netlist will be lost if a new descriptor is created and further changes made. For example, the following steps will cause Reconfigurable Module "A" to be lost:
 1. A descriptor is created based on the IP's configuration (`get_descriptor`) and Reconfigurable Module "A" is added.
 2. The descriptor is applied to the netlist (`apply_descriptor`). The PRC instance in the netlist now contains the new Reconfigurable Module "A".
 3. A descriptor is created based on the IP's configuration (`get_descriptor`) because further changes have to be made.

4. Reconfigurable Module "A" has now been lost. The descriptor is only based on the IP's configuration and Reconfigurable Module "A" does not exist in that (it was only added to the PRC instance in the netlist, not to the PRC's configuration).

To avoid losing netlist changes, all changes must be made to the same descriptor.

Partial Bitstream Preparation

The following process should be used to create partial bitstreams for use with the Partial Reconfiguration Controller core:

1. Use `write_bitstream` to generate partial bitstreams with a `.bin` extension.
2. Format the partial bitstreams using the `format_bin_for_icap` API function. For example:

```
prc_v1_3::format_bin_for_icap -i $input_file -o $output_file
```

The `-o` switch is optional. If omitted, the bitstream will be written to `$input_file.bin_for_icap`.

If bitstream compression has been enabled in the core then the `-c 1` parameter must be added to the command. If bitstream compression is disabled then `-c 0` can be added, or the `-c` option can be completely omitted. For example:

```
prc_v1_3::format_bin_for_icap -i $input_file -o $output_file
# Does not compress the file
prc_v1_3::format_bin_for_icap -i $input_file -o $output_file -c 0
# Does not compress the file
prc_v1_3::format_bin_for_icap -i $input_file -o $output_file -c 1
# Compresses the file
```

The `format_bin_for_icap` API function can optionally perform byte swapping on the file. To enable this, add `-bs 1` to the command. To disable this behavior, add `-bs 0` to the command (or omit the `-bs` option). Whether byte swapping is required depends on the design of the system. For example, if the partial bitstreams are being stored in BPI Flash, or transported over TFTP (see *Loading Partial Bitstreams using TFTP* [Ref 15]) then byte swapping is required. If the partial bitstreams are to be fetched directly from DDR memory, byte swapping is not required.

If you are unsure if byte swapping is required, a simple hardware check can be performed:

- a. Create a version of the design with an ILA on the PRC's Configuration Library interface.
- b. Trigger the ILA on `m_axi_mem_rvalid` asserting and look at the data on `m_axi_mem_rdata`.

- c. If compression is *disabled*, the following words should be quickly seen:

000000bb

11220044

aa995566

Note: These will be surrounded by FFFFFFFF words and may not appear immediately next to each other.

- d. If compression is *enabled*, the first word to be fetched should be 950000XX, where XX can take on any value. For example, 950000F, 9500001C.
- e. If you receive (compression off):

bb000000

44002211

665599aa

or

XX000095 (compression on)

then the byte swapping option needs to be changed.

If any other values are received, the partial bitstream isn't stored at the location programmed into the PRC.

3. Format the files created by `format_bin_for_icap` for storage if required. For example, to create a BPI flash image for the KC705 board:

```
write_cfgmem -force -checksum FF -size 32\  
-format MCS\  
-interface BPIx16\  
-loadbit "up 0 static.bit\  
-loaddata "up 00574EFA shift_left_partial.bin\  
up 005A2C20 shift_right_partial.bin\  
up 005D0946 count_up_partial.bin\  
up 00602006 count_down_partial.bin" \  
pr_prom
```

4. The addresses used must match the bitstream addresses programmed into the core. Also, the BIN file sizes must match the bitstream sizes programmed into the core. The sizes are available from the file system or the Vivado Tcl command line using this command:

```
file size <partial>.bin
```

Note: This information is required when the core is configured, but is not available until the static and partial bitstreams have been generated. There are several ways to program the core with this information:

- Program the values directly into the netlist using the commands described in [Customizing the Core Post Implementation](#).
- Use the AXI4-Lite interface to program the core at run time.
- Leave the information as zero in the Partial Reconfiguration Controller, implement the entire design, obtain the required values, configure the Partial Reconfiguration Controller with the correct information, and execute a second complete implementation run. This approach is not suitable if the partial bitstream files change size between runs. This can occur if bitstream compression is enabled, or if you change any of the following:
 - the composition of the Pblocks of the Reconfigurable Partition.
 - the bitstream generation options, such as, per-frame CRC.

If the design uses the PR Bitstream Monitor IP to trace partial bitstream flow then `prc_v1_3::format_bin_for_icap` can be used to instrument the partial bitstreams with the required identifiers.

Note: If bitstream compression is used in the PRC, `prc_v1_3::format_bin_for_icap` *must* be used for this.

To enable this, add `-insert_ids 1` to the command. To disable the behavior, add `-insert_ids 0` to the command (or omit the `-insert_ids` option).

When `-insert_ids 1` is used, the following command switches become mandatory:

- `-sp_id <32 bit identifier>`: Static Partition Identifier
- `-rp_id <32 bit identifier>`: Reconfigurable Partition Identifier
- `-rm_id <32 bit identifier>`: Reconfigurable Module Identifier
- `-bs_id <32 bit identifier>`: Bitstream Identifier

Note: For an explanation of these option, see the PR Bitstream Monitor product guide [\[Ref 16\]](#).

For example, the following command compresses the bin file and inserts:

- A static partition ID of 0xaabbccdd
- An RP ID of 0
- An RM ID of 1
- A Bitstream ID of 123

```
prc_v1_3::format_bin_for_icap -i rp0rm1.bin -o rp0rm1_ids.bin
-insert_ids 1 -sp_id 0xaabbccdd -rp_id 0 -rm_id 1 -bs_id 123 -c 1
```

Migrating and Upgrading

This appendix contains information about migrating a design from the ISE® Design Suite to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [\[Ref 7\]](#).

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

None

Port Changes

The timing of `vsm_<name>_rm_shutdown_req` has changed in this release.

Debugging

This appendix includes details about resources available on the Xilinx® Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Partial Reconfiguration Controller, the [Xilinx Support web page](#) (Xilinx Support web page) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the Partial Reconfiguration Controller. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Documentation Navigator from the [Downloads page](#). For more information, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main [Xilinx support web page](#). To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Partial Reconfiguration Controller

AR: [62044](#)

Technical Support

Xilinx provides technical support in the [Xilinx Support web page](#) for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the [Xilinx Support web page](#).

Debug Tools

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [\[Ref 8\]](#).

Hardware Debug

If the Partial Reconfiguration Controller does not load a new Reconfigurable Module, the following tips might be useful in debugging the issue:

- Visually check that the core is not being held in reset. The reset level is programmable and needs to match the active level of your design's reset.
- Enable the AXI4-Stream status channel on the failing Virtual Socket Manager. This allows live monitoring of the Virtual Socket Manager's operation. It can be very useful to add ILAs here, on the Configuration Library interface, and on the ICAP interface.
- Make sure the failing Virtual Socket Manager is not in the Shutdown state. A bit in the status information will tell you this. If the Virtual Socket Manager is in the Shutdown state, bring it out of shutdown using the appropriate `Restart` command.
- Make sure the registers are programmed correctly. The incoming trigger selects a row in the Trigger to Reconfigurable Module register bank which gives the row number in the Reconfigurable Module Information register bank to use. The `RM_BS_INDEX` register in this row provides the row number in the Bitstream Information register bank to use. The registers in this row provide the size and address in memory of the bitstream to be loaded. These are set correctly when the core is generated, but can be incorrect if programmed through the AXI Lite interface.
 - An ILA on the Configuration Library interface can help here. `m_axi_mem_araddr` gives the address from which the Virtual Socket Manager is trying to fetch the bitstream.
- Make sure the bitstreams are loaded into the memory addresses programmed into the core.
 - An ILA on the Configuration Library interface can help here. `m_axi_mem_rdata` contains the bitstream data read from memory. This should quickly contain the sync word `0xAA995566 (31:0)`.
 - Use the Linux command `xxd -c 4` to convert the BIN file to a HEX file. The data seen on `m_axi_mem_rdata` should match the HEX data directly.
- Make sure the bitstream sizes programmed into the core are the sizes in bytes, not words.
- Make sure the bitstreams are formatted correctly for the core.
 - Bitstreams must be created as described in [Partial Bitstream Preparation](#).
 - An ILA on the Configuration Library interface can help here. `m_axi_mem_rdata` contains the bitstream data read from memory. This should quickly contain the sync word `0xAA995566 (31:0)`.

- Check the ICAP interface using an ILA.
 - If an arbiter is implemented, make sure it grants access to the ICAP.
 - If an arbiter is not implemented, make sure the CAP_GNT port is tied to 1 and the CAP_REL port is tied to 0.
- Check the status channel for errors.
- Check that the Virtual Socket Manager is not waiting for:
 - a shutdown acknowledge response from the Reconfigurable Module or from the software, or
 - a startup acknowledge response from the software.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

References

These documents provide supplemental material useful with this product guide:

1. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))
2. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
3. *Vivado Design Suite User Guide: Partial Reconfiguration* ([UG909](#))
4. *Vivado Design Suite Tutorial: Partial Reconfiguration* ([UG947](#))
5. *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
6. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
7. *ISE to Vivado Design Suite Migration Guide* ([UG911](#))
8. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
9. [Partial Reconfiguration in the Vivado Design Suite](#) page on Xilinx.com
10. *Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics* ([DS181](#))
11. *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* ([DS182](#))
12. *Virtex-7 T and XT FPGAs Data Sheet: DC and AC Switching Characteristics* ([DS183](#))
13. *Zynq-7000 All Programmable SoC (Z-7010, Z-7015, and Z-7020): DC and AC Switching Characteristics Data Sheet* ([DS187](#))
14. *Zynq-7000 All Programmable SoC (Z-7030, Z-7045, and Z-7100): DC and AC Switching Characteristics Data Sheet* ([DS191](#))
15. *Loading Partial Bitstreams using TFTP* ([XAPP1292](#))

16. *Partial Reconfiguration Bitstream Monitor Product Guide* ([PG304](#))

17. *Partial Reconfiguration AXI Shutdown Manager Product Guide* ([PG305](#))

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/04/2017	1.3	<ul style="list-style-type: none"> Added support for the PR Bitstream Monitor IP's identifiers in compressed bitstreams. Changed the behavior of the <code>vsm_<name>_rm_shutdown_req</code> output.
10/4/2017	1.2	Added support for bitstream compression.
10/05/2016	1.1	<ul style="list-style-type: none"> Added support for controlling partial reconfiguration on UltraScale+ families. Added CP_ARBITRATION_PROTOCOL user parameter
04/06/2016	1.0	<ul style="list-style-type: none"> "Unsupported Features" section updated to clarify encrypted bitstream support "Decoupling" section added to document the revised <code>vsm_<name>_rm_decouple</code> behavior "RM_RESET" section added to document the revised <code>vsm_<name>_rm_reset</code> behavior Modified the "Status Register" description (Table 2-4) to include a new state (011) Clarified the maximum duration of <code>vsm_<name>_rm_reset</code> as 256 clock cycles
11/18/2015	1.0	Added support for UltraScale+ families.
04/01/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A

SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2018 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.