

Partial Reconfiguration Bitstream Monitor v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG304 (v1.0) December 5, 2018

Discontinued IP



Table of Contents

Chapter 1: IP Facts	4
Features.....	4
IP Facts.....	4
Chapter 2: Overview	6
Feature Summary.....	6
Unsupported Features.....	7
Licensing and Ordering.....	8
Chapter 3: Product Specification	9
Overview.....	9
Performance.....	15
Resource Use.....	15
Port Descriptions.....	15
Register Space.....	18
Chapter 4: Designing with the Core	23
USR_ACCESS Primitive.....	23
Bitstream Growth.....	23
Clocking.....	23
Chapter 5: Design Flow Steps	25
Customizing and Generating the Core.....	25
Constraining the Core.....	32
Simulation.....	33
Synthesis and Implementation.....	33
Partial Bitstream Preparation.....	33
Appendix A: Upgrading	36
Appendix B: Debugging	37
Finding Help on Xilinx.com.....	37

Debug Tools..... 38

Appendix C: Additional Resources and Legal Notices..... 40

Xilinx Resources..... 40

Documentation Navigator and Design Hubs..... 40

References..... 41

Training Resources..... 41

Revision History..... 41

Please Read: Important Legal Notices..... 42

Discontinued IP

IP Facts

The Xilinx® Partial Reconfiguration Bitstream Monitor can be used to identify partial bitstreams as they flow through the design. This information can be used for debugging or to help manage system applications such as blocking bitstream loads.

Features

- ICAP, AXI4MM (partial support), AXI4-Lite and Generic datapaths supported
- Partial Bitstreams can be traced in the Configuration Engine
- Live and buffered status
- Optional Signal based or AXI4-Lite control
- Optional AXI4-Lite status (signal status is always available)

IP Facts

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family ⁽¹⁾	UltraScale+™, UltraScale™, 7 series
Supported User Interfaces	AXI4-Lite
Resources	Performance and Resource Use 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	None
Tested Design Flows ³	
Design Entry	

LogiCORE IP Facts Table	
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 software development kit (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](#).

Discontinued IP

Overview

Mistakes in bitstream storage, handling, and formatting can cause failures in Partially Reconfigurable designs that can be frustrating and time consuming to debug. Commonly seen problems are:

- Bit files are used instead of bin files
- The system's Partial Reconfiguration Controller (either the PRC core, software, or custom logic) is given the wrong size or address information for the partial bitstreams
- Partial bitstreams have incorrect bit-swapping or endian setting
- Partial bitstreams from the latest implementation run are not used
- The logic communicating with the configuration port is incorrect

One or more Bitstream Monitors can be used to trace the flow of partial bitstreams from storage into the configuration engine. When armed, identifiers embedded at key places in partial bitstreams are extracted and reported by the core. This information can be passed to Vivado HW Debugger using an ILA core to work out what partial bitstream was fetched, if it was fetched in its entirety, and how far through the datapath it went.

This information can also be used by the system for other purposes such as:

- Blocking partial bitstreams that are intended for another static logic
- Blocking partial bitstreams that are not for the Reconfigurable Partition being debugged
- Blocking certain partial bitstreams from being loaded if other system conditions have not been met

Feature Summary

Multiple Protocol Support

The Bitstream Monitor IP can extract bitstream information from AXI4MM, AXI4-Lite, and ICAP buses. It also provides a “generic” protocol which can be used to attach the monitor to any protocol using some glue logic. The IP can be configured to monitor the AXI4MM and AXI4-Lite Read or Write channels.

The monitor only supports the subset of the AXI4MM protocol that is needed to work with the Partial Reconfiguration Controller IP core.

Start and End of Bitstream Reporting

The Bitstream Monitor IP reports whether it is the start or the end of the bitstream that has been seen. This feature is used to ensure that an entire bitstream is delivered to the configuration port.

Configuration Engine Support

The Bitstream Monitor IP can be configured to monitor bitstreams that have passed through the configuration port and into the configuration engine. This makes use of the AXSS register in the configuration engine, which is visible in the fabric using the `USR_ACSESSE2` primitive.

One Shot and Continuous Arming

The Bitstream Monitor IP can be programmed at run-time to report all bitstreams it sees, or to just report the next bitstream event (start or end) that it sees. It can be disarmed at any time.

Configurable History Depth and Behaviour

All observed events are stored in a buffer for later retrieval. The depth of this buffer is configurable, as is the core's behaviour when the buffer becomes full. In this case, the core can be configured to either discard the oldest data from the buffer to make space for the newest data, or it can be configured to discard the newest data.

Multiple Options for Status and Control

The Partial Reconfiguration Bitstream Monitor can be controlled and queried using signals or an AXI4-Lite interface.

Works with Partial Reconfiguration Controller

The subset of AXI4MM supported exactly matches the subset of AXI4MM used by the Partial Reconfiguration Controller IP.

Unsupported Features

AXI4MM

The Partial Reconfiguration Bitstream Monitor only supports a subset of the AXI4MM protocol. Specifically:

- Address and ID information is not used.

- Data is read from the bottom 32 bits of the data bus regardless of data bus width, write strobes or starting address.

USR_ACCESS and AXSS

Bitstreams annotated for use with the Partial Reconfiguration Bitstream Monitor set the AXSS register in the configuration engine, which changes the value read through the USR_ACCESE2 primitive. Designs that need to use the value in the AXSS register after the first partial bitstream has been loaded may be incompatible with the IP.

Licensing and Ordering

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](#).

For more information about this core, visit the [Partial Reconfiguration Bitstream Monitor](#) product web page.

Information about other Xilinx® LogiCORE™ IP modules is available at the [Xilinx Intellectual Property](#) page. For information about pricing and availability of other Xilinx® LogiCORE IP modules and tools, contact your [local Xilinx sales representative](#).

Product Specification

Overview

Operation States

The IP works in three states:

- Unarmed
In the Unarmed state, the core does not monitor or report events.
- Armed Continuous
In the Armed Continuous state, the core reports every event that it detects.
- Armed One Shot
In the Armed One Shot state, the core reports the next event that it detects, then reverts to the Unarmed state (an event is either the start or the end of a partial bitstream).

Identifiers

The IP extracts and reports the following four identifiers from the start and the end of partial bitstreams. See [Partial Bitstream Preparation](#) for information on how to annotate partial bitstreams.

Static Partition ID (SP_ID)

The Static Partition identifier is an FPGA wide identifier that is used to tie a partial bitstream to a particular static bitstream¹. All partial bitstreams for a particular FPGA should have the same SP_ID.

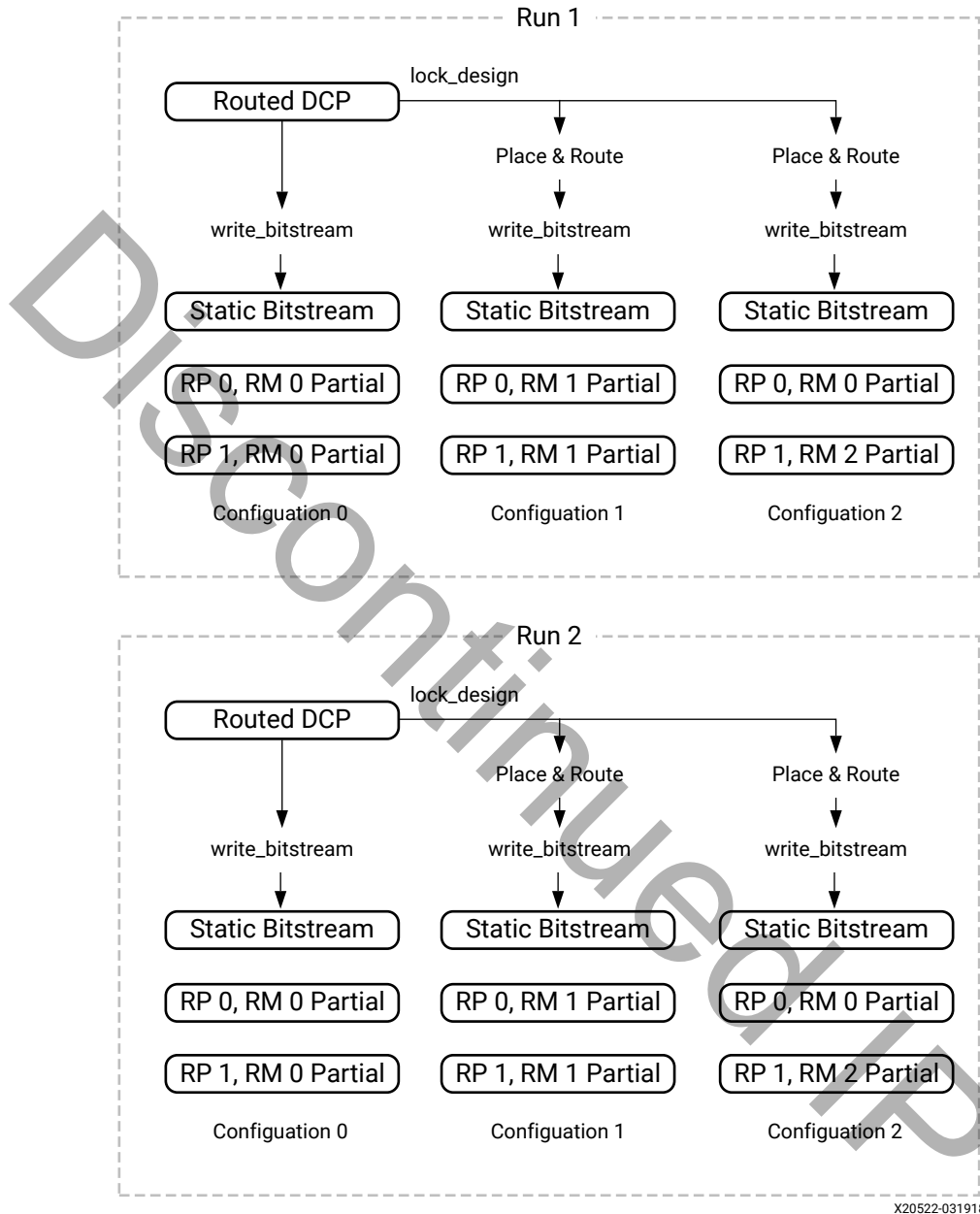
If the SP_ID of the partial bitstream does not match the SP_ID of the static bitstream then the core flags an error. Other elements in the system could use this information to block the partial bitstream from loading, as loading it could cause damage to the device.

The following figure shows two implementation runs of an example Partial Reconfiguration (PR) project where the static bitstream and partial bitstreams are generated from three implementation configurations (See *Vivado Design Suite User Guide: Partial Reconfiguration (UG909)* for a description of configurations and the PR implementation flow).

1. Partial bitstreams must be generated from the same routed design checkpoint (DCP) as the static bitstream. If they are not, device damage could occur.

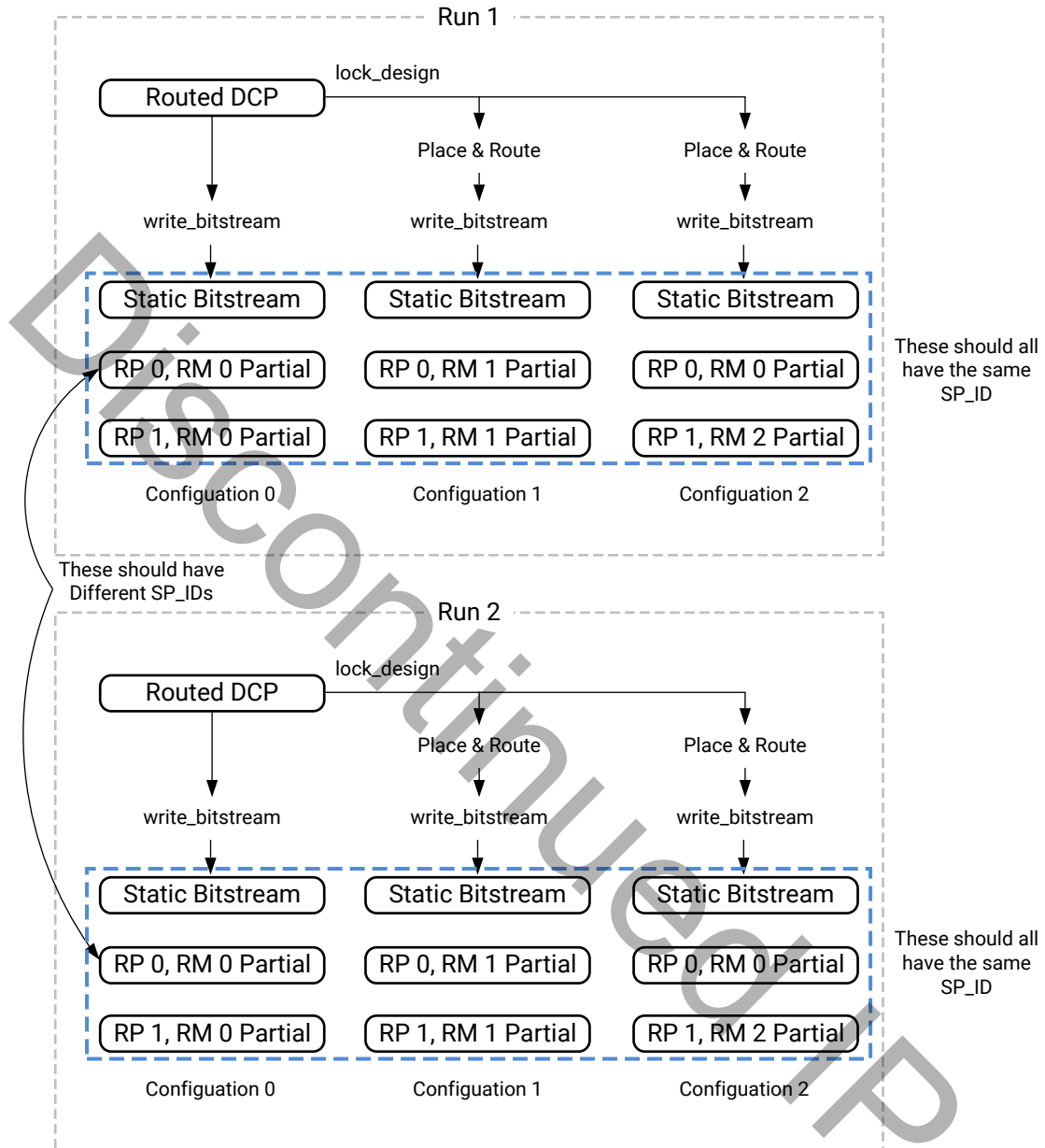
Discontinued IP

Figure 1: Example of a PR Project



To detect the error where bitstreams from both runs are mixed in an application, the SP_IDs should be set as shown in the following figure.

Figure 2: Setting SP_ID to detect errors



To provide maximum value, the SP_ID should be unique to every version of the routed static DCP. For example, the timestamp of the routed static DCP could be used.

The value of SP_ID that the IP uses for comparison can be set in three ways:

- The static bitstream can have this value set by write_bitstream so it becomes available through the USR_ACESSE2 primitive. The IP can capture this at power on.
- The value can be driven into the core using the ref_sp_id_i port.

- The value can be written into the core using the register interface. Note that the `ref_sp_id_i` port is still enabled in this case. The value used by the core is the bitwise OR of the `ref_sp_id_i` port and the register value. It is suggested that the `ref_sp_id_i` port is left unconnected, or tied to zero, if the register interface is to be used to set the value.

Reconfigurable Partition (RP_ID) and Reconfigurable Module (RM_ID) IDs

The RP_ID and RM_ID identifiers are used to match a partial bitstream in the design's datapath to a particular Reconfigurable Partition (RP) and Reconfigurable Module (RM) in the functional design description. The core does not interpret these values, and as such, any values that make sense for the application can be used.

The following guidelines may be helpful in choosing RP_ID and RM_ID values:

- Each RP should have a unique identifier.
- Each RM should have a unique identifier inside an RP. For example, a single RP shouldn't have two RMs with RM_ID = 0. However, it is ok for RP 0 to have an RM with RM_ID = 0, and RP 1 to have an RM with RM_ID = 0.

As an example, a design with two RPs with two RMs each may have the following identifiers:

Table 1: Reconfigurable Partition and Reconfigurable Module IDs

Reconfigurable Partition	Reconfigurable Module
SHIFT (RP_ID 0)	SHIFT_LEFT (RM_ID 0)
SHIFT (RP_ID 0)	SHIFT_RIGHT (RM_ID 1)
COUNT (RP_ID 1)	COUNT_DOWN (RM_ID 0)
COUNT (RP_ID 1)	COUNT_UP (RM_ID 1)

If a PR Bitstream Monitor monitoring the ICAP bus reported a bitstream with RP_ID 0 and RM_ID 1, it would be trivial to identify this as the SHIFT_RIGHT RM.

Note: If the Partial Reconfiguration Controller (PRC) IP is used, then RPs and RMs already have assigned identifiers (VSM ID and RM ID). Using these values would simplify debug. See the *Partial Reconfiguration Controller LogiCORE IP Product Guide* (PG193) for more information.

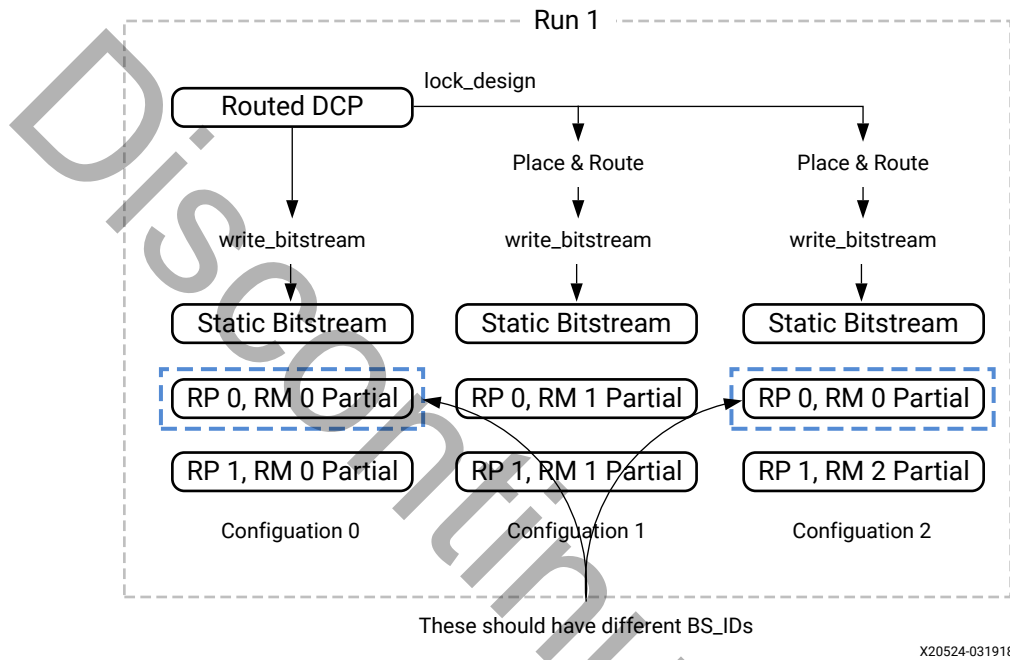
Bitstream Identifier (BS_ID)

The bitstream identifier serves two purposes:

- To differentiate between clearing and partial bitstreams on UltraScale devices
- To differentiate between partial bitstreams generated for the same RM from different implementation configurations (See *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909) for a description of configurations and the PR implementation flow.

The following figure shows an example PR project that uses three implementation configurations. RP 0 has two RMs and RP 1 has three RMs. This means one RM will get implemented twice for RP 0 (RM 0 in this example) and will have two separate bitstreams generated. These would both be compatible with the static platform, and could both be used successfully in the design. The BS_ID is intended to tell them apart.

Figure 3: Example Showing the Need for BS_ID



The identifiers for this design could be:

Table 2: IDs for Example Design

RP	RM	BS_ID
SHIFT (RP_ID 0)	SHIFT_LEFT (RM_ID 0)	0 (from configuration 0) 2 (from configuration 2)
SHIFT (RP_ID 0)	SHIFT_RIGHT (RM_ID 1)	0
COUNT (RP_ID 1)	COUNT_DOWN (RM_ID 0)	0
COUNT (RP_ID 1)	COUNT_UP (RM_ID 1)	0
COUNT (RP_ID 1)	COUNT_BY_2 (RM_ID 2)	0

The following guidelines may be helpful in choosing BS_ID values:

- Bitstream IDs should be unique within an RM, but do not have to be unique between RMs
- In UltraScale devices, the BS_ID for Clearing Bitstreams could be odd, and even for Partial Bitstreams
- Bitstream IDs could be based on the timestamp of the actual bitstream file

Error Detection

The core can detect the following errors:

- SP_ID mismatch: This is where the SP_ID extracted from a bitstream does not match the reference SP_ID in the core.
- Unexpected Error: This can be caused by the following conditions:
 - One or more of the identifiers received at the end of a bitstream do not match the identifiers received at the start of the bitstream.
 - The end of a bitstream is seen without first seeing the start of a bitstream (Arming the core part way through a bitstream does not trigger this error).
 - The start of a bitstream is seen without seeing the end of the previous bitstream.

Control and Status

The Bitstream Monitor IP provides a signal interface and an AXI4-Lite register interface to control the core. Only one of these can be enabled at a time. It also provides signal and AXI4-Lite register access to the core's status. The AXI4-Lite interface is optional, but the status signals are always present. They can be left unconnected if not required.

Performance

For full details about performance and resource use, visit the [Performance and Resource Use web page](#).

Resource Use

For full details about performance and resource use, visit the [Performance and Resource Use web page](#).

Port Descriptions

Port Names

The following table lists the port descriptions.

Table 3: Port Descriptions

Port Name	I/O	Description
clk	I	Clock.
reset/resetn	I	Reset. Active-High/Low depending on core configuration.
s_axi_ctrl_*	I,O	AXI4-Lite ¹ register interface.
arm	I	Assert to enter one of the Armed states. Deassert to enter the Unarmed state.
one_shot	I	Assert to enter the Armed One Shot state when a <code>arm</code> is asserted. Deassert to enter the Armed Continuous state when a <code>arm</code> is asserted.
armed	O	Asserted when the core has entered one of the Armed states.
armed_oneshot	O	Asserted when the core has entered the Armed One Shot state.
ref_sp_id_i	I	The reference SP_ID input. This is only enabled when the <code>HAS_REF_SP_ID_I</code> user parameter is set to TRUE. This is the value that the SP_ID extracted from a bitstream is compared against when determining if there is an SP_ID Mismatch error.
ref_sp_id_o	O	The reference SP_ID output. This is only enabled when the <code>HAS_REF_SP_ID_O</code> user parameter is set to TRUE. This signal carries the value of the reference SP_ID in the core. This can either be: <ol style="list-style-type: none"> The value captured from the <code>USR_ACESSE2</code> primitive at power on The value from <code>ref_sp_id_i</code> The value from <code>ref_sp_id_i ORd</code> with the value from the register interface
Live Information		
li_avail	O	Asserted for a clock cycle when information is available on the <code>li_*</code> ports.
li_end	O	0: The information on the <code>li_*</code> ports relates to the start of a partial bitstream. 1: The information on the <code>li_*</code> ports relates to the end of a partial bitstream. Only valid when <code>li_avail</code> is 1.
li_sp_id	O	The SP_ID extracted from the bitstream. Only valid when <code>li_avail</code> is 1.
li_rp_id	O	The RP_ID extracted from the bitstream. Only valid when <code>li_avail</code> is 1.
li_rm_id	O	The RM_ID extracted from the bitstream. Only valid when <code>li_avail</code> is 1.
li_bs_id	O	The BS_ID extracted from the bitstream. <code>li_avail</code> is 1.
li_err_sp_id_mismatch	O	valid when The SP_ID value extracted from the bitstream does not match the reference SP_ID in the core. Only valid when <code>li_avail</code> is 1.

Table 3: Port Descriptions (cont'd)

Port Name	I/O	Description
li_err_abort	O	The core has received an abort, either from the <code>protocol_abort</code> input or the registers. Only valid when <code>li_avail</code> is 1.
li_err_unexpected	O	The core has detected an unexpected error (see Error Detection for more information). Only valid when <code>li_avail</code> is 1.
Historic Information		
hi_avail	O	Data is available in the historic information buffer.
hi_end	O	0: The information on the <code>hi_*</code> ports relates to the start of a partial bitstream. 1: The information on the <code>hi_*</code> ports relates to the end of a partial bitstream. Only valid when <code>hi_avail</code> is 1.
hi_sp_id	O	The SP_ID extracted from the bitstream. Only valid when <code>hi_avail</code> is 1.
hi_rp_id	O	The RP_ID extracted from the bitstream. Only valid when <code>hi_avail</code> is 1.
hi_rm_id	O	The RM_ID extracted from the bitstream. Only valid when <code>hi_avail</code> is 1.
hi_bs_id	O	The BS_ID extracted from the bitstream. Only valid when <code>hi_avail</code> is 1.
hi_err_sp_id_mismatch	O	The SP_ID value extracted from the bitstream does not match the reference SP_ID in the core. Only valid when <code>hi_avail</code> is 1.
hi_err_abort	O	The core has received an abort, either from the <code>protocol_abort</code> input or the registers. Only valid when <code>hi_avail</code> is 1.
hi_err_unexpected	O	The core has detected an unexpected error (see Error Detection for more information). Only valid when <code>hi_avail</code> is 1.
hi_read	I	Assert for 1 clock cycle to read one entry from the historic information buffer.
General Protocol Signals		
protocol_clock	I	The clock for the interface being monitored. Only enabled when clock domain crossing is requested.
protocol_clock_out	O	The clock used to extract data from the interface being monitored. Only enabled when clock domain crossing is requested, the core is monitoring the USR_ACCESS protocol, and the core instantiates the USR_ACCESS2 primitive. This signal is provided in order to synchronise <code>protocol_abort</code> .
protocol_reset/ protocol_resetn	I	Reset for the data extraction logic. Active high/low depending on core configuration. Enabled when the datapath protocol is AXI4MM, AXI4-Lite or Generic.
protocol_abort	I	Assert to abort the monitoring of a bitstream. The core stays in the armed state.

Table 3: Port Descriptions (cont'd)

Port Name	I/O	Description
ICAP Protocol Signals		
icap_csib	I	ICAP CSIB control signal. Enabled when the datapath protocol is ICAP.
icap_rdwrb	I	ICAP RDWRB control signal. Enabled when the datapath protocol is ICAP.
icap_i	I	ICAP data. Enabled when the datapath protocol is ICAP.
USR_ACCESS Protocol Signals		
usr_access_data	I	USR_ACCESS data. Enabled when the datapath protocol is USR_ACCESS and the core does not instantiate the USR_ACCESE2 primitive. This port is also enabled when the datapath protocol is not USR_ACCESS, HAS_REF_SP_ID_I is disabled, and the core does not instantiate the USR_ACCESE2 primitive. In this case, the port should be attached to the USR_ACCESE2 primitive that exists elsewhere in the design so that the core can get a value for SP_ID at power on.
usr_access_datavalid	I	USR_ACCESS data is valid. Enabled when the datapath protocol is USR_ACCESS and the core does not instantiate the USR_ACCESE2 primitive. This port is also enabled when the datapath protocol is not USR_ACCESS, HAS_REF_SP_ID_I is disabled, and the core does not instantiate the USR_ACCESE2 primitive. In this case, the port should be attached to the USR_ACCESE2 primitive that exists elsewhere in the design so that the core can get a value for SP_ID at power on.
Generic Protocol Signals		
generic_datavalid	I	Active high signal to say that the generic_data input has valid data on this clock cycle. Enabled when the datapath protocol is Generic.
generic_data	I	Enabled when the datapath protocol is Generic.
AXI4MM and AXI4-Lite Protocol Signals		
s_axi_*	I, O	The AXI monitor interface ¹ Enabled when the datapath protocol is AXI4MM or AXI4-Lite.

Notes:

- For a description of AXI4, AXI4-Lite and AXI Stream signals, see the *Vivado Design Suite AXI Reference Guide* (UG1037).

Register Space

The PR Bitstream Monitor's register space is summarized in the following table:

Table 4: Register Names

Address Space Offset	Name	Description
00h	ARM	Arm/Disarm the core
04h	ABORT	Abort the monitoring for any ongoing bitstream
08h	REF_SP_ID	Set the Reference SP_ID
10h	ARMED	Get the state of the core (Unarmed, Armed Continuous, Armed One Shot)
14h	HI_STATUS	The first status entry in the historical information status buffer
18h	HI_SP_ID	The SP_ID entry in the historical information status buffer
1Ch	HI_RP_ID	The RP_ID entry in the historical information status buffer
20h	HI_RM_ID	The RM_ID entry in the historical information status buffer
24h	HI_BS_ID	The BS_ID entry in the historical information status buffer

Arm Control Register

The Arm Control Register is used to arm and disarm the core. All fields are R/W and reset to 0x0. Note that the values read back do not represent the state of the core, only what has been requested. The Armed Status Register must be used to get the state of the core.

Table 5: Arm Control Register

Bit	Name	Description
31:2	Reserved	Reserved
1	ONE_SHOT	1: Arm One Shot 0: Arm Continuous These bits only have an effect when ARM is 1
0	ARM	1: Arm the core 0: Disarm the core

Abort Control Register

The Abort Control Register is write only and resets to 0x0.

Table 6: Abort Control Register

Bits	Name	Description
31:1	Reserved	Reserved
0	ABORT	1: Abort the monitoring 0: No effect

Reference SP_ID Register

The Reference SP_ID Register is used to set the SP_ID value that the core uses to generate `sp_id_mismatch` errors.

If the `HAS_REF_SP_ID_I` User Parameter is set to `FALSE`, then this register is read only. In this case, the value read is the value set by the `USR_ACCESSE2` primitive.

If the `HAS_REF_SP_ID_I` User Parameter is set to `TRUE`, then this register can be written. However, the value written to this register is bitwise ORd with the value on the `ref_sp_id_i` port before being used in the core. The value read from this register is the bitwise ORd version.

It is recommended that the `ref_sp_id_i` port is left unconnected, or connected to all-zero, if this register is going to be used.

Table 7: SP_ID Register Bit Definition

Bits	Name	Description
31:W	Reserved	Reserved
W-1:0	REF_SP_ID	The reference SP_ID value

Note: W is set by the `STS_SP_ID_WIDTH` User Parameter.

Armed Status Register

The Armed Status Register is read only and resets to 0x0.

Table 8: Armed Status Register Bit Definitions

Bits	Name	Description
31:2	Reserved	Reserved
1	ONE_SHOT	1: The core is Armed One Shot 0: The core is Armed Continuous These bits only have meaning when ARMED is 1.
0	ARMED	1: The core is Armed 0: The core is Disarmed

Historical Information Status Register

The Historical Information Status Register is read only, and all fields reset to 0. Reading from this register updates the values in the following registers, and removes the entry from the Historical Information buffer:

- HI_SP_ID
- HI_RP_ID

- HI_RM_ID
- HI_BS_ID

Table 9: Historical Information Status Register Bit Definitions

Bits	Name	Description
31:5	Reserved	Reserved
4	HI_ERR_SP_ID_MISMATCH	1: The SP_ID value extracted from the bitstream did not match the reference SP_ID in the core. 0: The SP_ID value extracted from the bitstream matched the reference SP_ID in the core.
3	HI_ERR_UNEXPECTED	1: The core detected an unexpected error. 0: The core did not detect an unexpected error. (see Error Detection for more information).
2	HI_ERR_ABORT	1: The core received an abort, either from the <code>protocol_abort</code> input or the registers. 0: The core did not receive an abort.
1	HI_END	1: The information on the HI_* registers relates to the end of a partial bitstream. 0: The information in the HI_* registers relates to the start of a partial bitstream.
0	HI_AVAIL	1: Data is available in the historic information buffer. 0: No data available.

Historical Information SP_ID Register

The Historical Information SP_ID Register is read only, and all fields reset to 0.

Table 10: Historical Information SP_ID Bit Definitions

Bits	Name	Description
31:W	Reserved	Reserved
W-1:0	HI_SP_ID	The SP_ID extracted from the bitstream

Note: W is set by the STS_SP_ID_WIDTH User Parameter.

Historical Information RP_ID Register

The Historical Information RP_ID Register is read only, and all fields reset to 0.

Table 11: Historical Information RP_ID Bit Definitions

Bits	Name	Description
31:W	Reserved	Reserved
W-1:0	HI_RP_ID	The RP_ID extracted from the bitstream

Note: W is set by the STS_RP_ID_WIDTH User Parameter.

Historical Information RM_ID Register

The Historical Information RM_ID Register is read only, and all fields reset to 0.

Table 12: Historical Information RM_ID Bit Definitions

Bits	Name	Description
31:W	Reserved	Reserved
W-1:0	HI_RM_ID	The RM_ID extracted from the bitstream

Note: W is set by the STS_RM_ID_WIDTH User Parameter.

Historical Information BS_ID Register

The Historical Information BS_ID Register is read only, and all fields reset to 0.

Table 13: Historical Information BS_ID Bit Definitions

Bits	Name	Description
31:W	Reserved	Reserved
W-1:0	HI_BS_ID	The BS_ID extracted from the bitstream

Note: W is set by the STS_BS_ID_WIDTH User Parameter.

Designing with the Core

USR_ACCESS Primitive

The following configurations of the PR Bitstream Monitor core require information from the `USR_ACESSE2` primitive:

- When the datapath protocol is set to `USR_ACCESS`
- When the reference `SP_ID` is to be read from the static bitstream (`HAS_REF_SP_ID_I = FALSE`)

The `USR_ACESSE2` primitive can only be instantiated once in a design, so the IP offers the option to instantiate it or not. If the core instantiates it, no other core in the system can do so. If the core does not instantiate it, ports are made available on the core boundary to import the required information.

Bitstream Growth

Annotating a bitstream with identifiers for use with the PR Bitstream Monitor core adds 29 words (116 bytes) to the bitstream size. This may require changes to the addresses where the bitstreams are stored. Any system component that must know the address and/or size of a partial bitstream (for example, the Partial Reconfiguration Controller) has to be updated with the new addresses and/or sizes. It is useful to add the identifiers to the partial bitstreams during development, even if the Bitstream Monitor is not being used in a particular implementation run.

Clocking

The core has one or two clock inputs, depending on the configuration.

- `clk`: The main core clock
- `protocol_clock`: Used to clock the protocol decode logic when `DP_HAS_CDC` is set to `TRUE`

In cases where the core instantiates the `USR_ACESSE2` primitive, decodes the `USR_ACCESS` protocol, and has `DP_HAS_CDC` set to `FALSE`, carefully ensure that the core's `clk` input is identical to the configuration clock that is output from the `USR_ACESSE2` primitive. If not, data from the `USR_ACESSE2` primitive may be sampled incorrectly. If `DP_HAS_CDC` is set to `TRUE`, the `protocol_clock_out` output is enabled which provides access to the `USR_ACESSE2` primitive's `CFGCLK` output. This should be used to synchronise the generation of the `protocol_abort` input.

Discontinued IP

Design Flow Steps

This section 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](#))
- *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
- *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
- *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))

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](#)) 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](#)) and the *Vivado Design Suite User Guide: Getting Started* ([UG910](#)).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

Global Options Tab

The parameters in the Global Options tab are shown in the following figure.

Figure 4: Global Options Tab

Reset Active Level

This option sets the active level of the core reset.

- 0: The reset is active-Low and the `resetn` signal is enabled
- 1: The reset is active-High and the `reset` signal is enabled

Instantiate the USR_ACCESS Primitive

This option controls whether the USR_ACESSE2 primitive is instantiated by the core.

- TRUE: The USR_ACESSE2 primitive is instantiated
- FALSE: The USR_ACESSE2 primitive is not instantiated

Control Interface Type

This option sets the control interface type. Valid values are:

- Signal
- AXI4Lite

Control Interface Address Width

This option sets the address width used by the AXI4-Lite register interface. Valid values are 7 to 64 inclusive.

Has User Settable Reference SP_ID Input

This option is used to decide if the core will get the reference value of SP_ID from the `USR_ACCESE2` primitive, or from core inputs.

- TRUE: The reference SP_ID will be taken from the `ref_sp_id_i` port or the register interface
- FALSE: The reference SP_ID will be taken from the `USR_ACCESE2` primitive

Has Reference SP_ID Output

This option enables or disables the `ref_sp_id_o` output.

- TRUE: Enable the output
- FALSE: Disable the output

SP_ID_WIDTH

This option sets the width of the SP_ID identifier. Valid values are 1 to 32 inclusive.

RP_ID_WIDTH

This option sets the width of the RP_ID identifier. Valid values are 1 to 32 inclusive.

RM_ID_WIDTH

This option sets the width of the RM_ID identifier. Valid values are 1 to 32 inclusive.

BS_ID_WIDTH

This option sets the width of the BS_ID identifier. Valid values are 1 to 32 inclusive.

History Buffer Implementation

This option sets the type of memory used to implement the History Buffer. Valid options are:

- Distributed RAM
- Block RAM

History Buffer Depth

This option sets the number of entries in the History Buffer. Valid values are:

- 16
- 32
- 64
- 128
- 256
- 512
- 1024
- 2048
- 4096
- 8192
- 16384
- 32768
- 65536
- 131072

History Buffer Behavior When Full

This option specifies what will happen when there's new data for the History Buffer and it is already full. Valid options are:

- Discard new data: The new data is discarded
- Discard old data: The oldest data in the buffer is discarded

Datapath Protocol

This option specifies the protocol of the bus that the core will be connected to. Valid options are:

- AXI4MM
- AXI4Lite
- ICAP
- USR_ACCESS
- GENERIC

The Generic protocol is provided as a way of attaching the monitor to protocols that aren't natively supported by the core.

Datapath Active Reset Level

This option sets the active level of the datapath's reset.

- 0: The reset is active-Low
- 1: The reset is active-High

Data Format

This options tells the core how to interpret the data. Valid options are:

- Little endian with no bitswapping
- Little endian with bitswapping
- Big endian with no bitswapping
- Big endian with bitswapping

Has CDC

This option enables or disables the core's clock domain crossing logic.

- TRUE: Enable clock domain crossing
- FALSE: Disable clock domain crossing

CDC FIFO Type

This option sets the type of memory used to implement the clock domain crossing FIFO. Valid options are:

- Distributed RAM
- Block RAM

CDC FIFO Depth

This option sets the number of entries in the clock domain crossing FIFO. Valid values are:

- 32
- 64
- 128
- 256
- 512
- 1024
- 2048
- 4096
- 8192

- 16384
- 32768
- 65536
- 131072

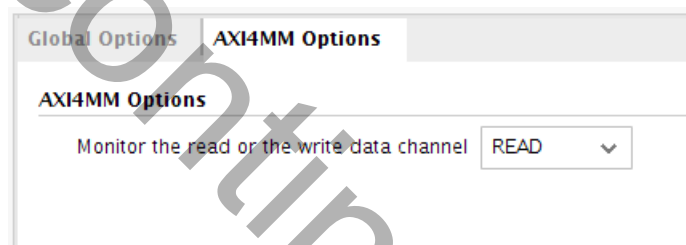
Number of CDC stages

This option sets the number of stages in the clock domain crossing synchronisers. Valid values are 2 to 8 inclusive.

AXI4MM Options Tab

The parameters in the AXI4MM Options tab are shown in the following figure and are described in this section.

Figure 5: AXI4MM Options Tab



Channel to Monitor

This option tells the core which AXI channel to monitor. Valid values are:

- READ: Data is read from the read data channel
- WRITE: Data is read from the write data channel

User Parameters

The following table shows the relationship between the fields in the Vivado® IDE and the user parameters (which can be viewed in the Tcl Console).

Table 14: User Parameters

Vivado IDE Parameter/ Value ¹	User Parameter/Value	Default Value
Reset Active Level	RESET_ACTIVE_LEVEL	0
Instantiate the USR_ACCESS Primitive	HAS_USR_ACCESS	FALSE

Table 14: User Parameters (cont'd)

Vivado IDE Parameter/ Value ¹	User Parameter/Value	Default Value
Control Interface Type	CTRL_INTERFACE_TYPE <ul style="list-style-type: none"> Signal: 0 AXI4LITE: 1 	0
Control Interface Address Width	CTRL_ADDR_WIDTH	32
Has User Settable Reference SP_ID Input	HAS_REF_SP_ID_I	FALSE
Has Reference SP_ID Output	HAS_REF_SP_ID_O	TRUE
SP_ID_WIDTH	STS_SP_ID_WIDTH	32
RP_ID_WIDTH	STS_RP_ID_WIDTH	32
RM_ID_WIDTH	STS_RM_ID_WIDTH	32
BS_ID_WIDTH	STS_BS_ID_WIDTH	32
History Buffer Implementation	STS_HIST_BUFFER_TYPE <ul style="list-style-type: none"> Block RAM : block Distributed RAM: distributed 	distributed
History Buffer Depth	STS_HIST_BUFFER_DEPTH	16
History Buffer Behavior When Full	STS_HIST_BUFFER_WHEN_FULL <ul style="list-style-type: none"> Discard Old Data : discard_old Discard New Data : discard_new 	discard_new
Datapath Protocol	DP_PROTOCOL <ul style="list-style-type: none"> AXI4MM AXI4LITE ICAP USR_ACCESS GENERIC 	AXI4MM
Datapath Active Reset Level	PROTOCOL_RESET_ACTIVE_LEVEL	0
Data Format	DP_DATA_FORMAT <ul style="list-style-type: none"> Little Endian with no Bitswapping : le_no_bs Little Endian with Bitswapping: le_bs Big Endian with no Bitswapping: be_no_bs Big Endian with Bitswapping: be_bs 	le_no_bs
Has CDC	DP_HAS_CDC	FALSE
CDC FIFO Type	DP_CDC_FIFO_DEPTH	32

Table 14: User Parameters (cont'd)

Vivado IDE Parameter/ Value ¹	User Parameter/Value	Default Value
CDC FIFO Depth	DP_CDC_FIFO_TYPE <ul style="list-style-type: none"> • Block RAM : block • Distributed RAM: distributed 	distributed
Number of CDC stages	DP_CDC_STAGES	6
Channel to Monitor	DP_AXI_CHAN_TO_MONITOR	READ

Notes:

- Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP (UG896)*.

Constraining the Core

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

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)*.

Synthesis and Implementation

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

Partial Bitstream Preparation

Note: Only BIN files generated by `write_bitstream -bin` are supported. BIT files are not supported.

Note: If the partial bitstreams are being loaded by the PRC IP then the `format_bin_for_icap` function in the PRC's API can be used to insert the identifiers. If the bitstreams are being compressed by the PRC's API then the PRC's `format_bin_for_icap` function **must** be used to add the identifiers.

To add identifiers to partial bitstreams, use the following steps:

1. Load the Bitstream Monitor's TCL API.

```
source [get_property REPOSITORY [get_ipdefs *pr_bitstream_monitor*]]/  
xilinx/pr_bitstream_monitor_v1_0/tcl/api.tcl -notrace
```

2. Add identifiers to each partial bitstream using the following command:

```
pr_bitstream_monitor_v1_0::add_identifiers
```

Mandatory switches:

`-i <file name>`: The path and name of partial's bin file

- 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

Optional switches:

-o <file name> : The path and name of the file to create. If omitted, the output file will be the input file name with a .ids extension

The following code can be used to make it easier to upgrade the core.

```
if {[pr_bitstream_monitor_v1_0::is_api_compatible pr_bitstream_monitor_v0_0]}
{pr_bitstream_monitor_v1_0::alias_api pr_bsm
}
```

is_api_compatible takes the name of the previous version of the core and returns 1 if the API from the new version is compatible with the API for the old version.

alias_api <name> imports all the API commands into a namespace called <name>.

Examples

```
if {[pr_bitstream_monitor_v1_0::is_api_compatible
pr_bitstream_monitor_v0_0]} {
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
pr_bitstream_monitor_v1_0::alias_api pr_bsm
}
set sp_id 1234
set partial "shift_left.bin"
pr_bsm::add_identifiers -sp_id $sp_id \
    -rp_id 0 \
    -rm_id 0 \
    -bs_id [file mtime $partial] \
    -i $partial
set partial "shift_right.bin"
pr_bsm::add_identifiers -sp_id $sp_id \
    -rp_id 0 \
    -rm_id 1 \
    -bs_id [file mtime $partial] \
    -i $partial
set partial "count_down.bin"
pr_bsm::add_identifiers -sp_id $sp_id \
    -rp_id 1 \
```

```
-rm_id 0 \  
-bs_id [file mtime $partial] \  
-i $partial  
set partial "count_up.bin"  
pr_bsm::add_identifiers -sp_id $sp_id \  
-rp_id 1 \  
-rm_id 1 \  
-bs_id [file mtime $partial] \  
-i $partial
```

Discontinued IP

Upgrading

This appendix is not applicable for the first release of the core.

Discontinued IP

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 core, the [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. The [Xilinx Community Forums](#) are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

Documentation

This product guide is the main document associated with the core. 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 Xilinx Documentation Navigator from the [Downloads page](#). For more information about this tool and the features available, open the online help after installation.

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

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 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 Core

AR [70696](#).

Technical Support

Xilinx provides technical support on the [Xilinx Community Forums](#) 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 ask questions, navigate to the [Xilinx Community Forums](#).

Debug Tools

There are many tools available to address Partial Reconfiguration Bitstream Monitor design issues. It is important to know which tools are useful for debugging various situations.

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 to interact with the logic debug LogiCORE 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](#)).

Discontinued IP

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, select **Start** → **All Programs** → **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on DocNav, see the [Documentation Navigator](#) page on the Xilinx website.

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: Getting Started \(UG910\)](#)
 4. [Vivado Design Suite User Guide: Logic Simulation \(UG900\)](#)
 5. [Vivado Design Suite AXI Reference Guide \(UG1037\)](#)
 6. [Vivado Design Suite User Guide: Partial Reconfiguration \(UG909\)](#)
 7. [Partial Reconfiguration Controller LogiCORE IP Product Guide \(PG193\)](#)
-

Training Resources

1. [Vivado Design Suite Hands-on Introductory Workshop](#)
 2. [Vivado Design Suite Tool Flow](#)
 3. [Xilinx Partial Reconfiguration Tools & Techniques](#)
-

Revision History

The following table shows the revision history for this document.

The following table shows the revision history for this document.

Section	Revision Summary
12/05/2018 Version 2018.3	
Operation States	Removed duplicated content.
04/04/2018 Version 2018.1	
Initial Xilinx® release.	N/A

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

© Copyright 2018 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, ISE, Kintex, Spartan, Versal, 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.