Fast Partial Reconfiguration Over PCI Express

XAPP1338 (v1.0) March 11, 2019

Summary

Designers often want to minimize the amount of time it takes to load a partial bitstream over a PCI Express® interface. For high-speed or time-sensitive applications that must partially reconfigure quickly, a PCIe®-based DMA can reduce load times, and it can make loading a partial bitstream up to 250 times as fast as a typical load over the embedded media configuration access port (MCAP) path. This solutions shows a simple example (based on AXI4 protocols) that can be customized and dropped into a new or existing design.

The example design targets two different Xilinx® evaluation boards:

- Kintex® UltraScale+™ on the KCU116 Evaluation Board
- Virtex® UltraScale+™ on the VCU118 Evaluation Board

For more information, see the following guides:

- *DMA/Bridge Subsystem for PCI Express Product Guide* (PG195)
- *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909)

Download the reference design files for this application note from the Xilinx® website. For detailed information about the design files, see Reference Design.

Introduction

UltraScale™ and UltraScale+™ devices support Partial Reconfiguration (PR), which is the ability to dynamically change the configuration of a portion of the device, while the rest of the device continues to operate normally. Most configuration ports are permitted for delivery of partial bitstreams, so users have a great deal of flexibility as they build their system requirements. For systems based on PCI Express® (PCIe®), users can use this established connection to store and deliver partial bitstreams.

One Endpoint per device has a dedicated connection to the FPGA's configuration engine via the MCAP. This connection uses resources efficiently, but the bandwidth of PCIe data partial bitstream delivery through the MCAP is limited to one DWORD configuration writes, which can give a bandwidth of 3-6 MB/s in typical systems. Most systems only send one configuration at a time, and because configuration writes are non-posted, a second configuration write will not be sent until the completion from the preceding write is received. These restrictions lead to very low PCIe bandwidth compared to what is achievable with the PCIe protocol.

The fastest possible interface to the configuration engine for PCIe systems is through the internal configuration access port (ICAP). This SelectMAP-style interface can support 32-bit wide bitstream data at 200 MHz (800 MB/s) for monolithic devices and 125MHz (500 MB/s) for devices using stacked silicon interconnect (SSI) technology. This application note shows a basic design that connects a PCIe direct memory access (DMA) IP to the ICAP, providing the maximum throughput, allowing users to partially reconfigure as fast as the silicon allows.

Reference Design

This Partial Reconfiguration project has been created with the Vivado® 2018.1 integrated design environment (IDE) and validated with the Vivado 2018.3 IDE. Two versions are available — the same basic design can target either the VCU118 or KCU116 development platforms. The only differences between the two versions are physical constraints (such as pin locations and Pblocks) and the frequency of clock driving the ICAP. While created specifically for UltraScale+™ devices, the same concepts can be applied to UltraScale™ devices. The only difference is the requirement for delivering clearing bitstreams for UltraScale prior to loading the next partial bitstream.

The design archives contain the base design ready for compilation in the Vivado IDE. Bitstreams and dual QSPI flash images are available in the `Bitstreams_VCU118` and `Bitstreams_KCU116` folders for immediate hardware testing. Please remember that any newly compiled version will create full and partial bitstreams that are incompatible with the bitstreams supplied in this design archive — always keep bitstream versions in sync.

Download the reference design files for this application note from the Xilinx® website.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Xilinx
Target devices	• XCVU9P-FLGA2104-2L • XCKU5P-FFVB676-2
Source code provided?	N
Source code format (if provided)	N/A
Design uses code or IP from existing reference design, application note, third-party, or Vivado software? If yes, list.	N
Simulation	
Functional simulation performed	N
Timing simulation performed?	N
Test bench provided for functional and timing simulation?	N
Test bench format	N/A
Simulator software and version	N/A
SPICE/IBIS simulations	N

Table 1: **Reference Design Matrix** *(cont'd)*

Parameter	Description
Implementation	
Synthesis software tools/versions used	Vivado synthesis feature
Implementation software tool(s) and version	Vivado implementation feature
Static timing analysis performed?	N
Hardware Verification	
Hardware verified?	Y
Platforms used for verification	• VCU118 • KCU116

Setup

Create the sample design project within the Vivado 2018.1 (or newer) IDE.

1. In the Vivado IDE, start a new session in GUI or Tcl mode.
2. From the command prompt, set one of the following boards to be the target:

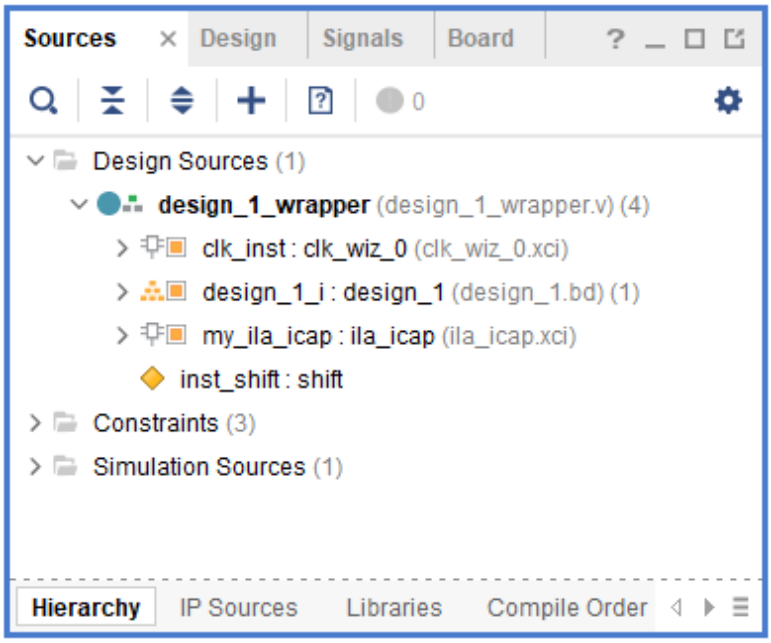
   ```
   set board "vcu118"
   set board "kcu116"
   ```

   If you do not set the target board, the KCU116 board is used by default. This option can also be set directly in the creation script.
3. Launch the creation script:

   ```
   source PR_over_PCIe_project.tcl
   ```

   When this script is completed, the full project will be ready to synthesize and implement.
4. If the script was launched in Tcl mode, open the Vivado IDE to see the full project:

   ```
   start_gui
   ```
5. Expand the design hierarchy in the Sources window to see the overall structure of the design.



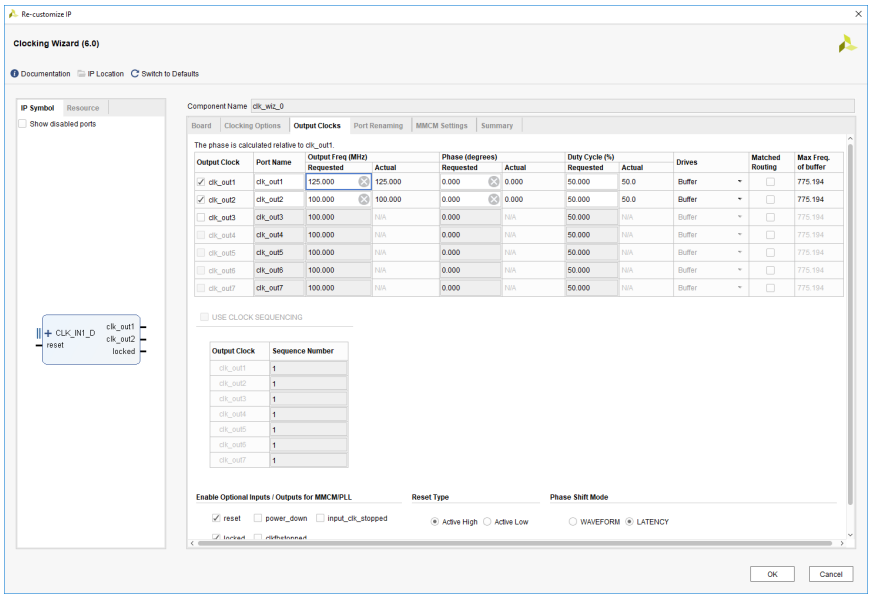
***Note*:** The diamond icon next to `inst_shift` indicates that it is a Reconfigurable Partition.

This design has a simple reconfigurable function of shifting LEDs on the demo board. This part of the design is kept simple so it compiles quickly and focuses the attention on the bitstream delivery details. The sample design is delivered with two Reconfigurable Modules:

- shift_left
- shift_right

Set the Frequency of the ICAP Clock

1. To open the customization GUI, double-click **clk_inst**.
2. On the Output Clocks tab, note the frequency of `clk_out1` for a particular board:
   - **KCU116:** 200 MHz
   - **VCU118:** 125 MHz



These represent the maximum frequency for clocking the ICAP accessing the local SLR or the entire SSI device, respectively.

Block Diagrams

Figure 1: **Logical Block Diagram**

The following block diagram shows what is going to be implemented.

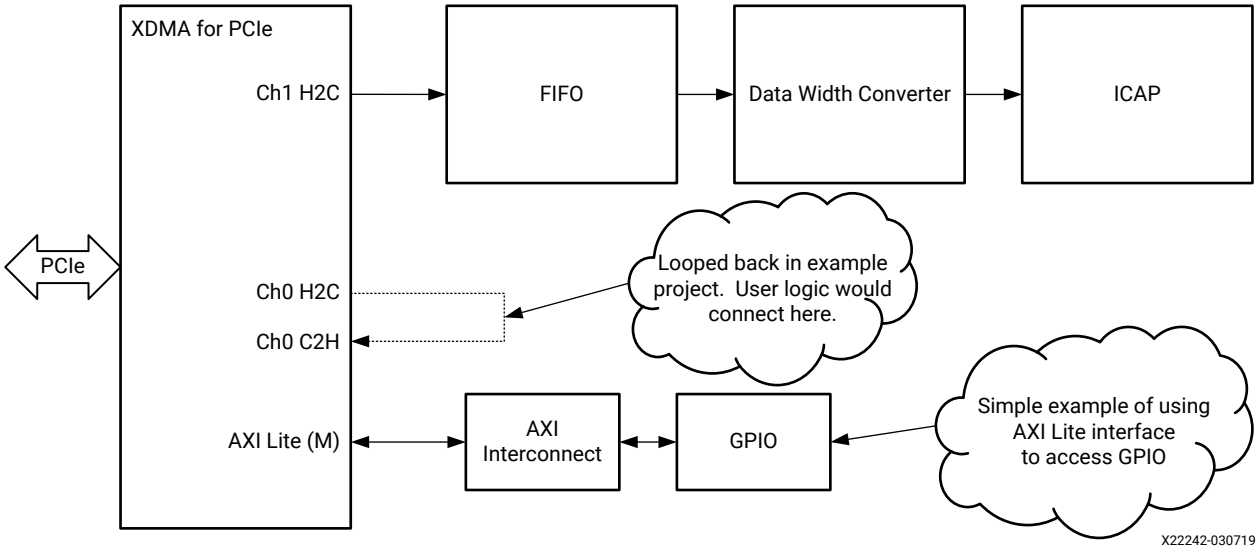
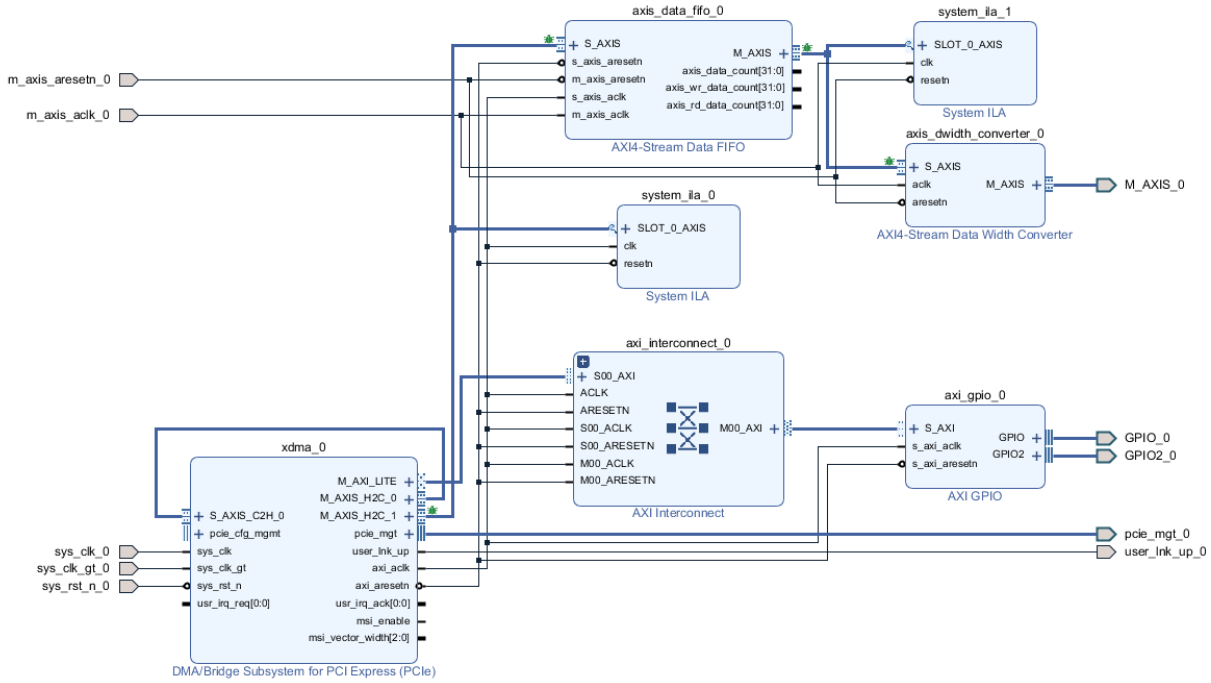


Figure 2: **PCIe® Subsystem Block Diagram**

To see the inter-processor interrupt (IPI) block diagram in the tool, open **design_1**.



This block diagram inserts a Xilinx DMA PCIe IP core with two AXI4-Stream receive channels and one AXI4-Stream transmit channel. Any number of AXI4-Stream channels can be turned on. However, one of the receive channels must be used to receive the bitstream from the PCIe host, and then transfer that bitstream to the ICAP. In this case, channel 1 is the designated receive channel for the ICAP. In this example, channel 0 receive is looped around to channel 0 transmit, but a real application would have these channels interfacing to the user logic.

The DMA channel 1 path goes to an asynchronous FIFO to make it easy to cross clock domains from the PCIe domain to the ICAP domain. The maximum frequency of the ICAP clock domain will depend on the device selected, and it can be found in the device data sheet.

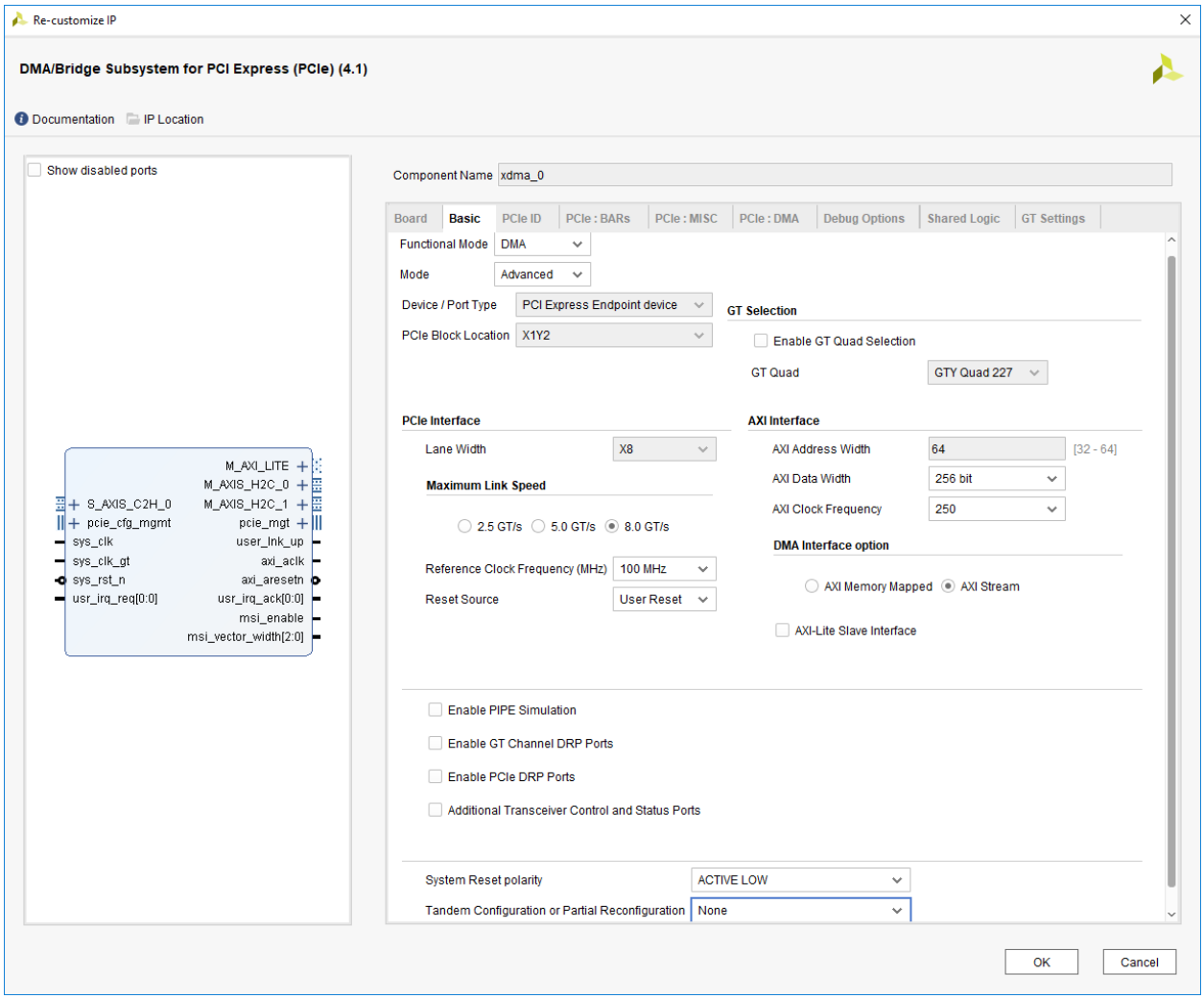
From the domain crossing FIFO, the AXI4-Stream goes to an AXI4-Stream data width converter. The size of the AXI4-Stream path from the DMA interface will be dependent on the PCIe link width and speed. The ICAP width is fixed at 32 bits for all devices.

Integrated Logic Analyzer (ILA) cores have been inserted throughout the data path to be able to monitor activity during operation. These ILAs can be removed and are not necessary for normal operation.

XDMA PCIe IP

Figure 3: **Customization of the XDMA PCIe IP, basic features**

To see the PCIe customization, when viewing the block diagram, double-click **xdma_0**.



This sample design has been set up to use a Gen3 x8 IP core, but other PCIe widths and speeds can be used. A Gen1 x1 system can typically achieve 400 MB/s transfer rates that will not saturate the ICAP interface, but can still be used and will be much faster than the MCAP path. All other link widths and speeds should saturate the ICAP interface, which maximizes configuration performance.

Under DMA Interface options, note that *AXI Stream* is selected.

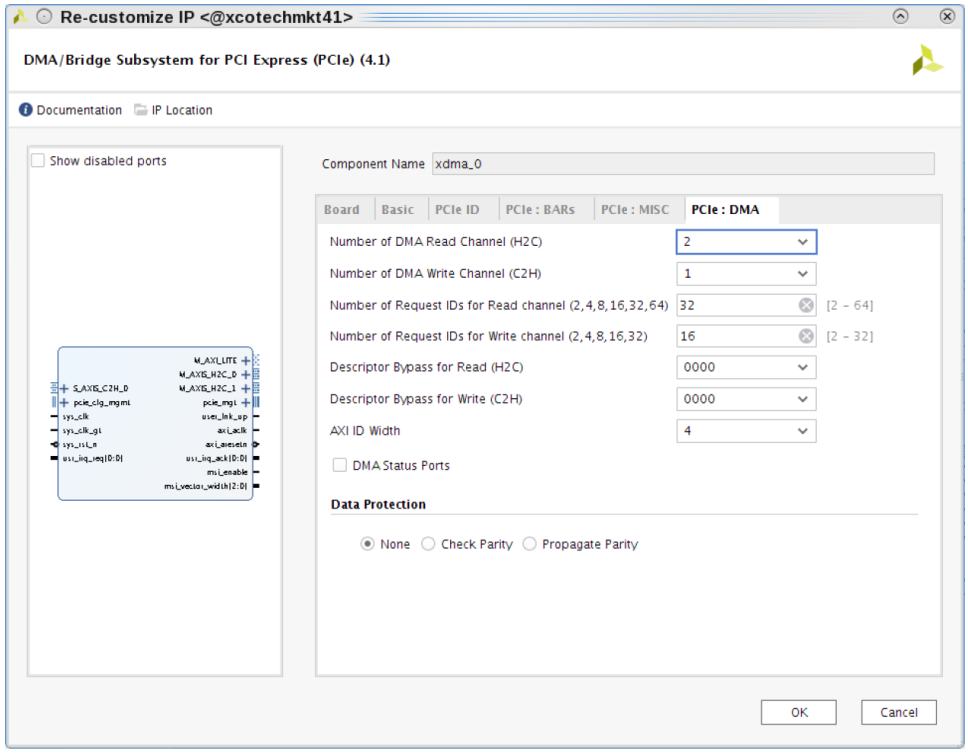
Any PCIe site on the target device can be used, but the most efficient location will be the one next to the ICAP site, which reduces routing delays.

In the Basic tab shown above, the Tandem Configuration or Partial Reconfiguration field is set to *None*. This design does not utilize Tandem Configuration to meet 120 ms enumeration during the initial device configuration. However, it *could*, because this feature is independent of what the fast partial reconfiguration over PCIe will require.

Select **PCIe : DMA**.

***Note*:** Do *not* select **PR over PCIe** in this drop-down, because that selection enables the MCAP interface for bitstream delivery, which is not used in this application.

Figure 4: **Customization of the XDMA PCIe IP, DMA features**



In this tab, see that the value for Number of DMA Read Channel (H2C) has been increased, and 2 is selected. On this tab, H2C means Host to Card, which is how the bitstream will move from the host to the PCIe block and eventually the ICAP. In this design, the channel is dedicated to the ICAP, but could be mixed to be used with other user application logic that has been added.

IMPORTANT! *It is important for developers to understand that one of the disadvantages of using the DMA for PCIe is that the configuration of the FPGA is now open to anyone using this host.*

In most systems, PCIe configuration packets used to program (via the MCAP and configuration transactions) typically require root access to send. While it is beyond the scope of this application note to implement security checks, there are several options available.

First, the ICAP itself has built-in checks to only accept bitstreams that are properly formatted with the correct device information. Also, this design does not implement readback, so the bitstream cannot be read back over the PCIe link.

In terms of protecting the FPGA from being downloaded with unintended bitstreams, you could do the following:

- Add a control register with a *secret* value that has to be set before data can be sent to the ICAP.
- Use built-in device encryption techniques.

The partial bitstream delivery path in this design starts at the PCIe IP core, passes through the FIFO and data width converter, then out of the block diagram as 32-bit AXI4-Stream data.

To see M_AXIS_0_tdata connect to the input port of the ICAP (which completes the path), open **design_1_wrapper.v**.

Compile the Reference Design

1. In the Flow Navigator, click **Run Implementation** to pull the design all the way through place and route, which will:
   - Run synthesis of all IP and sub-modules
   - Implement the parent configuration
   - Implement the child configuration using the Partial Reconfiguration project flow
2. When implementation completes, a dialog box opens. Click **Cancel** (or **Open Implemented Design**).

Note: Do *not* run bitstream generation here, because the design properties are not set up for flash programming of the full bitstream or PCIe-based delivery of the partial bitstreams. Different options are required for each and this scenario is not yet supported within Vivado project mode.

3. In the Tcl Console, to create all full and partial bit files, source this script:

   ```
   source create_all_bitstreams.tcl
   ```

   This creates the full device bitstream (with compression enabled) for the parent configuration with the required options for dual QSPI programming:
   - **CONFIG_MODE:** SPIx8
   - **SPI_BUSWIDTH:** 8
   - **CONFIGRATE:** 51.0

   The script will also generate partial bitstreams for the shift_right and shift_left Reconfigurable Modules with the required CONFIG_MODE of S_SELECTMAP32 needed for the ICAP. Bitstream compression is disabled to ensure consistent sizes for all partial bit files.
4. With these three bitstreams created and placed in the `Bitstreams` folder, the final step is to generate:
   - `.mcs` files for dual QSPI flash programming
   - `.bin` files for partial reconfiguration programming over PCIe
5. In the Tcl Console, to create all the required programming files, source this script:

   ```
   source create_bin_and_prom.tcl
   ```

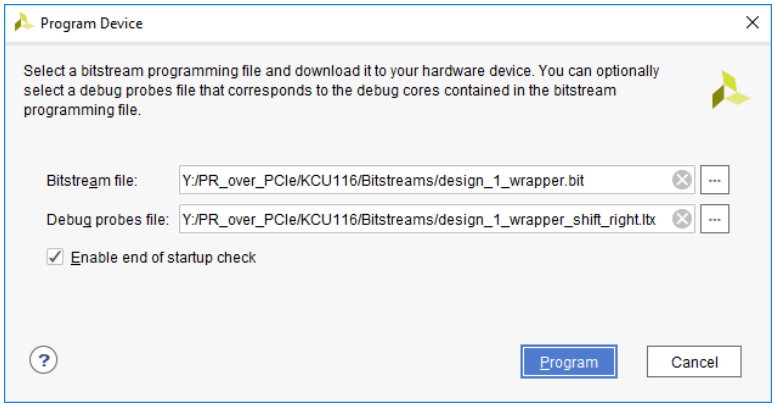
   This formats each partial bitstream for 32-bit SelectMAP delivery, and formats the full device bitstream for dual QSPI configuration.

Configure the Device via JTAG

The initial configuration of the device can be achieved via JTAG or dual QSPI. To configure the device via JTAG, perform the following steps.

1. If necessary, launch the Vivado IDE, and select **Open Hardware Manager**.

2. Connect to the target board.
3. Right-click the device (KU5P or VU9P), and select **Program Device**, which opens a dialog box.



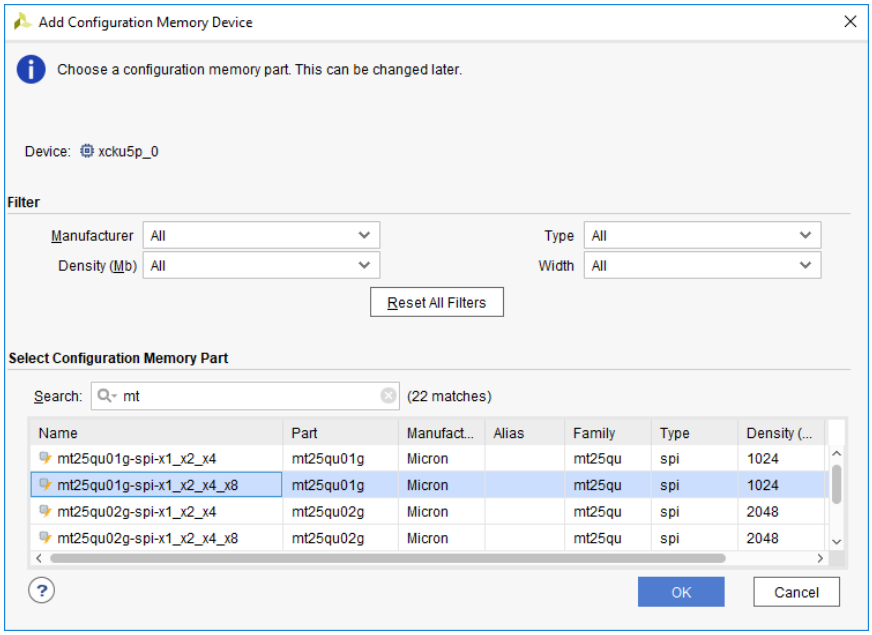
4. For Bitstream file, select **design_1_wrapper.bit** from the `Bitstreams` folder.
5. For Debug probes file, select **design_1_wrapper_shift_right.ltx**.
6. Click **Program**.

The result is that four LEDs shift to the right after DONE goes high.

Configure the Device via Dual QSPI

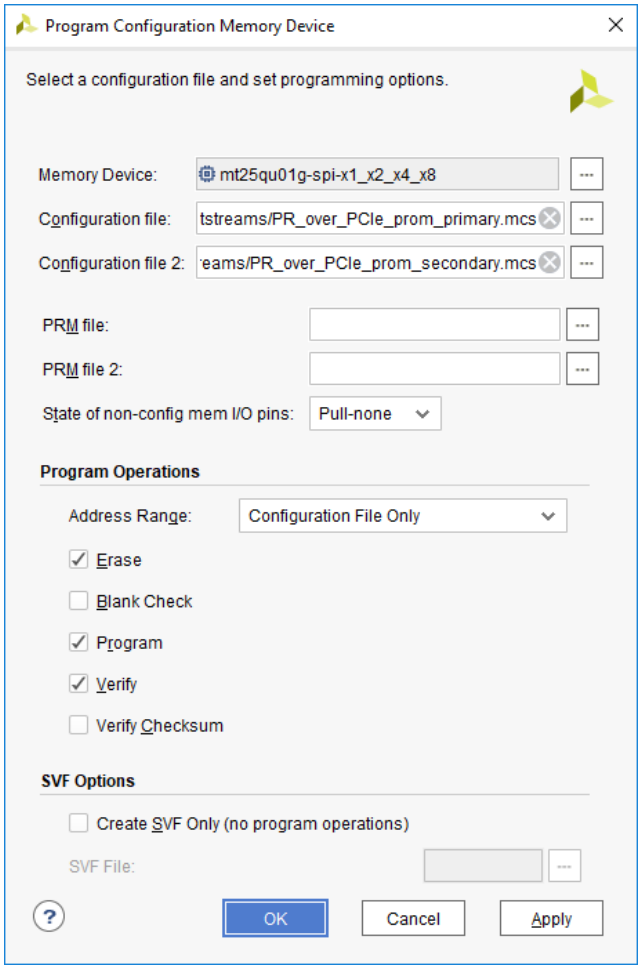
A prom image of the design can be loaded into the local QSPI flash to be automatically delivered upon power-up of the board.

1. Right-click the device (KU5P or VU9P), and select **Add Configuration Memory Device**.
2. From the list of configuration memory parts, select **mt25qu01g-spi-x1_x2_x4_x8**. You must select this part (which includes `_x8`), because the PROM image is set for dual QSPI.



3. Click **OK**, and then click **OK** again when asked if you want to program this device now.

4. When asked if you want to program the device now, click **OK**, which opens Program Configuration Memory Device.



5. For Configuration file, select `PR_over_PCIe_prom_primary.mcs`.
6. For Configuration file *2*, select `PR_over_PCIe_prom_secondary.mcs`.
7. To erase the program from the flash target, click **OK**.
8. After this completes, ensure the `Mode` pins are set to Master SPI mode before pushing PROG or re-powering the board.
9. Ensure the following values are set to 0:
   - **KCU116:** Bit 6 on SW21
   - **VCU118:** Bit 2 on SW16

Connect to PCIe Host and Partially Reconfigure the FPGA

Transferring the partial bitstream from the host to the FPGA is easily accomplished using the Xilinx DMA reference driver, available here: AR#65444.

The reference driver is available for both Windows and Linux and contains example applications that make it easy to transfer data from the host to the FPGA.

Follow the instructions within the answer record to install the driver. In this application note, Linux is used to accomplish the transfer, but Windows can be used as well.

Once the driver is installed, use the **dma_to_device** program to transfer the partial bitstream from the host to the ICAP. An example command is shown below:

```
./dma_to_device -d /dev/xdma/card0/h2c1 -s 475956 -c 1 -f ./
inst_shift_shift_left_partial.bin
```

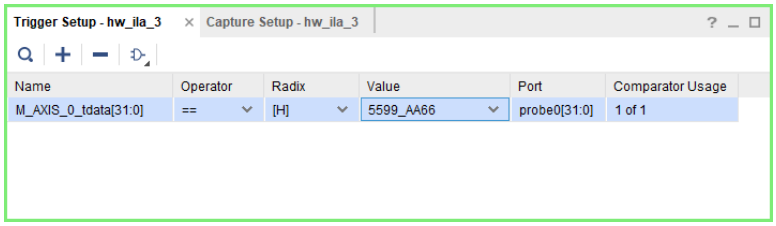
- `-d`: Character device for channel one. This location may vary so check that this path is correct for the specific Linux environment.
- `-s`: Size of the partial bin file in bytes.
- `-c`: Number of times to send the bin file. In this case, one time is all that is needed.
- `-f`: Actual partial bin file to be sent.

Different bin files can be sent using this same command, but be sure to update the size and the file location before executing the command.

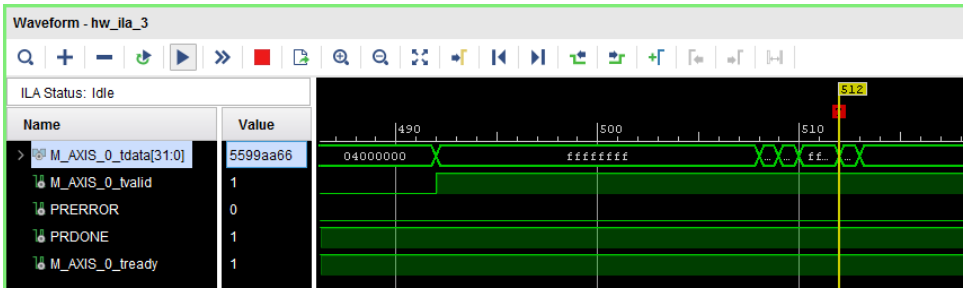
Use Debug Cores to Monitor Activity

While the device is undergoing partial reconfiguration, the data stream can be monitored to confirm bitstream formatting is correct. Multiple ILA debug cores have been inserted into this sample design, but the most straightforward one to use sits in the top level alongside the ICAP primitive.

1. Open the Vivado® Design Suite hardware manager feature, and connect to the target board.
2. To see the debug cores, refresh the device.
3. In one of the ILA core windows, click **Specify the probes file links** to find `Bitstreams/design_1_wrapper_shift_right.ltx`.
4. In `hw_ila_3`, click **+** to add probes in the Trigger Setup window.
5. Select `M_AXIS_0_tdata[31:0]`, and click **OK**.
6. For Radix, select **[H]** (hexadecimal).
7. For Value, set `5599_AA66` to watch for the bitstream sync word. This is the bit-swapped version that the ICAP expects to set the configuration engine to programming mode.



8. Arm the trigger, and then partially reconfigure the device (as described in the previous section). Data should be captured as soon as the partial bitstream is sent.
9. In the captured data, you will see the beginning of the bitstream delivery, not long after `tvalid` has risen high.



The AXI4 bus can be monitored for its activity by using the ILA core instances hw_ila_1 and hw_ila_2. These cores can be found within the block diagram. They monitor the output of:

- XDMA (system_ila_0)
- data_fifo (system_ila_1)

Conclusion

This application note shows one way to continuously stream configuration data over PCIe to saturate the ICAP. Ultimately, the highest performance solution is one that maximizes delivery bandwidth to the ICAP by sending configuration data as quickly as the silicon allows. This partial bitstream delivery must be coupled with a sequence of events before, during, and after partial reconfiguration. For more information on design considerations, see the *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909).

Supported Features

- Fast configuration over PCIe® at maximum ICAP bandwidth capacity.
- Initial configuration from dual QSPI or JTAG ports.
- Debug at key stages of bitstream delivery path.
- Kintex® UltraScale+™ and Virtex® UltraScale+™ support.
- Use of standard example PCIe drivers.

Unsupported Features

- Kintex® UltraScale™ and Virtex® UltraScale™ devices are not supported with the delivered example designs. The same approach can be used to achieve the same results, but this has not been explicitly built or tested by Xilinx.
- This version supports only AXI4-Stream mode, not AXI4 memory mapped mode.

Some features are not yet implemented, but might be considered for a future release.

Known Limitations

The Partial Reconfiguration project flow does not yet support Reconfigurable Partition (RP) definition within block diagrams. Define all RPs within RTL or export block diagrams to the scripted non-project implementation flow.

References

1. *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909)
2. *DMA/Bridge Subsystem for PCI Express Product Guide* (PG195)
3. *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947)
4. *UltraScale Architecture Configuration User Guide* (UG570)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
03/11/2019 Version 1.0	
Initial Xilinx release.	N/A

Please Read: Important Legal Notices

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