Configuration via Protocol (CvP)
Implementation in V-series FPGA Devices
User Guide
Contents

Overview .............................................................................................................................................. 1-1
  Benefits of Using CvP.......................................................................................................................... 1-1
  CvP System ......................................................................................................................................... 1-2
  CvP Modes ......................................................................................................................................... 1-3
  CvP Revision Design Flow .................................................................................................................. 1-4
  CvP Topologies .................................................................................................................................... 1-4
  Additional Information about PCI Express .......................................................................................... 1-5

FPGA Configuration using CvP ........................................................................................................... 2-1
  CvP Configuration Images .................................................................................................................. 2-1
  CvP Modes ......................................................................................................................................... 2-2
    CvP Initialization Mode .................................................................................................................... 2-2
    CvP Update Mode ............................................................................................................................ 2-2
  Alternative to CvP Initialization: Autonomous HIP Mode .................................................................. 2-3
  CvP Compression and Encryption Features ...................................................................................... 2-3
    Data Compression ............................................................................................................................ 2-3
    Data Encryption ............................................................................................................................... 2-3
  Core Image Update ............................................................................................................................ 2-4
  CvP Pins ............................................................................................................................................. 2-5

CvP Topologies ..................................................................................................................................... 3-1
  Single Endpoint .................................................................................................................................. 3-1
  Multiple Endpoints ............................................................................................................................ 3-2
  Mixed Chain ..................................................................................................................................... 3-3
    Configuring Slave FPGAs with Different Configuration Files ...................................................... 3-3
    Configuring Slave FPGAs with a Single Configuration File ......................................................... 3-4

Design Considerations .......................................................................................................................... 4-1
  Preparing the Design for CvP Revision Design Flow ....................................................................... 4-1
  Designing CvP for an Open System .................................................................................................... 4-2
    FPGA Power Supplies Ramp Time Requirement ........................................................................ 4-2
    PCIe Wake-Up Time Requirement ................................................................................................. 4-3
  Designing CvP for a Closed System ................................................................................................. 4-9
    Clock Connections for CvP Designs Including the Transceiver Reconfiguration Controller ........ 4-9

CvP Example Designs .......................................................................................................................... 5-1
  Understanding the Design Steps for CvP Initialization Mode ......................................................... 5-2
    Downloading and Generating the High Performance Reference Design .................................... 5-4
    Setting up CvP Parameters for CvP Initialization Mode ................................................................. 5-7
Document Revision History for the Configuration via Protocol (CvP) Implementation in V-series FPGA Devices User Guide.................................7-1
Configuration via Protocol (CvP) is a configuration scheme supported in Arria® V, Cyclone® V, and Stratix® V device families. The CvP configuration scheme creates separate images for the periphery and core logic. You can store the periphery image in a local configuration device and the core image in host memory, reducing system costs and increasing the security for the proprietary core image. CvP configures the FPGA fabric through the PCI Express® (PCIe) link and it is available for Endpoint variants only.

**Related Information**

*Arria 10 CvP Initialization and Partial Reconfiguration over PCI Express User Guide*

Provides more information about the CvP implementation in Arria 10 devices.

**Benefits of Using CvP**

The CvP configuration scheme has the following advantages:

- Reduces system costs by reducing the size of the local flash device used to store the periphery configuration data.
- Improves security for the proprietary core bitstream. CvP ensures that the PCIe host can exclusively access the FPGA core image.
- Enables dynamic core updates without requiring a system power down. CvP allows the FPGA fabric to be updated through the PCIe link without a host reboot or FPGA full chip reinitialization.
- Provides a simpler software model for configuration. A smart host can use the PCIe protocol and the application topology to initialize and update the FPGA fabric.
- Facilitates hardware acceleration.
CvP System

The following figure shows the required components for a CvP system.

**Figure 1-1: CvP Block Diagram**

A CvP system typically consists of an FPGA, a PCIe host, and a configuration device.

1. The configuration device is connected to the FPGA using the conventional configuration interface. The configuration interface can be any of the supported schemes, such as active serial (AS), passive serial (PS), or fast passive parallel (FPP). The choice of the configuration device depends on your chosen configuration scheme.

2. PCIe Hard IP block (bottom left) for CvP and other PCIe applications.

3. PCIe Hard IP block only for PCIe applications and cannot be used for CvP.

Most Arria V, Cyclone V, and Stratix V FPGAs include more than one Hard IP block for PCI Express. The CvP configuration scheme can only utilize the bottom left PCIe Hard IP block on each device. It must be configured as an Endpoint.
CvP Modes

The CvP configuration scheme supports the following modes:

- CvP initialization mode
- CvP update mode

CvP Initialization Mode

This mode configures the core of the FPGA through the PCIe link upon system power up. Initialization refers to the initial fabric configuration image loaded in the FPGA fabric after power up.

Benefits of using CvP initialization mode include:

- Satisfying the PCIe wake-up time requirement
- Saving cost by storing the core image in the host memory
- Preventing unauthorized access to the core image

CvP Update Mode

This mode assumes that you have configured the FPGA with the full configuration image (both periphery and core) from a local configuration device after the initial system power up. The PCIe link is used for subsequent core image updates (only core, the periphery must remain unchanged during CvP update).

Choose this mode if you want to update the core image for any of the following reasons:

- To change core algorithms
- To perform standard updates as part of a release process
- To customize core processing for different components that are part of a complex system

Note: The CvP update mode works after the FPGA enters user mode. In user mode, the PCIe link is available for normal PCIe applications as well as to perform an FPGA core image update.

Table 1-1: CvP Support for Device Families

<table>
<thead>
<tr>
<th>Device</th>
<th>CvP Modes Supported</th>
<th>PCIe Gen 1</th>
<th>PCIe Gen 2</th>
<th>PCIe Gen 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria 10 (1)</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
</tr>
<tr>
<td>Stratix V</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
<td>No support</td>
</tr>
<tr>
<td></td>
<td>CvP Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arria V GZ</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
<td>No support</td>
</tr>
<tr>
<td></td>
<td>CvP Update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arria V</td>
<td>CvP Initialization</td>
<td></td>
<td>CvP Initialization</td>
<td>No support</td>
</tr>
<tr>
<td></td>
<td>CvP Update</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) For Arria 10 devices, refer to the *Arria 10 CvP Initialization and Partial Reconfiguration over PCI Express User Guide*.
CvP Modes Supported

<table>
<thead>
<tr>
<th>Device</th>
<th>PCIe Gen 1</th>
<th>PCIe Gen 2</th>
<th>PCIe Gen 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone V</td>
<td>CvP Initialization</td>
<td>CvP Initialization</td>
<td>No support</td>
</tr>
<tr>
<td></td>
<td>CvP Update</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Information

- **Arria 10 CvP Initialization and Partial Reconfiguration over PCI Express User Guide**
  Provides more information about the CvP implementation in Arria 10 devices.
- **CvP Initialization Mode** on page 2-2
- **CvP Update Mode** on page 2-2
- **CvP Example Designs** on page 5-1

CvP Revision Design Flow

This design flow prepares your design for subsequent updates of the all or part of the core logic. The reconfigured logic is called the reconfigurable core logic. This reconfigurable core logic can be programmed in User Mode while the PCIe link is up and fully enumerated.

You can create multiple core images that connect to the same periphery image. The core image contains both static and reconfigurable regions. The reconfigurable region must contain only resources that are controlled by CRAM such as LABS, embedded RAM blocks, and DSP blocks in the FPGA core fabric. It cannot contain any periphery components such as GPIOs, transceivers, PLL, I/O blocks, the Hard IP for PCI Express IP Core, or other components included in the periphery.

Related Information

- **Preparing the Design for CvP Revision Design Flow** on page 4-1

CvP Topologies

You can configure the FPGA using the following topologies:

- Single endpoint—to configure a single device.
- Multiple endpoints—to configure multiple devices using a PCIe switch.
- Mixed chain—to configure multiple devices using a single configuration file or multiple configuration files for slave devices in the chain.

Related Information

- **CvP Topologies** on page 3-1
Additional Information about PCI Express

The following links provide information about the PCI Express specifications and Altera's offerings for PCI Express.

Altera's PCI Express IP cores and devices are compliant with the following PCI-SIG® specifications:

- PCI Express Base Specification, Rev 1.1 (2.5 GT/s)
- PCI Express Base Specification, Rev 2.0 (2.5 GT/s and 5.0 GT/s)
- PCI Express Specification, Rev 3.0 (2.5 GT/s, 5.0 GT/s, and 8.0 GT/s)

For more information, please refer to the PCI-SIG webpage.

Related Information:

- Avalon-ST
  - Stratix V Avalon-ST Interface for PCIe Solutions User Guide
  - Arria V GZ Avalon-ST Interface for PCIe Solutions User Guide
  - Arria V Avalon-ST Interface for PCIe Solutions User Guide
  - Cyclone V Avalon-ST Interface for PCIe Solutions User Guide
- Avalon-MM
  - Stratix V Avalon-MM Interface for PCIe Solutions User Guide
  - Arria V GZ Avalon-MM Interface for PCIe Solutions User Guide
  - Arria V Avalon-MM Interface for PCIe Solutions User Guide
  - Cyclone V Avalon-MM Interface for PCIe Solutions User Guide
- Avalon-MM with DMA
  - V-Series Avalon-MM DMA Interface for PCIe Solutions User Guide
- Single Root I/O Virtualization
  - Stratix V Avalon-ST Interface with SR-IOV for PCIe Solutions User Guide
CvP Configuration Images

In CvP, you partition your design into two images: core image and periphery image.

You use the Quartus Prime software to generate the images:

- Periphery image (*.periph.jic)—contains general purpose I/Os (GPIOs), I/O registers, the GCLK, QCLK, and RCLK clock networks, PLLs, transceivers, hardened memory PHY and logic that is implemented in hard IP such as the JTAG interface, PR block, CRC block, Oscillator block, Impedance control block, Chip ID, ASMI block, Remote update block, Temperature sensor, and Hard IP for PCI Express IP Core. These components are included in the periphery image because they are controlled by I/O periphery register bits. The entire periphery image is static and cannot be reconfigured.

- Core image (*.core.rbf)—contains logic that is programmed by configuration RAM (CRAM). This image includes LABs, DSP, and embedded memory. The core image consists of a single reconfigurable region or both static and reconfigurable regions.

- Reconfigurable region - This region can be programmed in user mode while the PCIe link is up and fully enumerated. It must contain only resources that are controlled by CRAM such as LABs, embedded RAM blocks, and DSP blocks in the FPGA core image. It cannot contain any periphery components such as GPIOs, transceivers, PLLs, I/O blocks, the Hard IP for PCI Express IP Core, or other components included in the periphery image.

- Static region - This region cannot be modified.

Related Information

- I/O Features in Arria V Devices
  Provides more information about the location of the transceiver banks and I/O banks.

- I/O Features in Cyclone V Devices
  Provides more information about the location of the transceiver banks and I/O banks.

- I/O Features in Stratix V Devices
  Provides more information about the location of the transceiver banks and I/O banks.
CvP Modes

CvP Initialization Mode

In this mode, the periphery image is stored in an external configuration device and is loaded into the FPGA through the conventional configuration scheme. The core image is stored in a host memory and is loaded into the FPGA through the PCIe link.

After the periphery image configuration is complete, the CONF_DONE signal goes high and allows the FPGA to start PCIe link training. When PCIe link training is complete, the PCIe link transitions to L0 state and then through PCIe enumeration. The PCIe host then initiates the core image configuration through the PCIe link.

After the core image configuration is complete, the CvP_CONF_DONE pin goes high, indicating the FPGA is fully configured.

After the FPGA is fully configured, the FPGA enters user mode. If the INIT_DONE signal is enabled, the INIT_DONE signal goes high after initialization is complete and the FPGA enters user mode.

In user mode, the PCIe links are available for normal PCIe applications. You can also use the PCIe link to change the core image. To change the core image, create one or more FPGA core images in the Quartus Prime software that have identical connections to the periphery image.

Related Information

Configuration Timing Waveforms
Provides more information with the timing diagram for different configuration modes.

CvP Update Mode

In this mode, the FPGA device is initialized after initial system power up by loading the full configuration image from the external local configuration device to the FPGA.

After the full FPGA configuration image is complete, the CONF_DONE signal goes high.

After the FPGA is fully configured, the FPGA enters initialization and user mode. If the INIT_DONE signal is enabled, the INIT_DONE signal goes high after initialization is completed and the FPGA enters user mode.

In user mode, the PCIe links are available for normal PCIe applications. You can use the PCIe link to perform an FPGA core image update. To perform the FPGA core image update, you can create one or more FPGA core images in the Quartus Prime software that have identical connections to the periphery image.

Note: You cannot combine the features of CvP update mode and CvP initialization mode in a single design. For example, you cannot create a CvP update image for your Quartus Prime project and then specify a CvP initialization periphery image in your configuration scheme.

Related Information

Configuration Timing Waveforms
Provides more information with the timing diagram for different configuration modes.
Alternative to CvP Initialization: Autonomous HIP Mode

Autonomous mode is useful when you are not using CvP initialization to configure the FPGA, but still need to satisfy the 100 ms PCIe wake up time requirement. Altera’s FPGA devices always receive the configuration bits for the periphery image first, then for the core image. After the core image configures, the device enters user mode. In autonomous HIP mode, the Hard IP for PCI Express begins operation when the periphery configuration completes. Autonomous HIP mode allows the PCIe IP core to operate before the device enters user mode, during on-going core configuration.

In autonomous HIP mode, after completing link training, the Hard IP for PCI Express responds to Configuration Requests from the host with a Configuration Request Retry Status (CRS).

Arria V, Cyclone V, and Stratix V are the first devices to offer autonomous mode. In earlier devices, the PCI Express IP Core was released from reset only after the FPGA core was fully configured.

CvP Compression and Encryption Features

Data Compression

You can choose to compress the core image by turning on the Generate compressed bitstream option in the Configuration page of the Device and Pin Options dialog box in the Quartus Prime software. The periphery image cannot be compressed. Compressing the core image reduces the storage requirement.

If you first configure the FPGA using a compressed core image, you must use a compressed image when updating the subsequent core image of the FPGA.

Data Encryption

You can choose to encrypt the core image. The periphery image cannot be encrypted. To configure the FPGA with an encrypted core image, you must pre-program the FPGA with a security key. This key is then used to decrypt the incoming configuration bitstream.

A key-programmed FPGA can accept both encrypted and unencrypted bitstreams if you configure the FPGA using the AS, PS, or FPP scheme. However, if you use CvP, a key-programmed FPGA can only accept encrypted bitstreams. Use the same key to encrypt all revisions of the core image.

Table 2-1: Supported Clock Source for Encrypted Configuration Data

<table>
<thead>
<tr>
<th>Key Types</th>
<th>Active Serial</th>
<th>Passive Serial</th>
<th>Fast Passive Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>External Clock</td>
<td>Internal Clock</td>
<td>External Clock</td>
</tr>
<tr>
<td>Volatile key</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Non-volatile key</td>
<td>No</td>
<td>12.5 MHz</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Related Information

Using the Design Security Features in Altera FPGAs
Core Image Update

After the FPGA enters user mode, the PCIe host can trigger an FPGA core image update through the PCIe link. Both CvP initialization mode and CvP update mode support core images updates.

You must choose the same bitstream settings for all core images. For example, if you have selected either encryption, compression, or both encryption and compression features for the first core image, you must ensure you turned on the same features for the other core images that you will use for core image update using CvP.

Figure 2-1: Periphery and Core Images Storage Arrangement for CvP Core Image Update

The periphery image remains the same for different core image updates. If you change the periphery image, you must reprogram the local configuration device with the new periphery image.

You can use CvP revision design flow to create multiple reconfigurable core images that connect to the same periphery image.

When you initiate a core image update, the CvP_CONF_DONE pin is pulled low, indicating a core image update has started. The FPGA fabric is reinitialized and reconfigured with the new core image. During the core image update through a PCIe link, the nCONFIG and nSTATUS pins of the FPGA remain at logic high. When the core image update completes, the CvP_CONF_DONE pin is released high, indicating the FPGA has entered user mode.

Related Information

- **Configuration Timing Waveforms**
  Provides more information with the timing diagram for different configuration modes.
- **Preparing the Design for CvP Revision Design Flow** on page 4-1
## CvP Pins

The following table lists the CvP pin descriptions and connection guidelines.

**Table 2-2: CvP pin descriptions and connection guidelines**

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Type</th>
<th>Pin Description</th>
<th>Pin Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>CvP_CONFDONE</td>
<td>Output</td>
<td>The <code>CvP_CONFDONE</code> pin is driven low during configuration. When configuration via PCIe is complete, this signal is released and either actively driven high, or pulled high by an external pull-up resistor. During FPGA configuration in CvP initialization mode, you must observe this pin after the <code>CONF_DONE</code> pin goes high to determine if the FPGA is successfully configured. If you are not using the CvP modes, you can use this pin as a user I/O pin.</td>
<td>If this pin is set as dedicated output, the $V_{CCPGM}$ power supply must meet the input voltage specification of the receiving side. If this pin is set as an open-drain output, connect the pin to an external 10-kΩ pull-up resistor to the $V_{CCPGM}$ power supply or a different pull-up voltage that meets the input voltage specification of the receiving side. This gives an advantage on the voltage leveling.</td>
</tr>
<tr>
<td>INIT_DONE</td>
<td>Output</td>
<td>When you enable this pin, a transition from low to high at the pin indicates the device has entered user mode. If the <code>INIT_DONE</code> output is enabled, the <code>INIT_DONE</code> pin cannot be used as a user I/O pin after configuration. This is a dual-purpose pin and can be used as an I/O pin when not enabled as the <code>INIT_DONE</code> pin.</td>
<td>When you use the optionally open-drain output dedicated <code>INIT_DONE</code> pin, connect this pin to an external 10-kΩ pull-up resistor to $V_{CCPGM}$. When you use this pin in an AS or PS multi-device configuration mode, ensure you enable the <code>INIT_DONE</code> pin in the Quartus Prime designs. When you do not use the dedicated <code>INIT_DONE</code> optionally open-drain output, and when this pin is not used as an I/O pin, connect this pin as defined in the Quartus Prime software.</td>
</tr>
<tr>
<td>Pin Name</td>
<td>Pin Type</td>
<td>Pin Description</td>
<td>Pin Connection</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>----------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| CONF_DONE  | Bidirectional | Dedicated configuration done pin.  
As a status output, the CONF_DONE pin drives low before and during configuration. After all configuration data is received without error and the initialization cycle starts, CONF_DONE is released.  
As a status input, the CONF_DONE pin goes high after all data is received. Then the device initializes and enters user mode. This pin is not available as a user I/O pin. | Connect an external 10-kΩ pull-up resistors to VCCPGM. VCCPGM must be high enough to meet the VIH specification of the I/O on the device and the external host.  
When you use passive configuration schemes, the configuration controller monitors this pin. |
### Pin Name

nPERST\([L,R]\) \[0:1\]

### Pin Type

Input

### Pin Description
This pin is connected to the Hard IP for PCI Express IP Core as a dedicated fundamental reset pin for PCIe usage. If the signal is low, the transceivers and dedicated PCIe Hard IP block that you use for CvP operation are in the reset mode.

### Pin Connection

Connect the \(n\text{PERST}[L,R]0\) and \(n\text{PERST}[L,R]1\) to the \(\text{PERST#}\) pin of the PCIe slot. This pin may be driven by 3.3V compatible I/O standards. Only one \(n\text{PERST}\) pin is used per instances of Hard IP for PCI Express.

In Arria V and Stratix V devices, these pins have the following locations:

- \(n\text{PERST}L0\) = Bottom Left PCIe Hard IP & CvP
- \(n\text{PERST}L1\) = Top Left PCIe Hard IP (When available)
- \(n\text{PERST}R0\) = Bottom Right PCIe Hard IP (When available)
- \(n\text{PERST}R1\) = Top Right PCIe Hard IP (When available)

In Cyclone V devices, these pins have the following locations:

- \(n\text{PERST}L0\) = Top Left PCIe Hard IP
- \(n\text{PERST}L1\) = Bottom Left PCIe Hard IP & CvP

**Note:** For maximum compatibility, always use the bottom left PCIe Hard IP first, as this is the only location that supports Configuration via Protocol (CvP) using the PCIe link.

### Related Information

- **Pin Connection Guidelines**
  Provides more information about related configuration pins. Refer to the respective device family pin connection guidelines.
- **Arria V Device Datasheet**
- **Cyclone V Device Datasheet**
- **Stratix V Device Datasheet**
CvP supports several types of topologies that allow you to configure single or multiple FPGAs.

Single Endpoint

Use the single Endpoint topology to configure a single FPGA. In this topology, the PCIe link connects one PCIe Endpoint in the FPGA device to one PCIe Root Port in the host.

Figure 3-1: Single Endpoint Topology
Multiple Endpoints

Use the multiple Endpoints topology to configure multiple FPGAs through a PCIe switch. This topology provides you with the flexibility to select the device to configure or update through the PCIe link. You can connect any number of FPGAs to the host in this topology.

The PCIe switch controls the core image configuration through the PCIe link to the targeted PCIe Endpoint in the FPGA. You must ensure that the Root Port can respond to the PCIe switch and direct the configuration transaction to the designated Endpoint based on the bus/device/function address of the Endpoint specified by the PCIe switch.

Figure 3-2: Multiple Endpoints Topology
Mixed Chain

Use the mixed chain topology to configure multiple FPGAs that are connected in a chain using both the PCIe link and conventional configuration scheme. In this topology, the PCIe link connects the Endpoint of the master FPGA (the first FPGA in the chain) to the PCIe Root Port in the host. The slave FPGAs are connected in the chain using the PS or FPP configuration scheme. The configuration device, which you use to store the periphery image in the CvP initialization mode and the full configuration image in the CvP update mode, is only connected to the master FPGA. The master FPGA is configured first, followed by the slave FPGAs.

You must design a user IP for the master FPGA to fetch the configuration data from the Root Port to the slave FPGAs in the chain. The data is latched out from the master device through the GPIOs and latched into the slave devices through the PS or FPP configuration pins—DCLK, DATA line, or DATA bus.

By tying DCLK, nCONFIG, nSTATUS, CONF_DONE pins, and DATA bus of the slave devices together, the slave devices enter user mode at the same time. If any device in the chain detects an error, the slave device chain reinitializes and reconfigures by pulling its nSTATUS pin low. You must ensure there is a suitable line buffering on the DCLK and DATA bus if you are configuring more than four slave devices in the chain.

Configuring Slave FPGAs with Different Configuration Files

To configure the slave FPGAs with different configuration files, connect the nCEO pin of one slave FPGA to the nCE pin of the next slave FPGA in the chain. When the first slave FPGA completes configuration, the slave FPGA pulls the nCEO pin low to enable configuration for the next slave FPGA. This process continues until the last slave FPGA in the chain is configured. You can leave the nCEO pin of the last device unconnected or use the pin as a user I/O pin.

Figure 3-3: Mixed Chain Topology with Different Configuration Files

The following figure shows the connections required to configure slave FPGAs with different configuration files using the FPP scheme.
Configuring Slave FPGAs with a Single Configuration File

To configure slave FPGAs with the same configuration file, connect the \texttt{nCEO} pin of the master FPGA to the \texttt{nCE} pins of all slave FPGAs in the chain. In this topology, all the slave devices are configured at the same time. You can leave the \texttt{nCEO} pin of the slave FPGAs in the chain unconnected or use it as a GPIO pin.

**Figure 3-4: Mixed Chain Topology with Single Configuration File**

The following figure shows the connections required to configure slave FPGAs with the same configuration file using the FPP scheme.
Preparing the Design for CvP Revision Design Flow

The CvP revision design flow requires separate bitstreams for design elements implemented in the I/O ring (periphery) and FPGA core fabric. To use an I/O bitstream with multiple FPGA core fabric bitstreams, separate periphery elements from the reconfigurable core logic.

- The I/O ring, or periphery partition controlled by I/O periphery register bits:
  - I/O registers
  - General-purpose I/Os (GPIOs)
  - Transceivers
  - Phase-locked loops (PLLs)
  - Hard IP for PCI Express
  - Hardened memory PHY
  - Global clocks (GCLK)
  - Regional clocks (RCLK)

- The core partition: Core logic to program the FPGA fabric. The core logic contains both the static core region and the reconfigurable core region.
  - Reconfigurable region - This region can be programmed in user mode while the PCIe link is up and fully enumerated. It must contain only resources that are controlled by CRAM such as LABs, embedded RAM blocks, and DSP blocks in the FPGA core image. It cannot contain any periphery components such as GPIOs, transceivers, PLL, I/O blocks, the Hard IP for PCI Express IP Core, or other components included in the periphery image.
  - Static region - This region cannot be modified.

You may create one or more partitions for the core fabric; however, only one partition can include the logic that you plan to reconfigure.

You must ensure the reconfigurable core logic does not contain any periphery components. Failure to make these connections results in the following Quartus Prime compilation error:

Error (142040): Detected illegal nodes in reconfigurable partitions. Only core logic is reconfigurable in this version of the Quartus Prime software.
The following figure shows the recommended design hierarchy for a design including the Hard IP PCI Express IP Core, an interface to DDR3 SDRAM, and core logic.

This design hierarchy represents the actual partition after the Quartus Prime compilation. You must ensure that the reconfigurable core logic does not contain any periphery elements. Separation of the core and peripheral logic is an iterative process and may take several Quartus Prime compilations to find all peripheral logic that needs to be isolated from reconfigurable core logic.

**Related Information**
- *Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design*

### Designing CvP for an Open System

While designing a CvP system for an Open System where you don't control both ends of the PCIe link completely, ensure that you observe the guidelines provided in this section.

### FPGA Power Supplies Ramp Time Requirement

For an open system, you must ensure that your design adheres to the FPGA power supplies ramp-up time requirement.

The power-on reset (POR) circuitry keeps the FPGA in the reset state until the power supply outputs are in the recommended operating range. A POR event occurs from when you power up the FPGA until the power supplies reach the recommended operating range within the maximum power supply ramp time, $t_{\text{RAMP}}$. If $t_{\text{RAMP}}$ is not met, the device I/O pins and programming registers remain tri-stated, during which device configuration could fail.

For CvP, the total $t_{\text{RAMP}}$ must be less than 10 ms, from the first power supply ramp-up to the last power supply ramp-up. You must select fast POR by setting the $\text{PORSEL}$ pin to high. The fast POR delay time is in the range of 4–12 ms, allowing sufficient time after POR for the PCIe link to start initialization and configuration.
Figure 4-2: Power Supplies Ramp-Up Time and POR

<table>
<thead>
<tr>
<th>Volts</th>
<th>POR Delay</th>
<th>Configuration Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Information

- Power Management in Arria V Devices
- Power Management in Cyclone V Devices
- Power Management in Stratix V Devices

PCle Wake-Up Time Requirement

For an open system, you must ensure that the PCle link meets the PCle wake-up time requirement as defined in the PCI Express CARD Electromechanical Specification. The transition from power-on to the link active (L0) state for the PCle wake-up timing specification must be within 200 ms. The timing from FPGA power-up until the Hard IP for PCI Express IP Core in the FPGA is ready for link training must be within 120 ms.

Related Information

PCI Express Card Electromechanical 3.0 Specification

PCle Wake-Up Time Requirement for CvP Initialization Mode

For CvP initialization mode, the Hard IP for PCI Express IP core is guaranteed to meet the 120 ms requirement because the periphery image configuration time is significantly less than the full FPGA configuration time. Therefore, you can choose any of the conventional configuration schemes for the periphery image configuration.

To ensure successful configuration, all POR-monitored power supplies must ramp up monotonically to the operating range within the 10 ms ramp-up time. The PERST# signal indicates when the FPGA power supplies are within their specified voltage tolerances and the REFCLK is stable. The embedded hard reset controller triggers after the internal status signal indicates that the periphery image has been loaded. This
reset does not trigger off of PERST#. For CvP initialization mode, the PCIe link supports the FPGA core image configuration and PCIe applications in user mode.

**Note:** For Gen 2 capable Endpoints, after loading the core .sof, Altera recommends that you verify that the link has been trained to the expected Gen 2 rate. If the link is not operating at Gen 2, host software can trigger the Endpoint to retrain.

**Figure 4-3: PCIe Timing Sequence in CvP Initialization Mode**

**Table 4-1: Power-Up Sequence Timing in CvP Initialization Mode**

<table>
<thead>
<tr>
<th>Timing Sequence</th>
<th>Timing Range (ms)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>Maximum ramp-up time requirement for all POR-monitored power supplies in the FPGA to reach their respective operating range.</td>
</tr>
<tr>
<td>b</td>
<td>4–12</td>
<td>FPGA POR delay time.</td>
</tr>
<tr>
<td>c</td>
<td>100</td>
<td>Minimum PERST# signal active time from the host.</td>
</tr>
<tr>
<td>d</td>
<td>20</td>
<td>Minimum PERST# signal inactive time from the host before the PCIe link enters training state.</td>
</tr>
<tr>
<td>e</td>
<td>120</td>
<td>Maximum time from the FPGA power up to the end of periphery configuration in CvP initialization mode.</td>
</tr>
<tr>
<td>f</td>
<td>100</td>
<td>Maximum time PCIe device must enter L0 after PERST# is deasserted.</td>
</tr>
</tbody>
</table>

**PCle Wake-Up Time Requirement for CvP Update Mode**

For CvP update mode, you initialize the FPGA by configuring it using one of the conventional configuration schemes upon device power-up. An open system requires that the FPGA initialization complete within 120 ms. To ensure that this requirement is met, choose the right conventional configuration scheme for your system.

To ensure successful configuration, all POR-monitored power supplies must ramp up monotonically to the operating range within the 10 ms ramp-up time. PERST# is one of the auxiliary signals specified in the PCIe electromechanical specification. The PERST# signal is sent from the PCIe host to the FPGA. The
The **PERST#** signal indicates whether the power supplies of the FPGA are within their specified voltage tolerances and are stable. The embedded hard reset controller triggers after the internal status signal indicates that the periphery image has been loaded. This reset does not trigger off of **PERST#**. The **PERST#** signal also initializes the FPGA state machines and other logic after power supplies are stabilized. The PCIe link supports PCIe applications in user mode for CvP update mode, therefore, you can use the PCIe link for core image update.

**Note:** For Gen 2 capable Endpoints, after loading the core .sof, Altera recommends to verify that the link has been trained to the expected Gen 2 rate. If the link is not operating at Gen 2, software can trigger the Endpoint to retrain.

**Figure 4-4: PCIe Timing Sequence in CvP Update Mode**

![Figure 4-4: PCIe Timing Sequence in CvP Update Mode](image)

**Table 4-2: Power-Up Sequence Timing in CvP Update Mode**

<table>
<thead>
<tr>
<th>Timing Sequence</th>
<th>Timing Range (ms)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>Maximum ramp-up time requirement for all POR-monitored power supplies in the FPGA to reach their respective operating range.</td>
</tr>
<tr>
<td>b</td>
<td>4–12</td>
<td>FPGA POR delay time.</td>
</tr>
<tr>
<td>c</td>
<td>100</td>
<td>Minimum <strong>PERST#</strong> signal active time from the host.</td>
</tr>
<tr>
<td>d</td>
<td>20</td>
<td>Minimum <strong>PERST#</strong> signal inactive time from the host before the PCIe link enters training state.</td>
</tr>
<tr>
<td>e</td>
<td>120</td>
<td>Maximum time from the FPGA power up to the end of the full FPGA configuration in CvP update mode.</td>
</tr>
<tr>
<td>f</td>
<td>100</td>
<td>Maximum time PCIe device must enter <strong>L0</strong> after <strong>PERST#</strong> is deasserted.</td>
</tr>
</tbody>
</table>

**Recommended Configuration Schemes**

**Design Considerations**

Send Feedback
For CvP initialization mode, you can configure the FPGA with the periphery image using the AS, PS, or FPP configuration scheme.

For CvP update mode, you can configure the FPGA fully using one of the configuration schemes listed in the table below. The table lists the configuration schemes based on the fastest DCLK frequency with data compression and encryption features disabled in the CvP update mode. These features require different data to clock ratios, which prolongs total configuration time. Consequently, total configuration time does not meet the 200-ms PCIe wake-up timing specification.

### Table 4-3: Recommended Configuration Schemes for CvP Update Mode

<table>
<thead>
<tr>
<th>Variant</th>
<th>Member Code</th>
<th>Configuration Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria V GX</td>
<td>A1</td>
<td>FPP x8</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>A5</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>A7</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>B3</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>B5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B7</td>
<td></td>
</tr>
<tr>
<td>Arria V GT</td>
<td>C3</td>
<td>FPP x8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>C7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D3</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>D7</td>
<td></td>
</tr>
<tr>
<td>Arria V GZ</td>
<td>E1</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>E3</td>
<td>FPP x32</td>
</tr>
<tr>
<td></td>
<td>E5</td>
<td>FPP x32</td>
</tr>
<tr>
<td></td>
<td>E7</td>
<td></td>
</tr>
<tr>
<td>Arria V SX</td>
<td>B3</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>B5</td>
<td></td>
</tr>
<tr>
<td>Arria V ST</td>
<td>D3</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>D5</td>
<td></td>
</tr>
<tr>
<td>Variant</td>
<td>Member Code</td>
<td>Configuration Scheme</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Cyclone V GX</td>
<td>C3</td>
<td>AS x4, FPP x8, FPP x16</td>
</tr>
<tr>
<td></td>
<td>C4</td>
<td>FPP x8</td>
</tr>
<tr>
<td></td>
<td>C5</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>C7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C9</td>
<td>FPP x16</td>
</tr>
<tr>
<td>Cyclone V GT</td>
<td>D5</td>
<td>FPP x8</td>
</tr>
<tr>
<td></td>
<td>D7</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>D9</td>
<td>FPP x16</td>
</tr>
<tr>
<td>Cyclone V SX</td>
<td>C2</td>
<td>TBD</td>
</tr>
<tr>
<td></td>
<td>C4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C5</td>
<td>FPP x8</td>
</tr>
<tr>
<td></td>
<td>C6</td>
<td>FPP x16</td>
</tr>
<tr>
<td>Cyclone V ST</td>
<td>D5</td>
<td>FPP x8</td>
</tr>
<tr>
<td></td>
<td>D6</td>
<td>FPP x16</td>
</tr>
<tr>
<td>Stratix V GX</td>
<td>A3</td>
<td>FPP x16</td>
</tr>
<tr>
<td></td>
<td>A4</td>
<td>FPP x32</td>
</tr>
<tr>
<td></td>
<td>A5</td>
<td>FPP x32</td>
</tr>
<tr>
<td></td>
<td>A7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A9</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>AB</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>B5</td>
<td>FPP x32</td>
</tr>
<tr>
<td></td>
<td>B6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B9</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>BB</td>
<td>—</td>
</tr>
<tr>
<td>Stratix V GT</td>
<td>C5</td>
<td>FPP x32</td>
</tr>
<tr>
<td></td>
<td>C7</td>
<td></td>
</tr>
</tbody>
</table>
### Estimating PCIe Wake-Up Time Requirement

**Figure 4-5: Estimating PCIe Wake-Up Time Requirement Equation**

\[
\left( \frac{\text{Full configuration file size}}{\text{Number of data lines}} \times \frac{1}{\text{DCLK frequency}} \right) + (\text{Power ramp up} + \text{POR delay time})
\]

Conventions used for the equation:

- Full configuration file size in bits—refer to uncompressed `.rbf` sizes.
- Number of data lines—refer to the width of data bus. For example, the width of data bus for FPP x16 is 16.
- DCLK frequency—refer to \( f_{\text{MAX}} \) for the DCLK frequency.
- Power ramp up—must be within 10 ms.
- POR delay—use fast POR, maximum time is 12 ms.

You can use the equation above to estimate whether your device meets the PCIe wake-up time requirement. The following figure shows an example calculation for the PCIe wake-up time requirement on an Arria V GX A5 device.

**Figure 4-6: Example Calculation of PCIe Wake-Up Time Requirement**

\[
\left( \frac{101,740,640}{16} \times \frac{1}{125,000,000} \right) + (10 + 12)
\]

\[= 50 \text{ ms} + 22 \text{ ms} \]

\[= 72 \text{ ms} \]

The estimation for Arria V GX A5 device is 72 ms, which meets the PCIe wake-up time requirement of 120 ms.
Related Information

- **Arria V Device Datasheet**
  Provides more information about uncompressed .rbf sizes for entire FPGA and IOCSR, DCLK frequency, and POR delay.

- **Cyclone V Device Datasheet**
  Provides more information about uncompressed .rbf sizes for entire FPGA and IOCSR, DCLK frequency, and POR delay.

- **Stratix V Device Datasheet**
  Provides more information about uncompressed .rbf sizes for entire FPGA and IOCSR, DCLK frequency, and POR delay.

---

**Designing CvP for a Closed System**

While designing CvP for a closed system where you control both ends of the PCIe link, estimate the periphery configuration time for CvP initialization mode or full FPGA configuration time for CvP update mode. You must ensure that the estimated configuration time is within the time allowed by the PCIe host.

**Clock Connections for CvP Designs Including the Transceiver Reconfiguration Controller**

If your design includes the following components:

- An Arria V, Cyclone V, or Stratix V device with CvP enabled
- Any additional transceiver PHY connected to the same Transceiver Reconfiguration Controller

then you must connect the PLL reference clock which is called refclk to the mgmt_clk_clk signal of the Transceiver Reconfiguration Controller and the additional transceiver PHY. In addition, if your design includes more than one Transceiver Reconfiguration Controllers on the same side of the FPGA, these controllers all must share the mgmt_clk_clk signal.

**Note:**

- For Stratix V and Arria V GZ devices, when CvP is enabled you cannot use dynamic transceiver reconfiguration for the transceiver channels in CvP-enabled Hard IP until after the core is loaded.

- For Cyclone V and Arria V devices, when CvP is enabled in PCIe Gen1 mode, you cannot use dynamic transceiver reconfiguration for the transceiver channels in CvP-enabled Hard IP until after the core is loaded.
The example designs in this chapter illustrate the steps required for CvP initialization mode, CvP initialization with subsequent changes to the core logic, and CvP update mode. All of them start with the PCI Express High Performance Reference Design that you can download from the Altera website. The example designs also show how to use the CvP revision design flow to prepare the design for reconfigurable core logic.

The CvP process involves the interactions between the PCI Express host, the FPGA Control Block, the Stratix V Hard IP for PCI Express IP Core, and the CRAM in FPGA as indicated in the following figure. The Control Block and FPGA CRAM are hidden. You cannot access them. Consequently, you cannot simulate the CvP functionality.

**Figure 5-1: Key Components in a CvP Design**
Table 5-1: Key Files for the CvP Qsys Example Design

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altpcied_sv.sdc</td>
<td>Synopsys Design Constraints (.sdc) for the Hard IP for PCI Express IP Core.</td>
</tr>
<tr>
<td>top_hw.sdc</td>
<td>Top-level timing constraint file .sdc for the complete design.</td>
</tr>
<tr>
<td>top_hw.v</td>
<td>Top-level wrapper for the PCI Express High Performance Reference Design.</td>
</tr>
<tr>
<td>top.cof</td>
<td>CvP conversion file for CvP initialization mode. This file specifies the input and output files that Quartus Prime software requires to split the original .sof or .pof file into periphery and core images.</td>
</tr>
<tr>
<td>pcie_lib</td>
<td>Design files that are used by synthesis tools.</td>
</tr>
</tbody>
</table>

Related Information

- [Getting started with PCI Express](#)
- [PCI Express High Performance Reference Design](#)

Understanding the Design Steps for CvP Initialization Mode

CvP initialization mode divides the design into periphery and core images. The periphery image can be stored in a local flash device on the PCB and the user can program the periphery via Active Serial (AS) mode. The core image is stored in host memory. You must download the core image to the FPGA using the PCI Express link.

**Note:** If you plan to create multiple versions of the core logic for the same periphery I/O, the new core images must work with the previous periphery image. Refer to [Understanding the Design Steps for CvP Initialization Mode with the Revision Design Flow](#) on page 5-14 for information about how to create reconfigurable images that connect to the same periphery image.

You must specify CvP initialization mode in the Quartus Prime software by selecting the CvP Settings Core initialization and update. You might choose CvP initialization mode for any of the following reasons:

- To save cost by storing the core image in external host memory.
- To prevent unauthorized access to the core image by storing it on the host.
Note: When you select CvP initialization mode, you must use the CMU PLL and the hard reset controller for the PCI Express Hard IP.

The CvP initialization demonstration walkthrough includes the following steps:

1. **Downloading and Generating the High Performance Reference Design** on page 5-4
2. **Setting up CvP Parameters for CvP Initialization Mode** on page 5-7
3. **Compiling the Design for the CvP Initialization Mode** on page 5-11
4. **Splitting the SOF File for the CvP Initialization Design Mode** on page 5-11
5. **Bringing Up the Hardware** on page 5-48

### Related Information
- **Transceiver PHY Reset Controller IP Core**

---

**Figure 5-2: Design Flow for CvP Initialization Mode**

1. **Enable CvP in PCIe Hard IP and set the CvP Parameters in Device and Pin Options**
2. **Compile the Design**
3. **Split the .sof file into core.rbf and periph.jic**
4. **Program Periphery via JTAG using periph.jic**
5. **Confirm Link and Data Rate**
6. **Program Core via PCIe Link using core.rbf**

**Note:**
1. You can split .sof file into core.rbf and periph.pof if you are using Passive Serial or Fast Passive Parallel mode for periphery image configuration.
2. You can use periph.pof if you are using Passive Serial or Fast Passive Parallel mode.
Transceiver PLL IP Core for Stratix V, Arria V, and Arria V GZ Devices

Downloading and Generating the High Performance Reference Design

Follow these steps to regenerate the PCI Express High Performance Reference Design with CvP enabled:

1. Download the PCIe AVST and On-Chip Memory Interface design files from the PCI Express Protocol web page. This design includes the correct pin assignments and project settings to target the Stratix V GX FPGA Development Kit.
3. Copy hip_s5gx_x1_g1_ast64_140.qar to your working directory.
4. Start the Quartus Prime software and restore hip_s5gx_x1_g1_ast64_140.qar.
5. On the Tools menu, select Qsys.
6. Open top.qsys.
7. On the System Contents tab, right-click DUT and select Edit.
8. Under System Settings, turn on Enable configuration via the PCIe link as shown in the following figure.

Figure 5-3: Hard IP for PCI Express Parameter Editor

<table>
<thead>
<tr>
<th>System Settings</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of lanes:</td>
<td>x1</td>
</tr>
<tr>
<td>Lane rate:</td>
<td>Gen1 (2.5 Gbps)</td>
</tr>
<tr>
<td>Port type:</td>
<td>Native endpoint</td>
</tr>
<tr>
<td>Application interface:</td>
<td>Avalon-ST 64-bit</td>
</tr>
<tr>
<td>RX buffer credit allocation – performance for received requests:</td>
<td>Low</td>
</tr>
<tr>
<td>Reference clock frequency:</td>
<td>100 MHz</td>
</tr>
<tr>
<td>Use 62.5 MHz application clock</td>
<td></td>
</tr>
<tr>
<td>Use deprecated RX Avalon-ST data byte enable port (rx_st_be)</td>
<td></td>
</tr>
<tr>
<td>Enable configuration via the PCIe link</td>
<td></td>
</tr>
<tr>
<td>Enable Hard IP reconfiguration</td>
<td></td>
</tr>
<tr>
<td>Number of Functions:</td>
<td>1</td>
</tr>
</tbody>
</table>

9. Click Finish.
10. On the Generation tab, specify the settings in the following table. Then click Generate at the bottom of the window.

Table 5-2: Qsys Generation Tab Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create simulation model</td>
<td>None</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Create testbench Qsys system</td>
<td>None</td>
</tr>
<tr>
<td>Create testbench simulation model</td>
<td>None</td>
</tr>
<tr>
<td>Create HDL design files for synthesis</td>
<td>Verilog</td>
</tr>
<tr>
<td>Create block symbol file (.bsf)</td>
<td>Leave this entry off.</td>
</tr>
<tr>
<td>Path</td>
<td><code>&lt;working_dir&gt;/top</code></td>
</tr>
<tr>
<td>Simulation</td>
<td>Leave this entry blank.</td>
</tr>
<tr>
<td>Testbench</td>
<td><code>&lt;working_dir&gt;/top/synthesis</code></td>
</tr>
</tbody>
</table>

Figure 5-4: Qsys Generation Window

11. After successful compilation, close Qsys.
12. After creating an IP Variation, to add this IP to your Quartus project, you must manually add the `.qip` and `.sip` files.
   The `.qip` is located in `<working_dir>/synthesis/top.qip`
   The `.sip` is located in `<working_dir>/simulation/top.sip`
13. On the Assignments menu, select **Settings**.

14. In the **Files** category, remove the existing **top.qip** IP Variation File.

15. Browse to the new **top.qip** file created after generating the IP Core, located in `<working_dir>/synthesis/top.qip`.

16. Click **Add** and **OK** to close the Settings window.
Setting up CvP Parameters for CvP Initialization Mode

Follow these steps to specify CvP parameters using the Quartus Prime software:

1. On the Quartus Prime Assignments menu, select Device, and then click Device and Pin Options.
2. Under Category select General, and then enable following options:
   a. **Auto-restart configuration after error.** If this option is enabled, CvP restarts after an error is detected.
   b. **Enable autonomous PCIe HIP mode.**
      Checking this box has no affect if you have enabled CvP by turning on Enable Configuration via the PCIe link in the Hard IP for PCI Express GUI, hence Altera recommends to keep it unchecked.
   c. Leave all other options disabled.
3. Under **Category**, select **Configuration** to specify the configuration scheme and device. Specify the settings in the following table:

**Table 5-3: CvP Initialization Mode Configuration Settings**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration scheme</td>
<td>Active Serial x4</td>
</tr>
<tr>
<td>Configuration mode</td>
<td>Standard</td>
</tr>
<tr>
<td>Configuration device</td>
<td>EPCQ256</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Configuration device I/O voltage</td>
<td>Auto</td>
</tr>
<tr>
<td>Force VCCIO to be compatible with configuration</td>
<td>Leave this option off.</td>
</tr>
<tr>
<td>I/O voltage</td>
<td></td>
</tr>
<tr>
<td>Generate compressed bitstreams</td>
<td>Turn this option off. Because this is a small example design, it</td>
</tr>
<tr>
<td></td>
<td>does not use a compressed bitstream. For larger designs, using a</td>
</tr>
<tr>
<td></td>
<td>compressed bitstream significantly reduces configuration time. In</td>
</tr>
<tr>
<td></td>
<td>addition, a compressed bitstream requires a smaller flash device.</td>
</tr>
<tr>
<td>Active serial clock source</td>
<td>100 MHz Internal Oscillator</td>
</tr>
<tr>
<td>Enable input tri-state on active configuration pins</td>
<td>Leave this option off.</td>
</tr>
<tr>
<td>in user mode</td>
<td></td>
</tr>
</tbody>
</table>
These **Configuration** settings use the configuration devices available on the Stratix V GX FPGA Development Board. The **EPCQ256** flash device is far larger than required to load a periphery image.

4. Under **Category** select **CvP Settings**. For **CvP Initialization** mode, specify the following settings in the following table:

<table>
<thead>
<tr>
<th>Table 5-4: CvP Initialization Category Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>CvP Initialization</td>
</tr>
<tr>
<td>Enable CvP_CONFDONE pin</td>
</tr>
<tr>
<td>Enable open drain on CvP_CONFDONE pin</td>
</tr>
</tbody>
</table>
5. Click OK to close the Device and Pin Options dialog box.
6. Click OK to close the Device dialog box.
7. Save your project.

Compiling the Design for the CvP Initialization Mode

1. To compile the design, on the Processing menu, select Start compilation. Compilation creates a .sof file in the pcie_quartus_files subdirectory.

Splitting the SOF File for the CvP Initialization Design Mode

Follow these steps to split your .sof file into separate images for the periphery and core logic.

2. Under Output programming files to convert, specify the options in the following table.
Table 5-5: CvP Initialization Output Programming Files Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming file type</td>
<td>JTAG Indirect Configuration File (.jic)</td>
</tr>
<tr>
<td>Configuration device</td>
<td>EPCQ256</td>
</tr>
<tr>
<td>Mode</td>
<td>Active Serial x4</td>
</tr>
<tr>
<td>File name</td>
<td>Browse to and select the /pcie_quartus_files/ directory. Type the file name top.jic. Then click Save.</td>
</tr>
<tr>
<td>Create Memory Map File</td>
<td>Turn this option on.</td>
</tr>
<tr>
<td>Create CvP files</td>
<td>Turn this option on. This box is greyed out until you specify the SOF Data file under Input files to convert.</td>
</tr>
</tbody>
</table>

3. Under Input files to convert, specify the options in the following table:

Table 5-6: CvP Initialization Input Files to Convert Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click Flash Loader</td>
<td>Click Add Device and select Stratix V and then 5SGXEA7K2, and click OK.</td>
</tr>
<tr>
<td>Click SOF Data</td>
<td>Click Add File and navigate to /pcie_quartus_files/top.sof. If you specified a compressed or encrypted bitstream in the Device and Pin Options dialog box, you must specify the same options for Conversion Programming File window. To enable these settings, click top.sof. Then click Properties and check the appropriate boxes.</td>
</tr>
<tr>
<td>Mode</td>
<td>Active Serial x4</td>
</tr>
</tbody>
</table>

The following figure illustrates the options that you specified.
4. Turn on the **Create CvP files (Generate top.periph.jic and top.core.rbf)** parameter in the **Output Programming Files** section.

   **Note:** If you do not check this box, the Quartus Prime software does not create separate files for the periphery and core images.

5. Click **Save Conversion Setup** to save these settings. For this exercise, call your settings *cvp_base.cof*. The Quartus Prime software does not automatically save your choices.

6. Click **Generate** to create **top.periph.jic** and **top.core.rbf**.

   **Note:** The generated CvP peripheral file size matches the size of the configuration device chosen.
Understanding the Design Steps for CvP Initialization Mode with the Revision Design Flow

The CvP initialization mode with the revision design flow allows you to create a reconfigurable core image that work with the single periphery image. The core image is stored in host memory. You download the core image to the FPGA using the PCI Express link. By using the Revision Design Flow, you can change the core image after the initial download to run alternate versions of the core logic.

You specify this mode in the Quartus Prime software by selecting the CvP Settings **Core initialization and update**. When the FPGA is fully programmed, the FPGA enters user mode. In user mode, you can reprogram the original static core image. The following are typical reasons to choose CvP initialization mode:

- To satisfy the PCIe initial power up requirement for plug-in cards if FPGA programming time exceeds this limit
- To save cost by storing the core image in external host memory
- To prevent unauthorized access to the core image by using encryption
- To change the core logic the following reasons:
  - To customize the core logic for different tasks
  - To provide periodic revisions for routine maintenance of the core logic

If you plan to create multiple versions of the core logic for the same periphery I/O, the new core images might not work with the previous periphery image. You can use the **CvP Revision Design Flow** to create reconfigurable images that connect to the same periphery image.
Figure 5-11: Design Flow for the CvP Initialization Mode with the Revision Design Flow

Note: When you select CvP initialization mode, you must use the CMU PLL and the hard reset controller for the PCI Express Hard IP.

The CvP initialization with the Revision Design Flow demonstration walkthrough includes the following steps:
1. Downloading and Generating the High Performance Reference Design on page 5-4
2. Workaround for a Known Issue with Transceiver Reconfiguration Controller IP Core on page 5-19
3. Creating an Alternate user_led.v File for the Reconfigurable Core Region on page 5-20
4. Setting up CvP Parameters for CvP Initialization Mode on page 5-7
5. Creating CvP Revisions of the Core Logic Region Using the CvP Revision Design Flow on page 5-24
6. Compiling both the Base and cvp_app Revisions in the CvP Revision Design Flow on page 5-27
7. Splitting the SOF File for the CvP Initialization Mode with the CvP Revision Design Flow on page 5-28
8. Bringing Up the Hardware on page 5-48

Related Information
- Transceiver PHY Reset Controller IP Core
- Transceiver PLL IP Core for Stratix V, Arria V, and Arria V GZ Devices

**Downloading and Generating the High Performance Reference Design**

Follow these steps to regenerate the PCI Express High Performance Reference Design with CvP enabled:

1. Download the **PCIe AVST and On-Chip Memory Interface** design files from the PCI Express Protocol web page. This design includes the correct pin assignments and project settings to target the Stratix V GX FPGA Development Kit.
2. Unzip **PCIe_SVGX_AVST_On_Chip_Mem_140.zip**.
3. Copy **hip_s5gx_x1_g1_ast64_140.qar** to your working directory.
4. Start the Quartus Prime software and restore **hip_s5gx_x1_g1_ast64_140.qar**.
5. On the Tools menu, select **Qsys**.
6. Open **top.qsys**.
7. On the **System Contents** tab, right-click **DUT** and select **Edit**.
8. Under **System Settings**, turn on **Enable configuration via the PCIe link** as shown in the following figure.
9. Click Finish.

10. On the Generation tab, specify the settings in the following table. Then click Generate at the bottom of the window.

Table 5-7: Qsys Generation Tab Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create simulation model</td>
<td>None</td>
</tr>
<tr>
<td>Create testbench Qsys system</td>
<td>None</td>
</tr>
<tr>
<td>Create testbench simulation model</td>
<td>None</td>
</tr>
<tr>
<td>Create HDL design files for synthesis</td>
<td>Verilog</td>
</tr>
<tr>
<td>Create block symbol file (.bsf)</td>
<td>Leave this entry off.</td>
</tr>
<tr>
<td>Path</td>
<td>&lt;working_dir&gt;/top</td>
</tr>
<tr>
<td>Simulation</td>
<td>Leave this entry blank.</td>
</tr>
<tr>
<td>Testbench</td>
<td>&lt;working_dir&gt;/top/synthesis</td>
</tr>
</tbody>
</table>
11. After successful compilation, close Qsys.
12. After creating an IP Variation, to add this IP to your Quartus project, you must manually add the .qip and .sip files.
   The .qip is located in `<working_dir>/synthesis/top.qip`
   The .sip is located in `<working_dir>/simulation/top.sip`
13. On the Assignments menu, select **Settings**.
14. In the **Files** category, remove the existing **top.qip** IP Variation File.
15. Browse to the new **top.qip** file created after generating the IP Core, located in `<working_dir>/synthesis/top.qip`.
16. Click **Add** and **OK** to close the Settings window.

**Figure 5-15: Settings Window**

---

**Related Information**

**PCI Express Reference Designs and Application Notes**

**Workaround for a Known Issue with Transceiver Reconfiguration Controller IP Core**

If you plan to perform almost continuous updates of the reconfigurable core logic in stress tests or in your actual system, you may encounter an issue with the Transceiver Reconfiguration Controller. This issue might cause the PCIe link to downtrain in Quartus II 13.0 release and earlier versions of the Quartus II software. If you are using Quartus II 13.0 SP1 or later versions of the Quartus II software, then you may not encounter this issue. Complete the following steps to avoid this issue:

1. **Open** `pcie_lib/top.v`
2. **Search** for the Reconfiguration Controller instance named `alt_xcvr_reconfig` and **comment out** the entire `reconfig_controller` in `top.v`. (The Transceiver Reconfiguration Controller instance includes 32 lines of Verilog HDL code.)
3. **Add** these 5 lines of Verilog HDL following after the commented out instance `alt_xcvr_reconfig`:

```verilog
wire [69:0] reconfig_to_xcvr_bus = {24'h0, 2'b11, 44'h0};
assign pcie_reconfig_driver_0_reconfig_mgmt_waitrequest = 1'b0;
```
assign pcie_reconfig_driver_0_reconfig_mgmt_readdata = 32'h0;
assign alt_xcvr_reconfig_0_reconfig_busy_reconfig_busy = 1'b0;
assign alt_xcvr_reconfig_0_reconfig_to_xcvr_reconfig_to_xcvr = { 2 {reconfig_to_xcvr_bus}};

In this example, the first statement hardwires the reconfig_to_xcvr_bus to the correct values per channel. The first three assignment statements specify the correct values for the waitrequest, readdata, reconfig_busy signals. The final assignment statement for alt_xcvr_reconfig_0_reconfig_to_xcvr_reconfig_to_xcvr represents the full reconfiguration bus for all active transceiver channels. This bus is replicated 2 times because 2 channels are active in the Gen1 x1 instance.

Creating an Alternate user_led.v File for the Reconfigurable Core Region

This example design creates a new version of the PCI Express High Performance Reference Design. The original version of this reference design includes an LED which turns on whenever the Link Training and Status and State Machine (LTSSM) enters the Polling. Compliance state (0x3). The alternate version of user_led.v turns on the LED based on the counter. The LED is instantiated as a separate module in the High Performance Reference Design to demonstrate the steps necessary to create a design with multiple versions of the core logic.

Complete the following steps to create the alternate version of the High Performance Reference Design:

2. Open and unzip user_led.zip.
3. Copy user_led.v and top_hw.v to your working directory.
   This version of user_led.v turns on when the Link Training and Status and State Machine (LTSSM) enters the Polling. Compliance state (0x3). top_hw.v is the top-level wrapper for the PCI Express High Performance Reference Design. It instantiates user_led.v as a separate module.
4. Move or copy the cvp_app_src to a subdirectory of your working directory.
   This alternate version of user_led.v turns on the LED whenever bit[23] of a counter is one.

Setting up CvP Parameters for CvP Initialization Mode

Follow these steps to specify CvP parameters using the Quartus Prime software:

1. On the Quartus Prime Assignments menu, select Device, and then click Device and Pin Options.
2. Under Category select General, and then enable following options:
   a. Auto-restart configuration after error. If this option is enabled, CvP restarts after an error is detected.
   b. Enable autonomous PCIe HIP mode.
      Checking this box has no affect if you have enabled CvP by turning on Enable Configuration via the PCIe link in the Hard IP for PCI Express GUI, hence Altera recommends to keep it unchecked.
   c. Leave all other options disabled.
3. Under **Category**, select **Configuration** to specify the configuration scheme and device. Specify the settings in the following table:

### Table 5-8: CvP Initialization Mode Configuration Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration scheme</td>
<td>Active Serial x4</td>
</tr>
<tr>
<td>Configuration mode</td>
<td>Standard</td>
</tr>
<tr>
<td>Configuration device</td>
<td>EPCQ256</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Configuration device I/O voltage</strong></td>
<td>Auto</td>
</tr>
<tr>
<td><strong>Force VCCIO to be compatible with configuration I/O voltage</strong></td>
<td>Leave this option off.</td>
</tr>
<tr>
<td><strong>Generate compressed bitstreams</strong></td>
<td>Turn this option off.</td>
</tr>
<tr>
<td><strong>Active serial clock source</strong></td>
<td>100 MHz Internal Oscillator</td>
</tr>
<tr>
<td><strong>Enable input tri-state on active configuration pins in user mode</strong></td>
<td>Leave this option off.</td>
</tr>
</tbody>
</table>

Because this is a small example design, it does not use a compressed bitstream. For larger designs, using a compressed bitstream significantly reduces configuration time. In addition, a compressed bitstream requires a smaller flash device.
These Configuration settings use the configuration devices available on the Stratix V GX FPGA Development Board. The EPCQ256 flash device is far larger than required to load a periphery image.

4. Under Category select CvP Settings. For CvP Initialization mode, specify the following settings in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CvP Initialization</td>
<td>Core initialization and update</td>
</tr>
<tr>
<td>Enable CvP_CONFDONE pin</td>
<td>Turn this option on.</td>
</tr>
<tr>
<td>Enable open drain on CvP_CONFDONE pin</td>
<td>Turn this option on.</td>
</tr>
</tbody>
</table>
5. Click OK to close the Device and Pin Options dialog box.
6. Click OK to close the Device dialog box.
7. Save your project.

Creating CvP Revisions of the Core Logic Region Using the CvP Revision Design Flow

This section provides the instructions to create CvP revisions for the reconfigurable core logic region that can be updated. The remainder of the design is treated as a static core region.

Follow these steps to create the base version of the core logic:

1. On the Assignments menu, select Settings and then select Files.
2. In the File name box, Browse and select user_led.v, then click Add.
3. Click OK.
4. Run Analysis & Synthesis so that the Quartus Prime software parses the design to create a design hierarchy that includes the user_led instance.
5. To set userLed as a design partition, right click userLed:userLed in the design hierarchy and select Design Partition. A small red box appears next to userLed:userLed indicating that it is a separate
partition. (If you perform the same steps again, you remove the separate design partition from user_led: user_led.) The following image illustrates this step.

**Figure 5-19: Setting a Design Partition**

6. Click the **Design Partitions Window** at the bottom of the menu cascade shown in the figure above. The **Design Partitions Window** appears.

7. To add the **Allow Multiple Personas** column to the **Design Partitions Window**, right click on the top bar of **Design Partition Window** next to the **Color** heading and select **Allow Multiple Personas** from the list as shown in the following figure.
8. Click the core instance user_led:userLed and set **Allow Multiple Personas** to **On**.
9. Click in the **Netlist Type** column and set the **user_led:userLed Netlist Type** to **Source File**.
10. Follow these steps to create a CvP revision for the modified project.

   a. Under the **Revisions** tab, right click on the Revision **top** and select **Create CvP Revision**. The **Create CvP Revision** dialog box appears.

   b. For the **Revision Name** type **cvp_app** and click **OK** to create a CvP revision as illustrated in the following figure.
c. Save the Quartus Prime project.

11. Change the CvP revision in the Quartus Prime software design revision list as shown in the following figure.

Figure 5-22: Changing the CvP Revision

12. To remove `user_led.v` from the `cvp_app` revision, on the Assignments menu, select Settings, then select Files.

This is the original `user_led.v` file that turns on the LED when the LTSSM enters the Polling.Compliance state.

13. In the Files list, click `user_led.v`, then click Remove.

14. To add `cvp_app_src/user_led.v` for the `cvp_app` revision, in the File name box, click Browse and browse to `cvp_app_src/user_led.v`, then click Add.

This is the modified `user_led.v` file that turns on the LED when the bit[23] of a counter is one.

15. In the File name box, click Browse and browse to `cvp_app_src/user_led.v`, then click Add.

16. Click OK.

17. In the Partition Name window, select `user_led:user_led` and change the Netlist Type to Source File and turn On Allow Multiple Personas.

18. Change back to the top revision in the Quartus Prime software design revision list.

Compiling both the Base and cvp_app Revisions in the CvP Revision Design Flow

To compile your project, click Compile All. Using Compile All guarantees that the Quartus Prime software compiles the top revision and cvp_app revision in the correct order.
You might need to go through a few iterations of compiling your designs to separate all of the periphery components from the core logic. As a result, the final design might not maintain the functional relationships between logic blocks that you originally planned. Adding extra comments in your design will help you to trace the HDL.

You must compile your project to update the reconfigured core image if any of the following conditions are true:

- The CvP revision has never been compiled.
- You have changed the periphery logic.
- You have changed the wrapper file for any of the core revisions.
- You have migrated to a new version of the Quartus Prime software.
- You have changed any project settings in the Quartus Prime Settings File (*.qsf).

### Splitting the SOF File for the CvP Initialization Mode with the CvP Revision Design Flow

To implement the **CvP Revision Design Flow** in CvP initialization mode, you must replace the base .sof (`top.sof`) with the revision .sof (`cvp_app1.sof`). You must also specify a different file name for the CvP revision. This example uses `cvp_app.jic`. The periphery and the core images created for the CvP revision are `cvp_app.periph.jic` and `cvp_app.core.rbf`, respectively. Follow these steps to create the periphery and core images for the CvP revision:

1. On the File menu, select **Convert Programming File**. Under **Output programming file** specify the options in the following table. These settings are illustrated in the figure below.

#### Table 5-10: CvP Revision Design Flow: Output Programming File Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming file type</td>
<td><strong>JTAG Indirect Configuration File (.jic)</strong></td>
</tr>
<tr>
<td>Configuration device</td>
<td><strong>EPCQ256</strong></td>
</tr>
<tr>
<td>Mode</td>
<td><strong>Active Serial x4</strong></td>
</tr>
<tr>
<td>File name</td>
<td>Click <strong>browse</strong> and specify <code>pcie_quartus_files/cvf_app.jic</code></td>
</tr>
</tbody>
</table>

---

5-28  Splitting the SOF File for the CvP Initialization Mode with the CvP Revision Design Flow

Figure 5-23: Compile Both Base and cvp_app Revisions Using the Compile All Option in Quartus 13.0
2. Under **Input files to convert** specify the options in the following table:

**Table 5-11: CvP Revision Design Flow: Input Files to Convert**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash Loader</td>
<td>Click <strong>Add Device</strong> and select <strong>Stratix V</strong> and then 5SGXEA7K2.</td>
</tr>
<tr>
<td>Click SOF Data</td>
<td>Click <strong>Add File</strong> and navigate to <em>/pcie_quartus_files/cvp_app.sof</em>. If you specified a compressed or encrypted bitstream in the <strong>Device and Pin Options</strong> dialog box, you must specify the same options in the <strong>Conversion Programming File</strong> window. To enable these settings, click <strong>cvp_app.sof</strong>, then click <strong>Properties</strong> and check the appropriate boxes.</td>
</tr>
</tbody>
</table>

3. Turn on the **Create CvP files (Generate cvp_app1.periph.jic and cvp_app1.core.rbf)** parameter in the **Output Programming Files** section.

4. To save the **Conversion Programming File** parameters, click **Save Conversion Setup** and type an output file name. Saving your conversion setup saves time on subsequent conversions. For this exercise, call your settings **cvp_revision.cof**. You can reload the conversion parameters by opening the Conversion Setup File (*.cof) using **Open Conversion Setup Data** in the **Convert Programming File** window. The following figure illustrates these options.

5. Click **Generate**. This conversion process creates the alternate core logic, **cvp_app.core.rbf**, and the periphery logic, **cvp_app.periph.jic**. The periphery logic, **cvp_app.periph.jic**, should be identical to the periphery logic created when splitting the base revision.

**Note:** The generated CvP peripheral file size matches the size of the configuration device chosen.
6. Proceed to **Bringing Up the Hardware** on page 5-48.

The file `top.cof` provided in *_cvp* designs is a template for CvP initialization mode. You can open this file in the **Open Conversion Setup Data** of **Convert Programming File** window to retrieve the parameters shown in figure above.
Understanding the Design Steps for CvP Update Mode

CvP update mode divides the design into periphery and core images (as the previous modes). Initially, you program the entire image (both periphery and core) using conventional programming options. Subsequently, you can download alternative versions of the core image using the PCI Express link.

You specify this mode in the Quartus Prime software by selecting the CvP Setting **Core update**. The following figure provides the high-level steps for CvP update mode.

**Figure 5-25: Design Flow for CvP Update Mode**
Note: When you select CvP update mode, you must use the CMU PLL and the hard reset controller for the PCI Express Hard IP.

The CvP update demonstration walkthrough includes the following steps:

1. Downloading and Generating the High Performance Reference Design on page 5-4
2. Workaround for a Known Issue with Transceiver Reconfiguration Controller IP Core on page 5-19
3. Creating an Alternate user_led.v File for the Reconfigurable Core Region on page 5-20
4. Setting up CvP Parameters for CvP Update Mode on page 5-36
5. Creating CvP Revisions of the Core Logic Region Using the CvP Revision Design Flow on page 5-24
6. Compiling the Design for the CvP Update Mode on page 5-43
7. Splitting the SOF File for the CvP Update Design Mode on page 5-44
8. Splitting the SOF File for the CvP Update Mode with the CvP Revision Design Flow on page 5-45
9. Bringing Up the Hardware on page 5-48

By default, once the FPGA enters user mode, you can only reprogram the original static core image. If you want to have multiple core images in user mode, you can use the CvP Revision Design Flow to create multiple core images that connect to the same periphery image.

Related Information
- Transceiver PHY Reset Controller IP Core
- Transceiver PLL IP Core for Stratix V, Arria V, and Arria V GZ Devices

Downloading and Generating the High Performance Reference Design

Follow these steps to regenerate the PCI Express High Performance Reference Design with CvP enabled:

1. Download the PCIe AVST and On-Chip Memory Interface design files from the PCI Express Protocol web page. This design includes the correct pin assignments and project settings to target the Stratix V GX FPGA Development Kit.
3. Copy hip_s5gx_x1_g1_ast64_140.qar to your working directory.
4. Start the Quartus Prime software and restore hip_s5gx_x1_g1_ast64_140.qar.
5. On the Tools menu, select Qsys.
6. Open top.qsys.
7. On the System Contents tab, right-click DUT and select Edit.
8. Under System Settings, turn on Enable configuration via the PCIe link as shown in the following figure.
9. Click Finish.

10. On the Generation tab, specify the settings in the following table. Then click Generate at the bottom of the window.

Table 5-12: Qsys Generation Tab Settings

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create simulation model</td>
<td>None</td>
</tr>
<tr>
<td>Create testbench Qsys system</td>
<td>None</td>
</tr>
<tr>
<td>Create testbench simulation model</td>
<td>None</td>
</tr>
<tr>
<td>Create HDL design files for synthesis</td>
<td>Verilog</td>
</tr>
<tr>
<td>Create block symbol file (.bsf)</td>
<td>Leave this entry off.</td>
</tr>
<tr>
<td>Path</td>
<td>&lt;working_dir&gt; top</td>
</tr>
<tr>
<td>Simulation</td>
<td>Leave this entry blank.</td>
</tr>
<tr>
<td>Testbench</td>
<td>&lt;working_dir&gt;/top/synthesis</td>
</tr>
</tbody>
</table>
11. After successful compilation, close Qsys.

12. After creating an IP Variation, to add this IP to your Quartus project, you must manually add the .qip and .sip files.

   The .qip is located in `<working_dir>/synthesis/top.qip`
   The .sip is located in `<working_dir>/simulation/top.sip`

Figure 5-28: Intel® Quartus® Prime Reminder
13. On the Assignments menu, select **Settings**.
14. In the **Files** category, remove the existing **top.qip** IP Variation File.
15. Browse to the new **top.qip** file created after generating the IP Core, located in `<working_dir>/synthesis/top.qip`.
16. Click **Add** and **OK** to close the Settings window.

**Figure 5-29: Settings Window**

---

**Related Information**

**PCI Express Reference Designs and Application Notes**

**Workaround for a Known Issue with Transceiver Reconfiguration Controller IP Core**

If you plan to perform almost continuous updates of the reconfigurable core logic in stress tests or in your actual system, you may encounter an issue with the Transceiver Reconfiguration Controller. This issue might cause the PCIe link to downtrain in Quartus II 13.0 release and earlier versions of the Quartus II software. If you are using Quartus II 13.0 SP1 or later versions of the Quartus II software, then you may not encounter this issue. Complete the following steps to avoid this issue:

1. Open **pcie_lib/top.v**.
2. Search for the Reconfiguration Controller instance named `alt_xcvr_reconfig` and comment out the entire `reconfig_controller` in **top.v**. (The Transceiver Reconfiguration Controller instance includes 32 lines of Verilog HDL code.)
3. Add these 5 lines of Verilog HDL following after the commented out instance `alt_xcvr_reconfig`:
   ```verilog
   wire [69:0] reconfig_to_xcvr_bus = {24'h0, 2'b11, 44'h0};
   assign pcie_reconfig_driver_0_reconfig_mgmt_waitrequest = 1'b0;
   ```

---

**CvP Example Designs**

Altera Corporation

---

**Send Feedback**
assign pcie_reconfig_driver_0_reconfig_mgmt_readdata = 32’h0;
assign alt_xcvr_reconfig_0_reconfig_busy_reconfig_busy = 1’b0;
assign alt_xcvr_reconfig_0_reconfig_to_xcvr_reconfig_to_xcvr_r = { 2[reconfig_to_xcvr_bus]};

In this example, the first statement hardwires the reconfig_to_xcvr_bus to the correct values per channel. The first three assignment statements specify the correct values for the waitrequest, readdata, reconfig_busy signals. The final assignment statement for alt_xcvr_reconfig_0_reconfig_to_xcvr_reconfig_to_xcvr represents the full reconfiguration bus for all active transceiver channels. This bus is replicated 2 times because 2 channels are active in the Gen1 x1 instance.

Creating an Alternate user_led.v File for the Reconfigurable Core Region

This example design creates a new version of the PCI Express High Performance Reference Design. The original version of this reference design includes an LED which turns on whenever the Link Training and Status and State Machine (LTSSM) enters the Polling. Compliance state (0x3). The alternate version of user_led.v turns on the LED based on the counter. The LED is instantiated as a separate module in the High Performance Reference Design to demonstrate the steps necessary to create a design with multiple versions of the core logic.

Complete the following steps to create the alternate version of the High Performance Reference Design:

2. Open and unzip user_led.zip.
3. Copy user_led.v and top_hw.v to your working directory.
   - This version of user_led.v turns on when the Link Training and Status and State Machine (LTSSM) enters the Polling. Compliance state (0x3). top_hw.v is the top-level wrapper for the PCI Express High Performance Reference Design. It instantiates user_led.v as a separate module.
4. Move or copy the cvp_app_src to a subdirectory of your working directory.
   - This alternate version of user_led.v turns on the LED whenever bit[23] of a counter is one.

Setting up CvP Parameters for CvP Update Mode

Follow these steps to specify CvP parameters using the Quartus Prime software:

1. On the Assignments menu, select Device, and then click Device and Pin Options.
2. Under Category first select General, and then enable following options:
   a. Auto-restart configuration after error. If this option is enabled, CvP restarts after an error is detected.
   b. Enable autonomous PCIe HIP mode.
      - Checking this box has no affect if you have enabled CvP by turning on Enable Configuration via the PCIe link in the Hard IP for PCI Express GUI, hence Altera recommends to keep it unchecked.
   c. Leave all other options disabled.
3. Under **Category** select **Configuration** to specify the configuration scheme and device. Specify the settings in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration scheme</td>
<td>Passive Serial</td>
</tr>
<tr>
<td>Configuration mode</td>
<td>Standard</td>
</tr>
<tr>
<td>Configuration device</td>
<td>Auto</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Configuration device I/O voltage</td>
<td>Auto</td>
</tr>
<tr>
<td>Force VCCIO to be compatible with configuration I/O voltage</td>
<td>Leave this option off.</td>
</tr>
<tr>
<td>Generate compressed bitstreams</td>
<td>Leave this option on.</td>
</tr>
<tr>
<td>Active serial clock source</td>
<td>100 MHz Internal Oscillator</td>
</tr>
<tr>
<td>Enable input tri-state on active configuration pins in user mode</td>
<td>Leave this option off.</td>
</tr>
</tbody>
</table>
4. Under **Category** select **CvP Settings**. Specify the settings in the following table:

**Table 5-14: CvP Update Category Settings**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CvP via Protocol</td>
<td>Core update</td>
</tr>
<tr>
<td>Enable CvP_CONFDONE pin</td>
<td>Turn this option on.</td>
</tr>
<tr>
<td>Enable open drain on CvP_CONFDONE pin</td>
<td>Turn this option on.</td>
</tr>
</tbody>
</table>
5. Click OK to close the Device and Pin Options dialog box.
6. Click OK to close the Device dialog box.
7. Save your project.

Creating CvP Revisions of the Core Logic Region Using the CvP Revision Design Flow

This section provides the instructions to create CvP revisions for the reconfigurable core logic region that can be updated. The remainder of the design is treated as a static core region.

Follow these steps to create the base version of the core logic:

1. On the Assignments menu, select Settings and then select Files.
2. In the File name box, Browse and select user_led.v, then click Add.
3. Click OK.
4. Run Analysis & Synthesis so that the Quartus Prime software parses the design to create a design hierarchy that includes the user_led instance.
5. To set user_led as a design partition, right click `user_led:user_led` in the design hierarchy and select Design Partition. A small red box appears next to `user_led:user:led` indicating that it is a separate partition. (If you perform the same steps again, you remove the separate design partition from `user_led:user:led`.) The following image illustrates this step.

![Figure 5-33: Setting a Design Partition](image)

6. Click the Design Partitions Window at the bottom of the menu cascade shown in the figure above. The Design Partitions Window appears.

7. To add the Allow Multiple Personas column to the Design Partitions Window, right click on the top bar of Design Partition Window next to the Color heading and select Allow Multiple Personas from the list as shown in the following figure.
8. Click the core instance `user_led:user_led` and set **Allow Multiple Personas** to **On**.

9. Click in the **Netlist Type** column and set the `user_led:user_led Netlist Type` to **Source File**.

10. Follow these steps to create a CvP revision for the modified project.

   a. Under the **Revisions** tab, right click on the Revision **top** and select **Create CvP Revision**. The **Create CvP Revision** dialog box appears.

   b. For the **Revision Name** type `cvp_app` and click **OK** to create a CvP revision as illustrated in the following figure.
c. Save the Quartus Prime project.

11. Change the CvP revision in the Quartus Prime software design revision list as shown in the following figure.

Figure 5-36: Changing the CvP Revision

12. To remove user_led.v from the cvp_app revision, on the Assignments menu, select Settings, then select Files.

This is the original user_led.v file that turns on the LED when the LTSSM enters the Polling.Compliance state.

13. In the Files list, click user_led.v, then click Remove.

14. To add cvp_app_src/user_led.v for the cvp_app revision, in the File name box, click Browse and browse to cvp_app_src/user_led.v, then click Add.

This is the modified user_led.v file that turns on the LED when the bit[23] of a counter is one.

15. In the File name box, click Browse and browse to cvp_app_src/user_led.v, then click Add.

16. Click OK.

17. In the Partition Name window, select user_led:user_led and change the Netlist Type to Source File and turn On Allow Multiple Personas.

18. Change back to the top revision in the Quartus Prime software design revision list.

Compiling the Design for the CvP Update Mode

1. To compile the design, on the Processing menu, select Start compilation. Compilation creates a .pof file in the pcie_quartus_files subdirectory.
You might need to go through a few iterations of compiling your designs to separate all of the periphery components from the core logic. As a result, the final design might not maintain the functional relationships between logic blocks that you originally planned. Adding extra comments in your design will help you to trace the HDL.

**Splitting the SOF File for the CvP Update Design Mode**

Follow these steps to split your design file into periphery and core images for CvP update mode. You use the core image, `top.core.rbf`, to perform CvP updates.

1. On the File menu, select **Convert Programming File**.
2. Under **Output programming file** specify the options in the following table. These options are illustrated in the figure below.

**Table 5-15: Output Programming File**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming file type</td>
<td>Programmer Object File (.pof)</td>
</tr>
<tr>
<td>Configuration device</td>
<td>CFI_128Mb</td>
</tr>
<tr>
<td>Mode</td>
<td>1-bit Passive Serial</td>
</tr>
<tr>
<td>File name</td>
<td>Click browse and specify <code>pcie_quartus_files/top.pof</code></td>
</tr>
<tr>
<td>Create Memory Map File</td>
<td>Turn this option on.</td>
</tr>
<tr>
<td>Create CvP files</td>
<td>Turn this option on. This box is greyed out until you specify the SOF Data file under <strong>Input files to convert</strong>.</td>
</tr>
</tbody>
</table>

3. Under **Input files to convert** specify the options in the following table:

**Table 5-16: Input files to convert**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click SOF Data</td>
<td>Click <strong>Add File</strong> and navigate to <code>./pcie_quartus_files/top.sof</code>. If you specified a compressed or encrypted bitstream in the <strong>Device and Pin Options</strong> dialog box, you must specify the same options in the <strong>Conversion Programming File</strong> window. To enable these settings, click <strong>top.sof</strong>, then click <strong>Properties</strong> and check the appropriate boxes.</td>
</tr>
</tbody>
</table>

4. Turn on the **Create CvP files** (Generate `top.periph.jic` and `top.core.rbf`) parameter in the **Output Programming Files** section.
5. Click **Generate** to create `top.periph.pof`, and `top.core.rbf`. The periphery file, `top.periph.pof` is generated, but it is not used.
Figure 5-37: Base Revision of CvP Update Mode: Convert Programming File Settings

Note: The Configuration scheme and Configuration device in Device and Pin Options must match with Configuration Device and Mode in Convert Programming File respectively.

Splitting the SOF File for the CvP Update Mode with the CvP Revision Design Flow

To implement the CvP Revision Design Flow with CvP update mode, you must replace the base .sof (top.sof) with the revision .sof (cvp_app.sof). You must also specify a different file name for the CvP revision. This example uses cvp_app. The periphery and the core images created for the CvP revision are cvp_app.periph.pof and cvp_app.core.rbf, respectively. Follow these steps to create the periphery and core images for the CvP revision:

1. On the File menu, select Convert Programming File. Under Output programming file specify the options in the following table. These settings are illustrated in the figure below.

Table 5-17: CvP Revision Design Flow: Output Programming File Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming file type</td>
<td>Programmer Object File (.pof)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Configuration device</td>
<td>CFL_128Mb</td>
</tr>
<tr>
<td>Mode</td>
<td>1-bit Passive Serial</td>
</tr>
<tr>
<td>File name</td>
<td>Click browse and specify <code>pcie_quartus_files/cvp_app.pof</code>.</td>
</tr>
<tr>
<td>Create Memory Map File</td>
<td>Turn this option on.</td>
</tr>
<tr>
<td>Create CvP files</td>
<td>Turn this option on. This box is greyed out until you specify the SOF Data file under Input files to convert.</td>
</tr>
</tbody>
</table>

2. Under **Input files to convert** specify the options in the following table:

### Table 5-18: CvP Revision Design Flow: Input Files to Convert

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click SOF Data</td>
<td>Click Add File and navigate to <code>./pcie_quartus_files/cvp_app.sof</code>. If you specified a compressed or encrypted bitstream in the Device and Pin Options dialog box, you must specify the same options in the Conversion Programming File window. To enable these settings, click <code>cvp_app.sof</code>, then click Properties and check the appropriate boxes.</td>
</tr>
</tbody>
</table>

3. Turn on the Create CvP files (Generate `cvp_app.periph.jic` and `cvp_app.core.rbf`) parameter in the Output Programming Files section.

4. To save the Conversion Programming File parameters, click Save Conversion Setup and type an output file name. Saving your conversion setup saves time on subsequent conversions. For this exercise, call your settings `cvp_update_revision_setup.cof`. You can reload the conversion parameters by opening the Conversion Setup File (`*.cof`) using Open Conversion Setup Data in the Convert Programming File window. The following figure illustrates these options.

5. Click Generate. This conversion process creates the alternate core logic, `cvp_app.core.rbf`, and the periphery logic, `cvp_app.periph.pof`.

**Note:** The generated CvP peripheral file size matches the size of the configuration device chosen.
Figure 5-38: CvP Revision Design Flow: Convert Programming Files

6. Proceed to **Bringing Up the Hardware** on page 5-48.

The file `top.cof` provided in *cvp* designs is a template for CvP initialization mode. You can open this file in the **Open Conversion Setup Data** of **Convert Programming File** window to retrieve the parameters shown in figure above.
Bringing Up the Hardware

Before testing the design in hardware, you must install the Jungo WinDriver or CvP driver in your DUT system. You can also install RW Utilities or other system verification tools to monitor the link status of the Endpoint and to observe traffic on the link. You can download these utilities for free from many web sites.

Note: Intel® does not provide Jungo WinDriver as a part of Intel Quartus® Prime install package starting with software version 19.1. You can develop your own custom CvP driver for Linux using the sample Linux driver source code provided by Intel.

Note: The Linux driver provided by Intel is not a production driver. You must adapt this driver to your design’s strategy.

The test setup includes the following components:

- Stratix V GX FPGA Development Kit
- USB Blaster
- A DUT PC with PCI Express slot to plug in the Stratix V GX FPGA Development Kit
- A host PC running the Quartus Prime software to program the periphery image, .sof or .pof file

Although a separate host PC is not strictly necessary, it makes testing less cumbersome.

Installing Jungo WinDriver in Windows Systems

1. Navigate to <Quartus Prime installation path>\quartus\drivers\wdrvr\windows<32 or 64>.
2. Run the command:
   - wdreq -inf windrvr6.inf install
3. Copy the wdapi1021.dll file to the %windir%\system32 directory.

Installing Jungo WinDriver in Linux Systems

1. Navigate to <Quartus Prime installation path>/quartus/drivers/wdrvr/Linux<32 or 64>.
2. Run the following commands:
   a. ./configure --disable-usb-support
   b. make
   c. su
   d. make install
3. You can change the permissions for the device file. For example, chmod 666 /dev/windrvr6.
4. For 64-bit Linux systems, set the Quartus_64BIT environment variable before you run quartus_cvp using the following command:
   - export QUARTUS_64BIT=1
5. You can use the quartus_cvp command to download *core .rbf files to your FPGA. The following table lists the quartus_cvp commands for all modes.
Table 5-19: Syntax for quartus_cvp Commands

<table>
<thead>
<tr>
<th>Mode</th>
<th>quartus_cvp Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>quartus_cvp --vid=&lt;Vendor ID&gt; --did=&lt;Device ID&gt; &lt;Core .rbf file path&gt;</td>
</tr>
<tr>
<td>Unencrypted</td>
<td>quartus_cvp --vid=&lt;Vendor ID&gt; --did=&lt;Device ID&gt; &lt;Core .rbf file path&gt;</td>
</tr>
<tr>
<td>Compressed</td>
<td>quartus_cvp -c --vid=&lt;Vendor ID&gt; --did=&lt;Device ID&gt; &lt;Core .rbf file path&gt;</td>
</tr>
<tr>
<td>Encrypted</td>
<td>quartus_cvp -e --vid=&lt;Vendor ID&gt; --did=&lt;Device ID&gt; &lt;Core .rbf file path&gt;</td>
</tr>
<tr>
<td>Compressed and encrypted</td>
<td>quartus_cvp -c -e --vid=&lt;Vendor ID&gt; --did=&lt;Device ID&gt; &lt;Core .rbf file path&gt;</td>
</tr>
</tbody>
</table>

Installing Open Source CvP Driver in Linux System

1. Download the open source CvP driver from the CvP Driver.
2. Navigate to the driver directory.
3. Unzip the driver by typing the following command:
   ```
   tar -zxvf .<driver>.tar.bz2
   ```
4. Run the installation by typing the following commands:
   ```
   sudo make
   sudo make install
   ```
5. Once the installation completed successfully, it generates the altera_cvp file in the following location: directory/dev/altera_cvp.

Modifying MSEL for Active Serial x4 Flash on Stratix V Dev-Kit

The MSEL switch labeled SW4 on the back of the Stratix V GX FPGA Development Kit PCB specifies the flash type. The correct setting for an active serial x4 flash is 5’b10010 as shown in the following figure. The factory default value is 5’b01000.
In this figure, the switch head is outlined by a green rectangle. The up position signifies logic zero and the down position signifies logic one. The MSB of the switch, SW4[6], is on the far right. This bit is unused and must be set to zero (SW4[6]=up). The MSB bit of MSEL[4] is position 5, the second bit from the right. To set the unused bit to 0 and MSEL[4:0] = 5'b10010, the SW4[6:1] sequence is up(0), down(1), up(0), up(0), down(1), up(0), reading from right to left.

**Related Information**
- Stratix V GX FPGA Development Kit User Guide

**Programming CvP Images and Validating the Link**

For CvP initialization mode, you must load the periphery image (top.periph.jic) and then download the core image core image (top.core.rbf) using the PCIe Link. You can use JTAG to load different programming files (i.e. .sof/.jic/periph.pof) into your selected CvP initialization enabled Stratix V, Cyclone V, or Arria V device.

After loading the periphery image via the JTAG port, the link should reach the expected data rate and link width. You can confirm the PCIe link status using the RW Utilities. Then, you can update the core image (top.core.rbf) using the quartus_cvp command over the PCIe link.

For CvP update mode, you first program the FPGA using the .sof or .pof image. After the programming completes the FPGA enters user mode. You can now reconfigure the core image (.core.rbf), the quartus_cvp command or your own driver.

Follow these steps to program and test the CvP functionality:

1. Plug in the Stratix V GX FPGA Development Kit to the PCI Express slot of the DUT PC and power it on. Altera recommends that you use the external power supply that the development kit includes.
2. On the host PC, open the Quartus Prime Tools menu and select **Programmer**. The Programmer appears.
3. Click **Auto Detect** to verify that the USB Blaster recognizes the Stratix V FPGA.

4. Follow these steps to program the periphery image:
   a. Select Stratix V device, then right click **None** under **File** column.
   b. Navigate to `pcie_quartus_files/topPeriph.jic` and click **Open**.
   c. Under **Program/Configure** column, select **5SGXEA7K2** and **EPCQ256**.
   d. Click **Start** to program the periphery image to EPCQ256 flash.

5. To force the host PC to re-enumerate the link with the new image, reboot/power cycle the DUT PC.

6. You can use RW Utilities or another system software driver to verify the link status. The following figure shows that the RW Utilities enumeration includes an Altera PCIe on Bus 01.
7. You can also confirm expected link speed and width. For this Gen1 x1 example design, the following figure shows that both Altera EP Link Capability Register at 0x8C and Link Status register at 0x92 have value 0x11, which confirms the link successfully comes up as Gen1 x1.
Follow these steps to program the core image (top.core.rbf) into the FPGA:

a. Open a DOS command window.

b. Change to appropriate Quartus Prime bin install directory. Both 64-bit and 32-bit bin directories are available. This example uses C:\altera\13.0\quartus\bin64.

c. Type the following command to program the core image. The value of Vendor ID (vid) and Device ID (vid) are in hexadecimal and must match the values you specified on the Device Identification Registers tab of the Stratix V Hard IP for PCI Express IP Core GUI:

```
quartus_cvp --vid=1172 --did=e001 <path>/top.core.rbf
```

d. The figure below shows the results of successful CvP programming.
If you implement your own software driver to program the core image, refer to the CvP Driver Support section in Chapter 6 of CvP User Guide.

If you are using the OpenSource Linux driver, perform the following steps to program the core image:

1. Copy the .core.rbf file to your working directory.
2. Open a console in Linux. Change directory to the same mentioned above where the file is copied.
3. Program the core image by typing the following command:

   ```bash
   cp *.core.rbf /dev/altera_cvp
   ```

You are now ready to run your own tests.

Related Information

CvP Driver Support on page 6-1
CvP Debugging Check List

1. Check the PCIe configuration to ensure that it supports CvP. For instance, Stratix V does not support Gen3 CvP.
2. Confirm that the current Quartus Prime software version supports CvP.
3. Check that your physical pin assignments support CvP. For Example, for Stratix V, only the bottom left Stratix V Hard IP for PCI Express supports CvP. Other Hard IP cores in the same device do not support CvP.
4. Check the PCB connections for PERST# and refclk.
5. Make sure the reset and clock connections to the Transceiver Reconfiguration Controller IP Core are correct. For CvP mode, you must use refclk to drive reconfig_clk. PERST# must drive reconfig_reset to the Hard IP for PCI Express IP Core.
6. Confirm that the design meets timing constraints for setup, hold time, and recovery for multi-corners.
7. Check that the test_in bus is hardwired to 0xA8. The following test_in signals are most important when debugging:
   b. Setting test_in[5]=1 prevents the core from entering the Compliance Mode.
   c. Setting test_in[3]=1 indicates that the Hard IP for PCI Express is implemented in an FPGA.
   d. Setting test_in[0] = 0 uses timeout and counter values which meet the PCIe specifications. Otherwise the shorter, non-compliant simulation values are used.
8. Disable power management support in the host BIOS settings.
9. Confirm that the Vendor ID and Device ID arguments specified as arguments to the quartus_cvp.exe command match the values specified in the Hard IP for PCI Express IP Core GUI.
10. If you are designing an open system, test with different PCs and compare the results.
11. If the first CvP update fails, check that the correct quartus_cvp.exe command is used. For 32-bit systems, you must use .\quartus\bin\quartus_cvp.exe. For 64-bit systems, the correct quartus_cvp version is .\quartus\bin64.
12. Before executing the quartus_cvp command, make sure that the Memory Space Enable bit is set in the Command register of PCI Express Configuration Space. If the BIOS does not enable this bit, you must use a system tool such as RW Utilities to write a value of 0x0006 to the Command register at offset 0x4. Writing this value will set both Memory Space Enable and Master Bus Enable bits.
13. If encryption or compression is enabled, disable them and retry. Record the symptoms.
14. Check if the design works after configuring the FPGA with the SOF via JTAG and then doing a warm reboot. The .sof file contains both periphery image and the core image; consequently, this test will not determine which type of image causes the failure.
15. Use a PCI Express analyzer to capture the PCIe trace of the failing scenarios. Observe the transitions of LTSSM if it fails to get insight into the link failures.
16. Use the Power On Trigger of the SignalTap II Embedded Logic Analyzer and record the LTSSM transitions. Determine whether the LTSSM goes to L0 (0xF) or toggles between Detect states and Polling states.
17. Disable the Transceiver Reconfiguration Controller and hardwire other inputs to zero, except the reconfig_to_xcvr() bus which requires bit[44] of each channel to be driven high. The remaining bit of reconfig_to_xcvr() are tied to low. The sample file is top_wo_reconfig.v under ./altera_pcie_cvp/hw_devkit_ed directory.
18. If CvP fails, try a similar Altera CvP example design to determine if the symptoms remain the same. If the failure still persists, try a non-CvP design and take note of the differences.
19. Determine if the example CvP design with similar configuration works on the same platform.
20. If you have tried all these suggestions and your design is still not working, file a Service Request (SR).
   In your SR, include the following information:
a. Describe what you have tried and the results of your tests.
b. List the Quartus Prime software version, the target device, information about the system under test, and what CvP modes are being used.
c. Specify where the failure occurs. Does it occur after loading the periphery, on the first CvP update, or on subsequent CvP updates?
d. If possible, attach your design so that we can review the reset and clock connections and try to replicate your failure.
e. Describe the steps necessary to run your design.

21. For subsequent CvP update, you must compile both revisions for any changes to any of the following logic:
   a. The periphery logic
   b. The I/O ports or core wrapper
   c. The Quartus Prime software version

Related Information
- Stratix V GX High Development Board
- Stratix V Device Handbook

**Known Issues and Solutions**

1. CvP designs with Gen1 x2 configurations fail to link up after loading the periphery image. One way to work around this issue is to use Gen1 x4 configuration and let the link downtrain to Gen1 x2.
2. You cannot use the Transceiver Reconfiguration Controller IP Core in CvP update mode. Refer to Workaround for a Known Issue with Transceiver Reconfiguration Controller IP Core on page 5-19 for more a workaround.

**Using MSI-X in CvP Initialization Mode**

If you are using message signaled interrupts (MSI-X) in CvP Initialization mode, you must ensure that the MSI-X table loads after the Quartus Prime software loads the core image.

To ensure that MSI-X tables are set up correctly, follow the steps below:

1. Set up the MSI-X in the Quartus Prime software and enable MSI-X.
2. Load the periphery image.
3. Load the core image.
4. If a driver was set up previously, uninstall and reinstall the driver using your application software after the quartus_cvp core image has been loaded. Or, disable and re-enable the driver after the quartus_cvp core image has been loaded.
5. The MSI-X table is set up. You can now observe interrupts on the link.
CvP Driver Support

You can develop your own custom CvP driver for Linux using the sample Linux driver source code provided by Intel\(^{(2)}\). The sample driver is written in C and can be downloaded from the Configuration via Protocol webpage.

**Note:** The Linux driver provided by Intel is not a production driver. You must adapt this driver to your design’s strategy.

You can also develop your own CvP driver using the Jungo WinDriver tool. You need to purchase a WinDriver license for this purpose.

**Note:** Intel does not provide Jungo WinDriver as a part of Intel Quartus Prime install package starting with software version 19.1. You can develop your own custom CvP driver for Linux using the sample Linux driver source code provided by Intel.

**Related Information**

- [Configuration via Protocol](#)
  Provides more information about sample Linux device source code.
- [Download the Open Source Linux CvP Driver](#)

CvP Driver Flow

The following figure shows the flow of the provided CvP driver. The flow assumes that the FPGA is powered up and the control block has already configured the FPGA with the periphery image, which is indicated by the CVP\_EN bit in the CvP status register.

As this figure indicates, the third step of the Start Teardown Flow requires 244 dummy configuration writes to the CVP\_DATA register or 244 memory writes to an address defined by a memory space BAR for this device. Memory writes are preferred because they are higher throughput than configuration writes. The dummy writes cause a 2 ms delay, allowing the control block to complete required operations.

\(^{(2)}\) You must set the value of the CVP\_NUMCLKS register to specific constant value as mentioned in CvP Mode Control Register section to leverage the code of the sample Linux driver.
Set the CVP_NUMCLKS byte to 0x01 in the CvP mode control register and issues 244 dummy writes to the hard IP. Dummy writes can be config writes to the CVP_DATA register or memory writes to any address defined by a BAR for this device.

Write '1' to the CVP_CONFIG bit in the CvP programming control register.

Poll the CVP_CONFIG_READY bit from the CvP status register.

Read the CVP_CONFIG_ERROR bit in the CvP status register.

CVP_CONFIG_ERROR=1?

no

CVP_CONFIG_ERROR_LATCHED = 0?

no

(1) Wait for periphery load completion.

Start Transfer

Write 32-bit fabric configuration data from *.core.rbf file to the CVP data register.

CVP_CONFIG_ERROR=1?

no

Log configuration error

End of fabric configuration file (core.rbf)?

yes

End

Start Teardown

Write '0' to the CVP_CONFIG bit in the CvP programming control register.

Set the CVP_NUMCLKS byte to 0x01 in the CvP mode control register and issue 244 dummy writes to the hard IP. Dummy writes can be config writes to the CVP_DATA register or memory writes to any address defined by a BAR for this device.

Write '0' to the CVP_CONFIG bit in the CvP programming control register.

Poll the CVP_CONFIG READY bit from the CvP status register.

CVP_CONFIG READY=0?

no

CVP_CONFIG_ERROR_LATCHED = 0?

no

CVP_CONFIG_ERROR_LATCHED = 0?

no

Write '0' to CVP_MODE and HIP_CLK_SEL.

PLD_CLK_IN_USE = 1 and USER_MODE = 1?

no

yes

End
VSEC Registers for CvP

The Vendor Specific Extended Capability (VSEC) registers occupy byte offset 0x200 to 0x240 in the PCIe Configuration Space. The PCIe host uses these registers to communicate with the FPGA control block. The following table shows the VSEC register map. Subsequent tables provide the fields and descriptions of each register.

Table 6-1: VSEC Registers for CvP

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Register Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x200</td>
<td>Altera-defined Vendor Specific Capability Header</td>
</tr>
<tr>
<td>0x204</td>
<td>Altera-defined Vendor Specific Header</td>
</tr>
<tr>
<td>0x208</td>
<td>Altera Marker</td>
</tr>
<tr>
<td>0x20C:0x218</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x21C</td>
<td>CvP Status</td>
</tr>
<tr>
<td>0x220</td>
<td>CvP Mode Control</td>
</tr>
<tr>
<td>0x224</td>
<td>CvP Data 2</td>
</tr>
<tr>
<td>0x228</td>
<td>CvP Data</td>
</tr>
<tr>
<td>0x22C</td>
<td>CvP Programming Control</td>
</tr>
<tr>
<td>0x230</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x234</td>
<td>Uncorrectable Internal Error Status Register</td>
</tr>
<tr>
<td>0x238</td>
<td>Uncorrectable Internal Error Mask Register</td>
</tr>
<tr>
<td>0x23C</td>
<td>Correctable Internal Error Status Register</td>
</tr>
<tr>
<td>0x240</td>
<td>Correctable Internal Error Mask Register</td>
</tr>
</tbody>
</table>

Altera-defined Vendor Specific Capability Header Register

Table 6-2: Altera-defined Vendor Specific Capability Header Register (Byte Offset: 0x200)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15:0]</td>
<td>PCI Express Extended Capability ID</td>
<td>0x000B</td>
<td>RO</td>
<td>PCIe specification defined value for VSEC Capability ID.</td>
</tr>
<tr>
<td>[19:16]</td>
<td>Version</td>
<td>0x1</td>
<td>RO</td>
<td>PCIe specification defined value for VSEC version.</td>
</tr>
<tr>
<td>[31:20]</td>
<td>Next Capability Offset</td>
<td>Variable</td>
<td>RO</td>
<td>Starting address of the next Capability Structure implemented, if any.</td>
</tr>
</tbody>
</table>
Altera-defined Vendor Specific Header Register

Table 6-3: Altera-defined Vendor Specific Header Register (Byte Offset: 0x204)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15:0]</td>
<td>VSEC ID</td>
<td>0x1172</td>
<td>RO</td>
<td>A user configurable VSEC ID.</td>
</tr>
<tr>
<td>[19:16]</td>
<td>VSEC Revision</td>
<td>0</td>
<td>RO</td>
<td>A user configurable VSEC revision.</td>
</tr>
<tr>
<td>[31:20]</td>
<td>VSEC Length</td>
<td>0x044</td>
<td>RO</td>
<td>Total length of this structure in bytes.</td>
</tr>
</tbody>
</table>

Altera Marker Register

Table 6-4: Altera Marker Register (Byte Offset: 0x208)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>Altera Marker</td>
<td>Device Value</td>
<td>RO</td>
<td>An additional marker. If you use the standard Altera Programmer software to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>configure the device with CvP, this marker provides a value that the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>programming software reads to ensure that it is operating with the correct</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VSEC.</td>
</tr>
</tbody>
</table>

CvP Status Register

Table 6-5: CvP Status Register (Byte Offset: 0x21C)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:26]</td>
<td>—</td>
<td>0x00</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[25]</td>
<td>PLD_CORE_READY</td>
<td>Variable</td>
<td>RO</td>
<td>From FPGA fabric. This status bit is provided for debug.</td>
</tr>
<tr>
<td>[24]</td>
<td>PLD_CLK_IN_USE</td>
<td>Variable</td>
<td>RO</td>
<td>From clock switch module to fabric. This status bit is provided for debug.</td>
</tr>
<tr>
<td>[23]</td>
<td>CVP_CONFIG_DONE</td>
<td>Variable</td>
<td>RO</td>
<td>Indicates that the FPGA control block has completed the device configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>via CvP and there were no errors.</td>
</tr>
<tr>
<td>[22]</td>
<td>—</td>
<td>Variable</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[21]</td>
<td>USERMODE</td>
<td>Variable</td>
<td>RO</td>
<td>Indicates if the configurable FPGA fabric is in user mode.</td>
</tr>
<tr>
<td>Bits</td>
<td>Name</td>
<td>Reset Value</td>
<td>Access</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------</td>
<td>-------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[20]</td>
<td>CVP_EN</td>
<td>Variable</td>
<td>RO</td>
<td>Indicates if the FPGA control block has enabled CvP mode.</td>
</tr>
<tr>
<td>[19]</td>
<td>CVP_CONFIG_ERROR</td>
<td>Variable</td>
<td>RO</td>
<td>Reflects the value of this signal from the FPGA control block, checked by software to determine if there was an error during configuration.</td>
</tr>
<tr>
<td>[18]</td>
<td>CVP_CONFIG_READY</td>
<td>Variable</td>
<td>RO</td>
<td>Reflects the value of this signal from the FPGA control block, checked by software during programming algorithm.</td>
</tr>
<tr>
<td>[17:0]</td>
<td>—</td>
<td>Variable</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

### CvP Mode Control Register

**Table 6-6: CvP Mode Control Register (Byte Offset: 0x220)**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:16]</td>
<td>—</td>
<td>0x0000</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
| [15:8]  | CVP_NUMCLKS    | 0x00        | RW     | This is the number of clocks to send for every CvP data write. This is also known as CDRATIO (clock to data ratio). Set this field to one of the values below depending on your configuration image:  
  - 0x01 for uncompressed and unencrypted images  
  - 0x04 for uncompressed and encrypted images  
  - 0x08 for all compressed images |
<p>| [7:3]   | —              | 0x0         | RO     | Reserved.                                                                  |
| [2]     | CVP_FULLCONFIG  | 1'b0        | RW     | A value of 1 indicates a request to the control block to reconfigure the entire FPGA including the Hard IP for PCI Express and bring the PCIe link down. |</p>
<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>HIP_CLK_SEL</td>
<td>1'b0</td>
<td>RW</td>
<td>Selects between PMA and fabric clock when USER_MODE = 1 and PLD_CORE_READY = 1. The following encodings are defined:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 1: Selects internal clock from PMA which is required for CVP_MODE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0: Selects the clock from soft logic fabric. This setting should only be used when the fabric is configured in USER_MODE with a configuration file that connects the correct clock.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To ensure that there is no clock switching during CvP, you should only change this value when the Hard IP for PCI Express has been idle for 10 µs and wait 10 µs after changing this value before resuming activity.</td>
</tr>
<tr>
<td>[0]</td>
<td>CVP_MODE</td>
<td>1'b0</td>
<td>RW</td>
<td>Controls whether the Hard IP for PCI Express is in CVP_MODE or normal mode. The following encodings are defined:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 1: CVP_MODE is active. Signals to the FPGA control block active and all TLPs are routed to the Configuration Space. This CVP_MODE cannot be enabled if CVP_EN = 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• 0: The IP core is in normal mode and TLPs are routed to the FPGA fabric.</td>
</tr>
</tbody>
</table>
CvP Data Registers

Table 6-7: CvP Data Register (Byte Offsets: 0x224 - 0x228)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:0]</td>
<td>CVP_DATA2</td>
<td>0x00000000</td>
<td>RW</td>
<td>Contains the upper 32 bits of a 64-bit configuration data. Software must ensure that all Bytes in both dwords are enabled. Use of 64-bit configuration data is optional.</td>
</tr>
<tr>
<td>[31:0]</td>
<td>CVP_DATA</td>
<td>0x00000000</td>
<td>RW</td>
<td>Write the configuration data to this register. The data is transferred to the FPGA control block to configure the device. Every write to this register sets the data output to the FPGA control block and generates (&lt;n&gt;) clock cycles to the FPGA control block as specified by the CVP_NUM_CLKS field in the CvP Mode Control register. Software must ensure that all bytes in the memory write dword are enabled. You can access this register using configuration writes. Alternatively, when in CvP mode, this register can also be written by a memory write to any address defined by a memory space BAR for this device. Using memory writes are higher throughput than configuration writes.</td>
</tr>
</tbody>
</table>

CvP Programming Control Register

Table 6-8: CvP Programming Control Register (Byte Offset: 0x22C)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:2]</td>
<td>—</td>
<td>0x0000</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[1]</td>
<td>START_XFER</td>
<td>1'b0</td>
<td>RW</td>
<td>Sets the CvP output to the FPGA control block indicating the start of a transfer.</td>
</tr>
<tr>
<td>[0]</td>
<td>CVP_CONFIG</td>
<td>1'b0</td>
<td>RW</td>
<td>When set to 1, the FPGA control block begins a transfer via CvP.</td>
</tr>
</tbody>
</table>
Uncorrectable Internal Error Status Register

This register reports the status of the internally checked errors that are uncorrectable. When specific errors are enabled by the Uncorrectable Internal Error Mask register, they are handled as Uncorrectable Internal Errors as defined in the PCI Express Base Specification 3.0. This register is for debug only. Use this register to observe behavior, not to drive custom logic.

Table 6-9: Uncorrectable Internal Error Status Register (Byte Offset: 0x234)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:12]</td>
<td>0x00</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[11]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates an RX buffer overflow condition in a posted request or Completion segment.</td>
</tr>
<tr>
<td>[10]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a parity error was detected on the R2CSEB interface.</td>
</tr>
<tr>
<td>[9]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a parity error was detected on the Configuration Space to TX bus interface.</td>
</tr>
<tr>
<td>[8]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a parity error was detected on the TX to Configuration Space bus interface.</td>
</tr>
<tr>
<td>[7]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a parity error was detected in a TX TLP and the TLP is not sent.</td>
</tr>
<tr>
<td>[6]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates that the Application Layer has detected an uncorrectable internal error.</td>
</tr>
<tr>
<td>[5]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a configuration error has been detected in CvP mode which is reported as uncorrectable. This CVP_CONFIG_ERROR_LATCHED bit is set whenever a CVP_CONFIG_ERROR is asserted while in CVP_MODE.</td>
</tr>
<tr>
<td>[4]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a parity error was detected by the TX Data Link Layer.</td>
</tr>
<tr>
<td>[3]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a parity error has been detected on the RX to Configuration Space bus interface.</td>
</tr>
<tr>
<td>[2]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a parity error was detected at input to the RX Buffer.</td>
</tr>
<tr>
<td>[1]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a retry buffer uncorrectable ECC error.</td>
</tr>
<tr>
<td>[0]</td>
<td>1'b0</td>
<td>RW1CS</td>
<td>A value of 1 indicates a RX buffer uncorrectable ECC error.</td>
</tr>
</tbody>
</table>
Uncorrectable Internal Error Mask Register

This register controls which errors are forwarded as internal uncorrectable errors. With the exception of the configuration errors detected in CvP mode, all of the errors are severe and may place the device or PCIe link in an inconsistent state. The configuration error detected in CvP mode may be correctable depending on the design of the programming software.

Table 6-10: Uncorrectable Internal Error Mask Register (Byte Offset: 0x238)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:12]</td>
<td>0x00</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[10]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for parity error on the R2CSEB interface.</td>
</tr>
<tr>
<td>[9]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for parity error on the Configuration Space to TX bus interface.</td>
</tr>
<tr>
<td>[8]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for parity error on the TX to Configuration Space bus interface.</td>
</tr>
<tr>
<td>[7]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for parity error in the transaction layer packet.</td>
</tr>
<tr>
<td>[6]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for parity error in the application layer.</td>
</tr>
<tr>
<td>[5]</td>
<td>1'b0</td>
<td>RWS</td>
<td>Mask for configuration error in CvP mode.</td>
</tr>
<tr>
<td>[4]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for data parity errors detected during TX Data Link LCRC generation.</td>
</tr>
<tr>
<td>[3]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for data parity errors detected on the RX to Configuration Space Bus interface.</td>
</tr>
<tr>
<td>[2]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for data parity error detected at the input to the RX Buffer.</td>
</tr>
<tr>
<td>[1]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for the retry buffer uncorrectable ECC error.</td>
</tr>
<tr>
<td>[0]</td>
<td>1'b1</td>
<td>RWS</td>
<td>Mask for the RX buffer uncorrectable ECC error.</td>
</tr>
</tbody>
</table>

Correctable Internal Error Status Register

This register reports the status of the internally checked errors that are correctable. When these specific errors are enabled by the Correctable Internal Error Mask register, they are forwarded as Correctable Internal Errors as defined in the PCI Express Base Specification 3.0. This register is for debug only. Use this register to observe behavior, not to drive custom logic.

Table 6-11: Correctable Internal Error Status Register (Byte Offset: 0x23C)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:7]</td>
<td>0x00</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
Correctable Internal Error Mask Register

This register controls which errors are forwarded as Internal Correctable Errors. This register is for debug only.

Table 6-12: Correctable Internal Error Mask Register (Byte Offset: 0x240)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Reset Value</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:7]</td>
<td>0x000</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[6]</td>
<td>1'b0</td>
<td>RWS</td>
<td>Mask for corrected internal error reported by the Application Layer.</td>
</tr>
<tr>
<td>[5]</td>
<td>1'b0</td>
<td>RWS</td>
<td>Mask for configuration error detected in CvP mode.</td>
</tr>
<tr>
<td>[4:2]</td>
<td>0x0</td>
<td>RO</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[1]</td>
<td>1'b0</td>
<td>RWS</td>
<td>Mask for retry buffer correctable ECC error.</td>
</tr>
<tr>
<td>[0]</td>
<td>1'b0</td>
<td>RWS</td>
<td>Mask for RX buffer correctable ECC error.</td>
</tr>
</tbody>
</table>

2020.09.04

• Added information about custom CvP driver for Linux in section Bringing Up the Hardware.
• Added a note about the availability of Jungo WinDriver in following sections:
  • Bringing Up the Hardware
  • CvP Driver Support
• Added new section Installing Open Source CvP Driver in Linux System.
• Added steps to program the core image using the open source Linux diver in section Programming CvP Images.

Date | Version | Changes
--- | --- | ---
October 2016 | 2.0 | • Changed the document title to Configuration via Protocol (CvP) Implementation in V-series FPGA Devices User Guide.
• Added references to the Configuration Timing Waveforms.
• Corrected the PCIe Timing Sequence diagrams for both CvP initialization and update mode.
• Added multiple snapshots of the Intel Quartus Prime Standard Edition illustrating the following design steps:
  • Downloading and Generating the High Performance Reference Design
  • Setting up CvP Parameters for CvP Initialization Mode
  • Setting up CvP Parameters for CvP Update Mode
• Simplified the Figure: Switch 4 (SW4) Configuration for MSEL[4:0] = 5'b10010 on the back view of Stratix V Device Kit.
• Added the timeout value for CvP_CONFIG READY in the Figure: CvP Driver Flow.
• Fixed assorted typos and formatting issues.
### Document Revision History for the Configuration via Protocol (CvP) Implementation in V-series FPGA Devices

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| May 2016       | 1.9     | • Corrected the nPERST pin locations for Cyclone V devices in the Table: CvP pin descriptions and connection guidelines.  
• Added the pin descriptions for INIT_DONE and CONFDONE.  
• Corrected the values of CvP Mode Control Register in the Table: CvP Mode Control Register (Byte Offset: 0x220).  
• Clarified the Bit[5] as CVP_CONFIG_ERROR_LATCHED bit in the Table: Uncorrectable Internal Error Status Register (Byte Offset: 0x234).  
• Removed references to Arria 10 devices. For CvP implementation in Arria 10 devices, refer to the Arria 10 CvP Initialization and Partial Reconfiguration over Protocol User Guide.  
• Fixed assorted typos and formatting issues. |
| November 2015  | 1.8     | • Updated the Table: CvP Support for Device Families.  
• Clarified the pin descriptions in the Table: CvP pin descriptions and connection guidelines.  
• Updated the example design reference to PCIe AVST and On-Chip Memory Interface design files.  
• Changed instances of Quartus II to Quartus Prime.  
• Removed the Daisy chain topology section. |
| December 2014  | 1.7     | • Updated the CvP Modes section to indicate difference between CvP Initialization and CvP Update modes.  
• Added a new section called Autonomous Mode.  
• Corrected a typographical error in Figure: "Switch 4 (SW4) Configuration for MSEL[4:0] = 5'b10010 on the back view of Stratix V Device Kit" to indicate that down position signifies logic one.  
• Corrected modes supported for Arria V and Cyclone V Gen2 variants. Gen2 supports CvP initialization mode. Gen2 does not support CvP update mode.  
• Updated the Clock Connections for CvP Designs Including the Transceiver Reconfiguration Controller section dynamic reconfiguration usage guidelines.  
• Added a new section called Using MSI-X in CvP Initialization Mode under Known Issues and Solutions. |
| November 2013  | 1.6     | • Added optional CVP_DATA2 register for use when configuration data is 64 bits wide.  
• Added design constraint for input reference clock when more than one Transceiver Reconfiguration Controller connects to the transceivers on one side of the device.  
• Corrected errors in software driver flow diagram.  
• Added clarification that you must select either CvP Initialization mode or CvP Update mode. The two modes cannot be combined.  
• Updated CvP Pins section.  
• Updated CvP Example Designs chapter.  
• Updated CvP Features section in Design Considerations chapter. |
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 2013</td>
<td>1.5</td>
<td>• Added support for 64-bit data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Changed supported clock frequencies for CvP updates using encrypted data and a non-volatile key. Only 12.5 MHz is supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added reasons for using a compressed bitstream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clarified process to create separate design hierarchy for periphery and core logic.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added table showing supported features in CvP Initialization and Update Mode and CvP Update Mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Clarified use of dummy writes for CvP driver in the teardown flow.</td>
</tr>
<tr>
<td>May 2013</td>
<td>1.4</td>
<td>• Added CvP Example Designs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Partitioning a Design for the CvP Design Flow section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated the Power Supplies Ramp-Up Time and POR, PCIe Timing Sequence in CvP Initialization Mode, and PCIe Timing Sequence in CvP Update Mode figures.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Moved all links in all topics to the Related Information section for easy reference.</td>
</tr>
<tr>
<td>December 2012</td>
<td>1.3</td>
<td>• Removed Gen 3 support from the CvP Update Mode in Table 2-1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Table 5-3 to include the supported clock source for encrypted configuration data in CvP mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Table 6-2 to include the <code>quartus_cvp</code> command for compression and encryption modes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Table 6-3 to include the <code>CVP_NUMCLKS</code> settings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Bit 1 and Bit 0 to reserved bits in Table 6-8.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated the <code>CVP_NUMCLKS</code> settings in Figure 6-1.</td>
</tr>
<tr>
<td>July 2012</td>
<td>1.2</td>
<td>• Updated &quot;Data Compression&quot; and Data Encryption&quot; sections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Moved Table 5-6. Uncompressed <code>.rbf</code> sizes to Stratix V, Arria V, and Cyclone V Configuration, Design Security, and Remote System Upgrades chapters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added Jungo WinDriver and the Linux-based Plain Text C CvP driver in &quot;Software Support&quot; chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated <code>CVP_NUMCLKS</code> byte settings in Table 6-8 and Figure 6-1.</td>
</tr>
<tr>
<td>January 2012</td>
<td>1.1</td>
<td>• Added Arria V and Cyclone V devices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed references to the CvP Off mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Chapter 6, Software Support with PCIe driver and CvP programming information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added &quot;Generating Periphery Image and Fabric Image&quot; and &quot;VSEC for CvP&quot; sections.</td>
</tr>
<tr>
<td>May 2011</td>
<td>1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>