XAPP883 (v1.0) November 19, 2010

Fast Configuration of PCI Express Technology through Partial Reconfiguration

Author: Simon Tam and Martin Kellermann

Summary

The PCI Express® specification requires ports to be ready for link training at a minimum of 100 ms after the power supply is stable. (Refer to the *Virtex®-6 FPGA Integrated Block for PCI Express User Guide* [Ref 1] for more information.) This becomes a difficult task due to the ever-increasing configuration memory size of each new generation of FPGA, such as the Xilinx® Virtex-6 family. One innovative approach to addressing this challenge is leveraging the advances made in the area of FPGA partial reconfiguration to split the overall configuration of a PCIe® specification based system in a large FPGA into two sequential steps:

1. Initial PCIe system link configuration
2. Subsequent user application reconfiguration

It is feasible to configure only the FPGA PCIe system block and associated logic during the first stage within the 100 ms window before the fundamental reset is released. Using the Virtex-6 FPGA's partial reconfiguration capability, the host can then reconfigure the FPGA to apply the user application via the now-active PCIe system link.

This methodology not only provides a solution for faster PCIe system configuration, it enhances user application security because the bitstream is only accessible by the host and can be better encrypted. This approach also helps lower the system cost by reducing external configuration component costs and board space.

This application note describes the methodology for building a Fast PCIe Configuration (FPC) module using this two-step configuration approach. A reference design is available to help designers quick-launch a PlanAhead™ software partial reconfiguration project. (Refer to the *Partial Reconfiguration User Guide* [Ref 2] for more information about the partial reconfiguration design flow.) The reference design implements an eight-lane PCIe technology generation-1 link and targets the Virtex-6 FPGA ML605 Evaluation Board. The reference design serves as a guide for designers to develop their own solutions.

Overview

An FPC module consists of two partitions: the static partition and the reconfigurable partition. The static partition is configured during the initial first-stage configuration. It provides the channel and framework necessary for the host to reconfigure through the PCIe system link and internal configuration access port (ICAP) of the Virtex-6 FPGA. (Refer to the *Virtex-6 FPGA Configuration User Guide* [Ref 3] for more information about ICAP.) The static partition consists of an Integrated Block for PCI Express, a switcher, and a partial reconfiguration (PR) loader, as shown in Figure 1. The user application resides in the reconfigurable partition and is applied to the FPGA during second-stage configuration. The PlanAhead tool creates one unique bitstream for each partition.

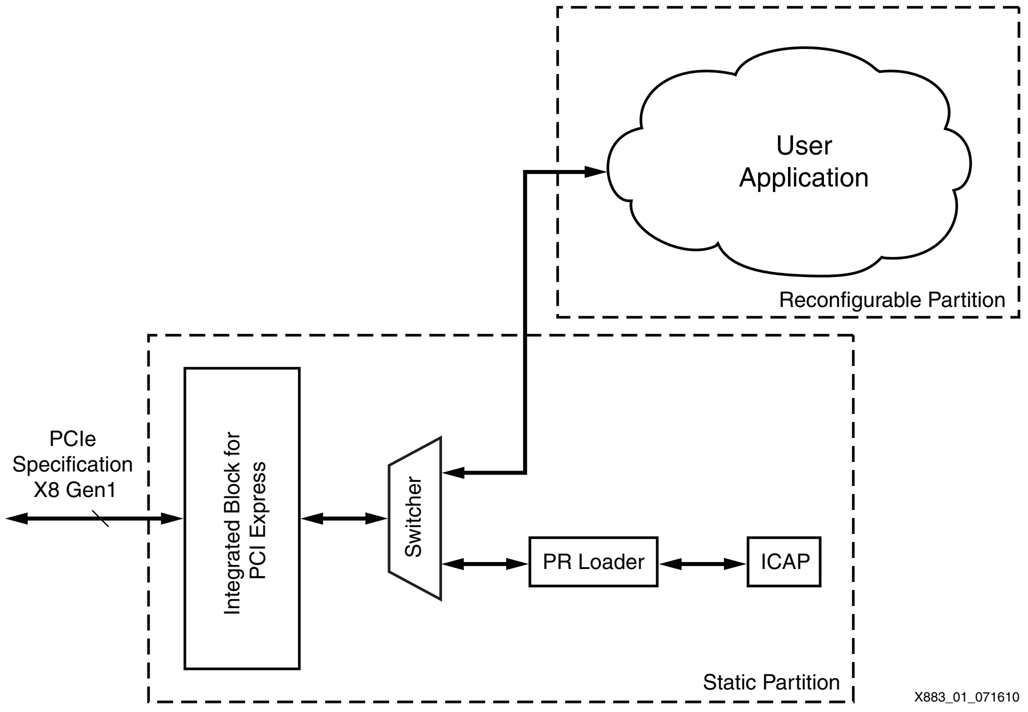


Figure 1: **FPC Module Architecture**

Upon power-up, the FPGA's static partition is configured from an external PROM (Figure 2).

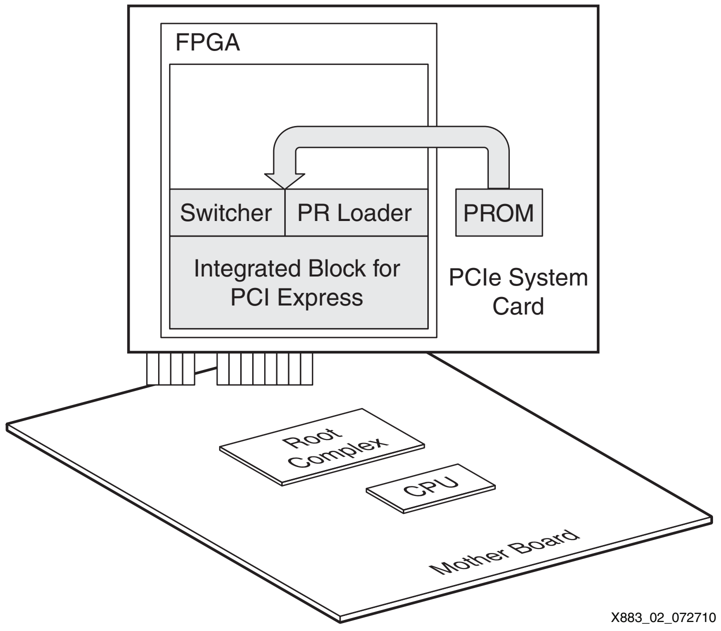


Figure 2: **First Stage Configuration from a PROM**

Although the Virtex-6 FPGA must be configured with a full bitstream during this initial stage, the total logic resources of the static partition are small and compact. The configuration bitstream size can be greatly reduced using the multiple configuration frame write feature to enable bitstream compression. This compression scheme takes advantage of the fact that most of the FPGA is empty space consisting of "blank" configuration frames, and these blank frames are identical. These identical frames can be configured simultaneously using multi-frame writes, greatly speeding up the configuration process. The compressed static partition bitfile takes much less time to configure than a full configuration bitfile. This allows the Virtex-6 FPGA to complete its configuration and be ready for PCIe system link initialization within the required time after power-up.

At this point, the static partition is fully operational, but the user application has not yet been configured in the Virtex-6 FPGA. The unused FPGA resources, including the reconfigurable partition, is filled with blanks during first-stage configuration.

After the link between the FPGA and the host system is established, the host system can proceed with partial reconfiguration. The host transmits the partial bitstream to the Virtex-6 FPGA in the form of configuration instruction sequences. Initially, the switcher connects the PCIe system transaction interface of the FPGA with the PR loader. The PR loader receives the reformatted bitstream and feeds it to the ICAP to complete the partial reconfiguration process (Figure 3). The user application reset is asserted before, during, and slightly after the partial reconfiguration process to prevent a false start in the application.

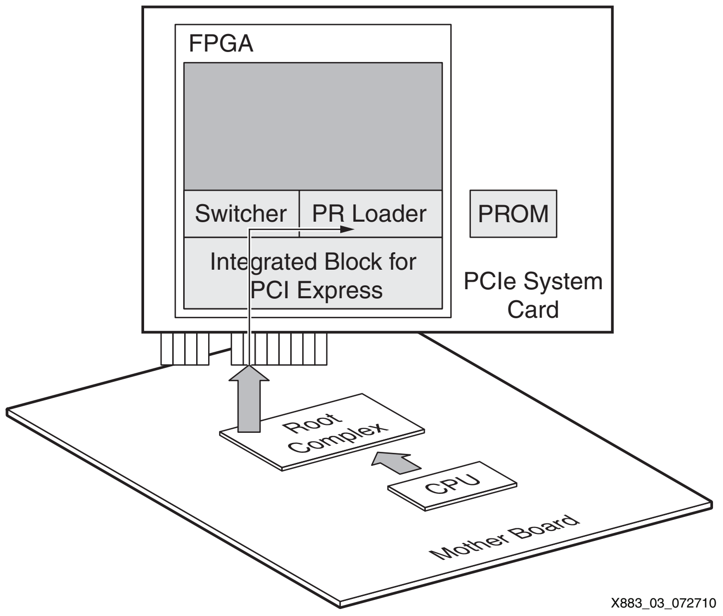


Figure 3: **Second Stage Partial Reconfiguration**

After successful partial reconfiguration, the PR loader startup module releases the user application reset and waits for the user application ready signature. The ready signature is a unique 32-bit word that indicates the user application is functioning and has completed its initialization. After the startup module verifies the ready signature, it instructs the switcher to redirect the PCIe system transaction layer interface signals to the user application, allowing access to the PCIe system bus. At this point, the user application is operational and can begin communicating with the host through the PCIe system interface (Figure 4).

Note: In the provided reference design, the PR loader is no longer accessible after the user application takes control of the PCIe system link. However, this functionality can be added by the user if desired.

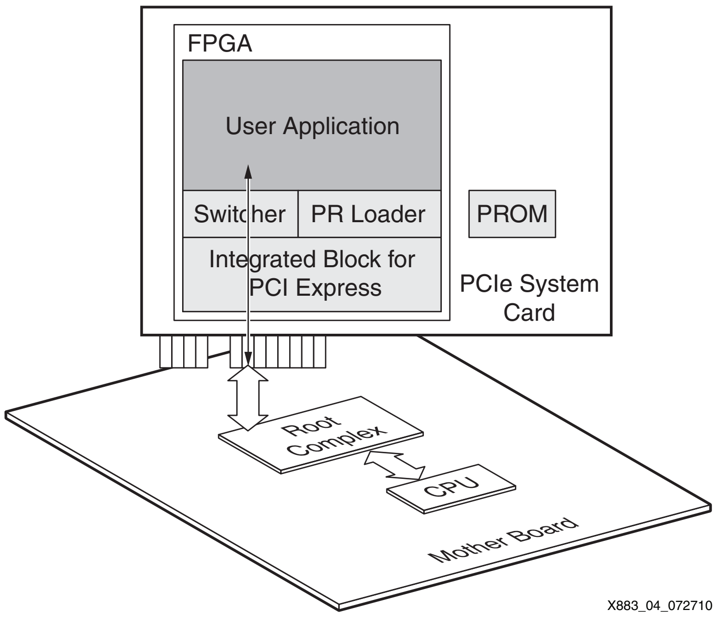


Figure 4: **User Application in Operation**

The static partition remains in the FPGA after the reconfigurable partition becomes operational. Because the static partition size is very small (about 200 LUTs), its impact on FPGA resource usage is minimal.

Static Partition Bitstream Compression and Timing

The total first-stage configuration time is made up of the power-on reset time (T_{POR}) plus the static partition configuration time. Refer to the *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics* [Ref 4] for the duration of power-on reset.

The bitstream of the static partition is compressed to reduce the bitfile size and its configuration time. Bitstream compression uses the Virtex-6 FPGA multiple configuration frame write feature to write identical configuration frames all at once instead of writing each frame individually. Therefore, the bitstream size compression ratio is not deterministic because the outcome depends on the final FPGA resource usage, placement, and other factors. In general, the compressed static partition bitfile size is between 2 MB and 5 MB.

For reference, the total size of an uncompressed XC6VLX240T bitfile is 9.23 MB. The compressed static partition bitfile size of this reference design is approximately 2 MB. The static partition configuration time on an ML605 demonstration platform is about 20 ms using a Xilinx Platform Flash XL running at a clock rate of 47 MHz.

Equation 1 is the formula for calculating the static partition configuration time.

$$T_{CONFIG} = \frac{BF_{SIZE}}{BW_{PROM}} \qquad \textit{Equation 1}$$

where

T_{CONFIG} = Configuration time
BF_{SIZE} = Static partition bitfile size in bytes
BW_{PROM} = PROM bandwidth in MB/s

In general, minimizing static partition resources utilization and confining the partition in a smaller FPGA array area is desirable because it optimizes bitstream compression.

Static Partition Detailed Descriptions

Figure 5 shows the detailed block diagram of the FPC module.

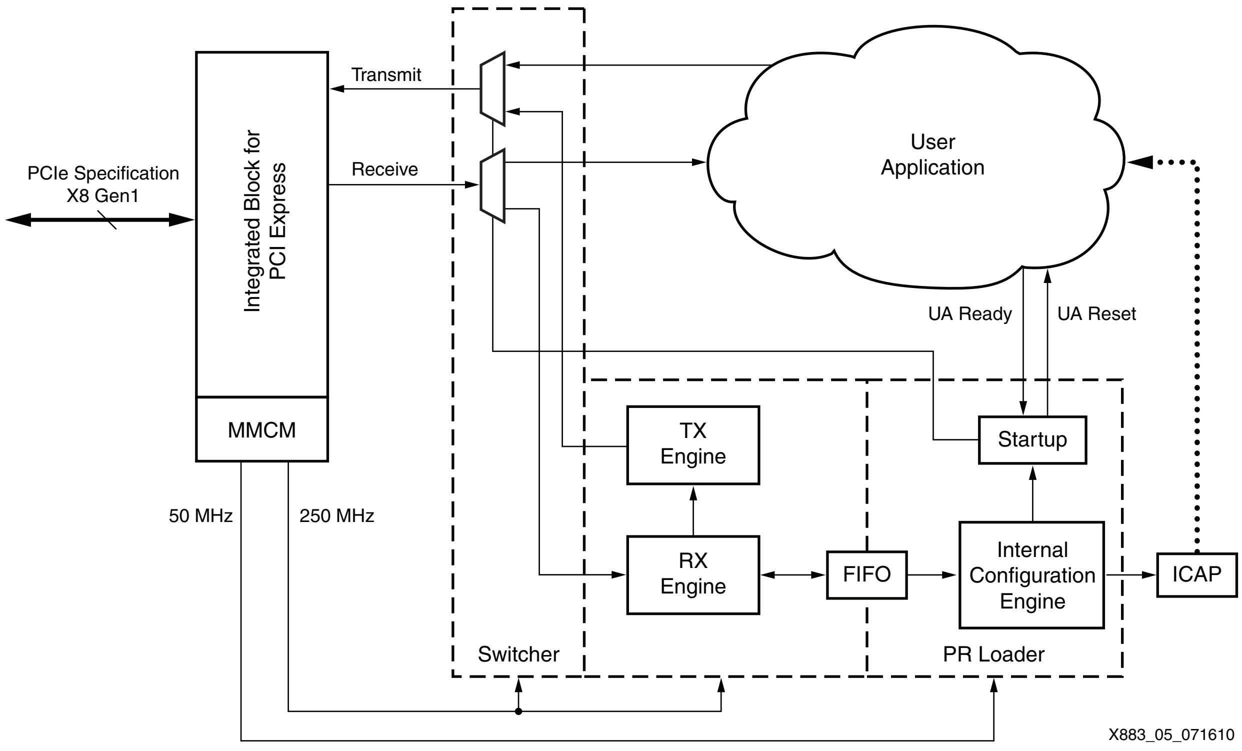


Figure 5: **Detailed Block Diagram**

Clocking

The FPC module is divided into two clock regions: the PCIe system interface and the ICAP clock regions. The PCIe system interface clock region operates at 250 MHz, while the ICAP clock region operates at 50 MHz. A built-in FIFO within the PR loader acts as a cross clock domain buffer. All clocks, including those for the integrated block, are generated from the same mixed-mode clock manager (MMCM) within the Integrated Block wrapper. It is recommended that one MMCM be used for all clocks. If multiple MMCMs are implemented, users must ensure clock phase alignment between different clock regions. A requirement of partial reconfiguration is that all global clocks (BUFG, etc.) and clock modifying logic (MMCM, etc.) be located in the static partition only.

Integrated Block for PCI Express

The design uses the Integrated Block v1.5 for PCI Express generated by the CORE Generator™ software. It uses the built-in integrated block for PCI Express of the Virtex-6 FPGA and eight GTX transceivers. The physical layer operates in eight lanes at Gen1 speed (2.5 Gb/s). The external reference clock frequency is 100 MHz.

The transaction layer interface datapath is 64 bits wide, running at 250 MHz. The reference design is not tested for other link width/speed combinations. The PCIe system configuration space must be configured for the eventual user application that resides in the reconfigurable module space. In the reference design, base address register 2 (BAR2) is used for the PR loader logic; therefore, BAR2 must exist in the user application regardless of whether or not it is used in the user application.

Note: BAR2 can be shared between the PR loader and the user application because the PR loader is disabled after reconfiguration has taken place.

Switcher

The switcher is essentially a multiplexer connecting to the Integrated Block for PCI Express, the PR loader, and the user application, as shown in Figure 6.

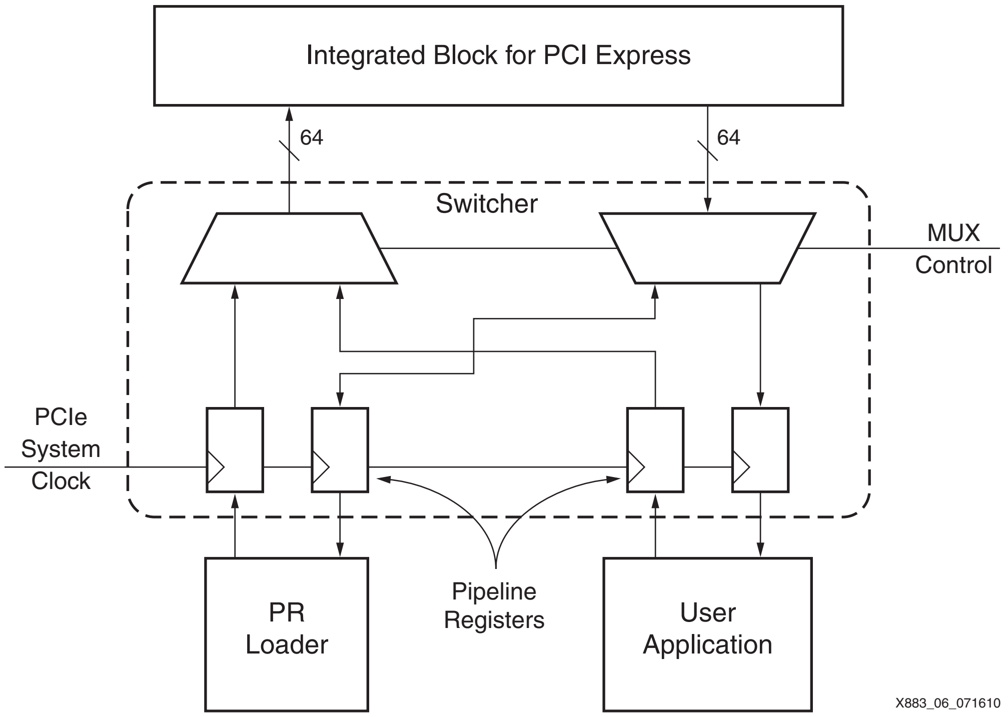


Figure 6: **Switcher Architecture**

On the PCIe device side, the switcher connects to the 64-bit transaction layer interface of the Integrated Block for PCI Express. The switcher connects to the PR loader and user application on the other side. During the first-stage configuration, the switcher enables the paths between the PCIe system interface and the PR loader while all the reconfigurable partition input ports are driven by static values. After partial reconfiguration, the switcher switches to the paths between the PCIe system interface and the user application, with all static partition input ports being driven by static values.

The reference design as provided does not support subsequent partial reconfigurations after the initial partial reconfiguration has been completed. The Virtex-6 FPGA is designed to support multiple partial reconfigurations, and users can alter the design to implement this capability if desired.

As shown in Figure 6, the switcher is pipelined to meet the 250 MHz timing requirement. This introduces differences in timing for user applications that do not expect this added latency. Special shim logic is introduced in the switcher to compensate for this effect. Existing user applications that connect directly to the CORE Generator software transaction interface do not need to be modified.

Figure 7 illustrates the effect of the shim on two back-to-back DMA transactions. When the shim receives an end of file (EOF), it deasserts the receive destination ready signal (`trn_rdst_rdy`) for two cycles. In other words, the shim inserts wait states to prevent another transaction from beginning immediately after the previous one. This action is necessary because the receive destination ready signal from the user application (`pr_rdst_rdy_n`) does not reach the integrated block two cycles later due to the added latency in the switcher.

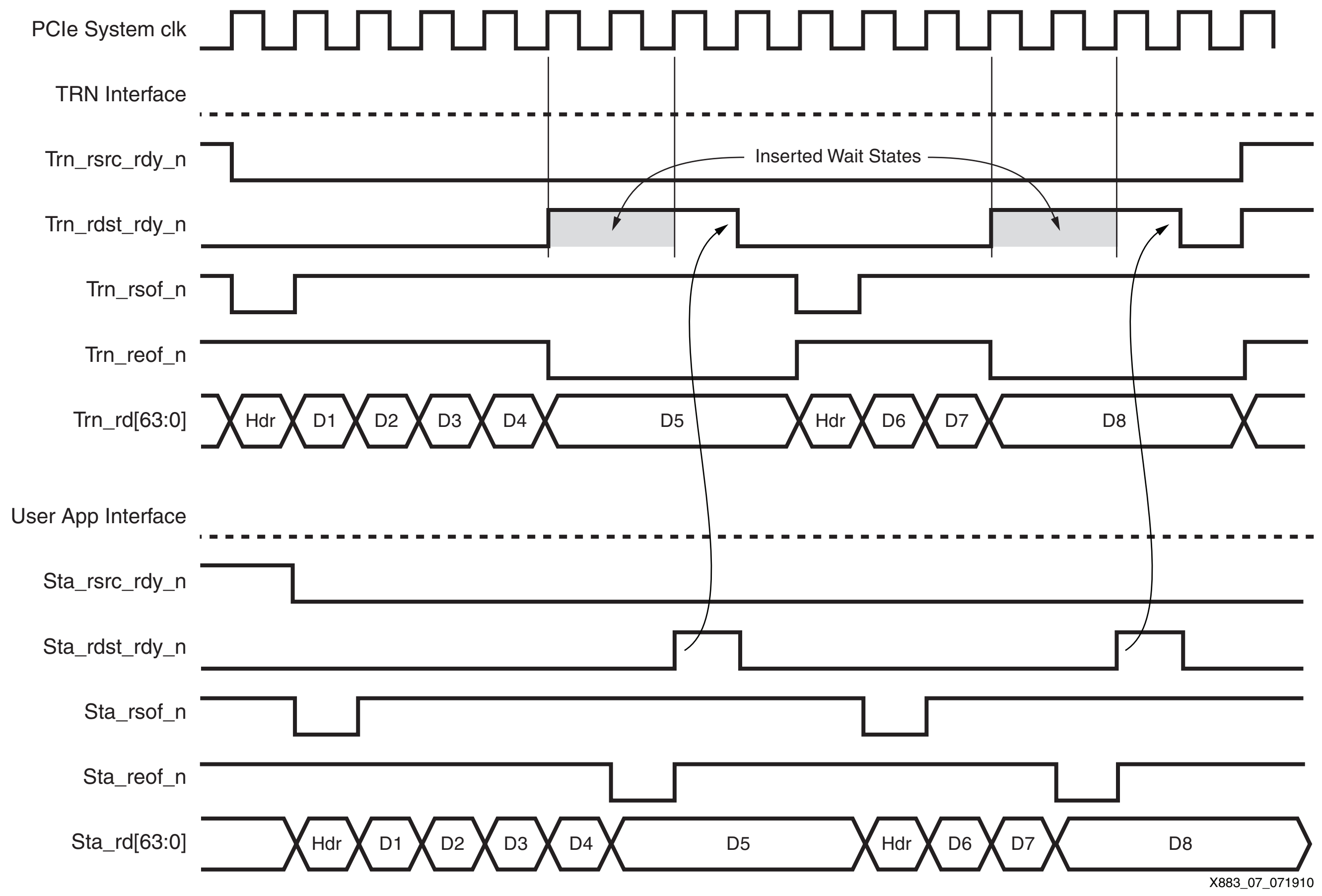


Figure 7: **Shim Logic Timing**

PR Loader

The PR loader consists of transmit, receive, and internal configuration engines. It is based on the programmed input/output (PIO) example design delivered with the integrated block and optimized for partial reconfiguration. The PR loader only supports 32-bit address memory write request (3DW MWr) transaction layer packets (TLPs) and PIO mode transactions. The Integrated Block BAR2 is designated for the PR loader. The supported PIO transaction payload size is one 32-bit double word only. In addition, the PR loader records only BAR hits and does not decode the exact target address because configuration data can only go to the ICAP.

The receive engine validates the incoming TLPs and extracts the configuration data from the payload. Configuration data is then sent to the write port of a 512 double words (DWs) deep asynchronous FIFO. The internal configuration engine retrieves the data from the read port of the FIFO. The FIFO acts as a cross clock domain buffer from the PCIe system user application clock rate of 250 MHz to the selected ICAP clock rate. Designers can adjust the ICAP clock rate up to the maximum rate allowed.

Partial reconfiguration is initiated when the PR loader software first writes the start-of-configuration (SOC) sequence to the FPGA. The SOC sequence consists of three DWs that trigger the PR loader state machines to prepare for pending partial reconfiguration. Table 1 shows the definitions of the SOC words.

Table 1: **SOC Sequence**

SOC_1	`x53545254h`
SOC_2	`x434F4E46h`
SOC_3	`x50434965h`

The sequence must be written in the order shown in Table 1. Any configuration word written to BAR2 prior to the SOC sequence is ignored. The SOC timing is shown in Figure 8.

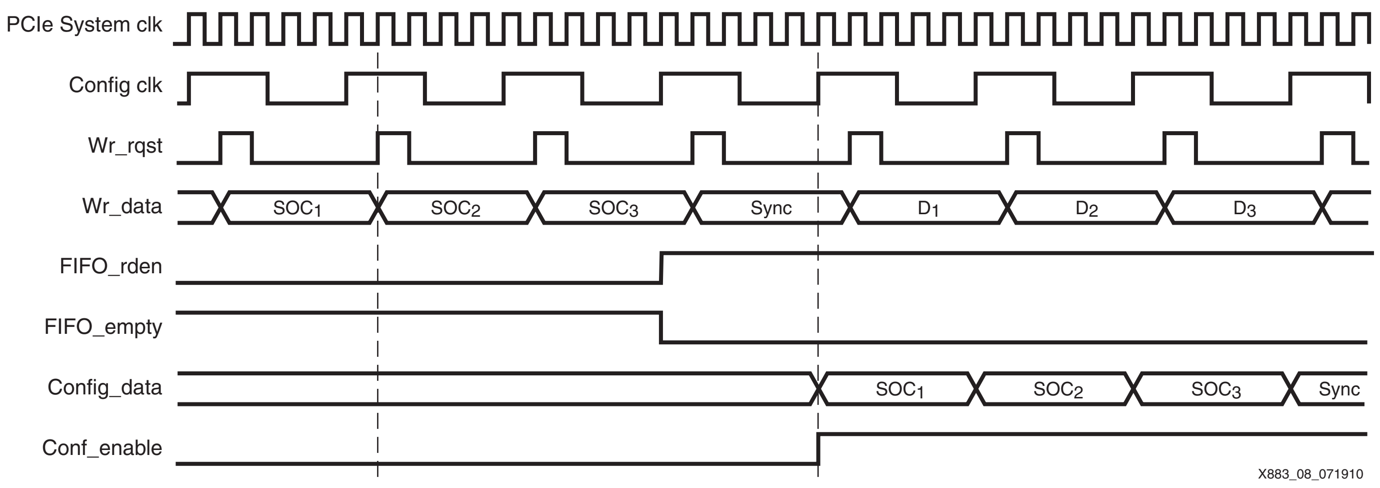


Figure 8: **SOC Timing**

After writing the SOC sequence, the PR loader application writes the content of the partial reconfiguration bitfile. The PR loader software reads the partial reconfiguration bitfile from system memory, extracts the configuration content, and writes a block of double words to the target address range defined in BAR2. The PR loader software repeats the process until the last configuration word is written. Software designers do not need to format the partial reconfiguration file content as long as the entire content is written in the original order.

The internal configuration engine reads the stored configuration words from the FIFO as soon as the empty flag is deasserted. It reformats and sends the 32-bit configuration DW to the ICAP while monitoring the FIFO and ICAP status.

After writing the last partial reconfiguration word, the PR loader software writes the end-of-configuration (EOC) sequence. The EOC sequence consists of three DWs that inform the PR loader state machines to terminate the partial reconfiguration process. After the EOC is received, the partial reconfiguration process is considered complete (signal PR_done asserted). The partial reconfiguration process cannot restart by writing the SOC sequence. Table 2 shows the definitions of the EOC words.

Table 2: **EOC Sequence**

EOC_1	`x454E445Fh`
EOC_2	`x434F4E46h`
EOC_3	`x50434965h`

The EOC timing is shown in Figure 9.

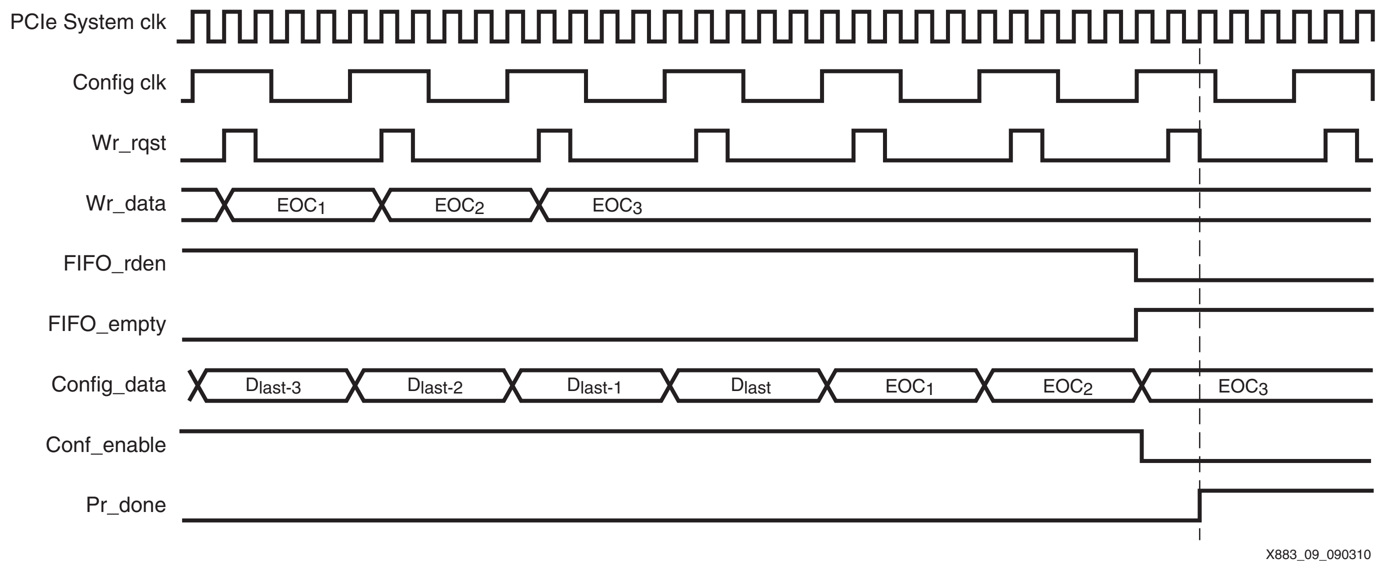


Figure 9: **EOC Timing**

Startup

Asserting the Virtex-6 FPGA internal global reset after partial reconfiguration resets the PCIe system interface. Therefore, it is necessary to use a user-synchronous reset for the user application instead of the global reset. All synchronous elements in the user application should have synchronous reset. The startup module asserts synchronous reset (`RM_trn_reset_n`) throughout partial reconfiguration and for an additional 25 clock cycles after the PCIe system interface has switched over. This ensures that the user application is properly reset.

After the reset, the user application initializes itself to a state ready for engaging PCIe system transactions. It then sends a ready signature (`rdy_sig`) to the startup module to indicate its readiness. The startup module signals (`static_control`) the switcher to enable the PCIe system interface to the reconfigurable partition and disconnects from the static partition. The start-up switchover timing is shown in Figure 10.

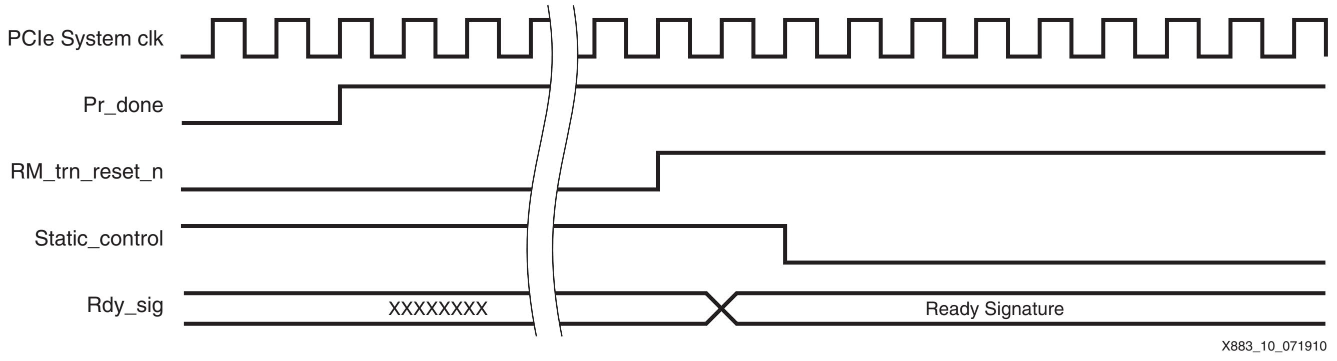


Figure 10: **Startup Switchover Timing**

Reconfigurable Partition — User Application

The user application is taken from the reference design in *Bus Master DMA Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions*. [Ref 5] The bus master DMA (BMD) reference design is selected to show that partial reconfiguration and PCIe system interface switching in the switcher do not:

- Fail on a relatively complex design
- Impact high-performance PCIe system DMA transactions
- Affect the integrity of the PCIe system
- Cause host operating system and application errors

The time needed for partial reconfiguration varies between designs and depends on the size of the reconfigurable partition as well as the PCIe system throughput. The average throughput of the tested systems (see Tested Systems, page 44) is about 30 MB/s. The uncompressed partial reconfiguration file size of this design is about 7.5 MB. The time to load the file based on a 30 MB/s throughput is about 250 ms.

Design Flow Overview

Implementing a partially reconfigurable FPGA design is similar to implementing multiple non-partial reconfiguration designs that share common logic. The designer designates the portions of the FPGA to be reconfigured, both in terms of the physical size of the region and the types of resources desired. Then, the designer describes the different module variants that are to occupy that region. Nearly all FPGA resources are reconfigurable, including block RAM, DSP blocks, and I/O. The Xilinx ISE® software ensures that the resources used to construct the reconfigurable functions are completely contained within the defined physical regions and that no interference with the non-reconfiguring portion of the design occurs.

The PlanAhead design tools manage all the details of building such a design. Multiple netlists, representing the static (non-reconfiguring) logic and each variant of the reconfigurable portions of the design, are loaded in; the designer then creates full FPGA design images with these pieces. Floorplanning, constraint entry, and design rule checks (DRCs) are all accessed through the PlanAhead software environment. Multiple passes through the place-and-route tools are used to generate the necessary bitstreams for all full and partial design images; each pass (called a configuration) represents a complete FPGA design.

Partitions ensure that the logic and routing common to each of the multiple designs is absolutely identical. After one design configuration meets all requirements, the designer can reuse the results from that implementation to create the other configurations. After all configurations are implemented, verification routines validate consistency among all the versions. All these checks combine to guarantee a safe environment when loading a partial bitstream into an operating FPGA.

Partial reconfiguration support in the PlanAhead software is a special license-enabled feature. Visit the Xilinx partial reconfiguration website (http://www.xilinx.com/tools/partial-reconfiguration) for more availability information.

Figure 11 shows the partial reconfiguration design flow summary.

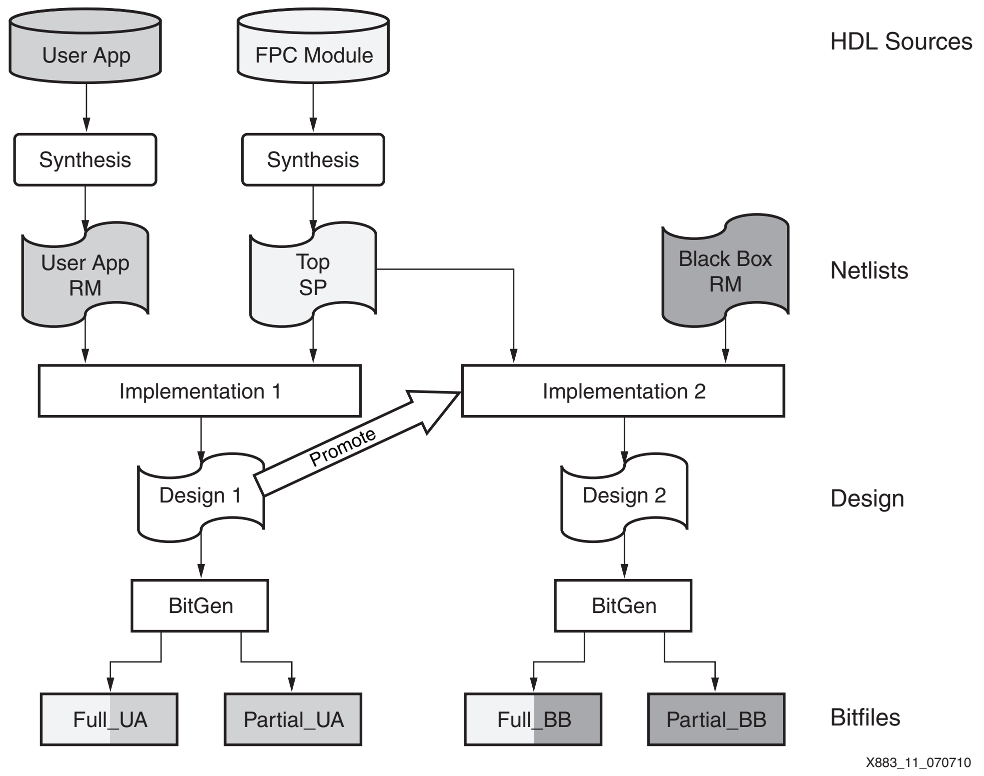


Figure 11: **Partial Reconfiguration Design Flow**

Below is a high-level description of the partial reconfiguration flow:

1. Customize and create Integrated Block for PCI Express in the CORE Generator software.
2. Synthesize the static partition (switcher, PR loader, and Integrated Block for PCI Express) and reconfigurable partition (user application) source codes.
3. Create a PlanAhead software partial reconfiguration project. Import the static partition as the top-level module.
4. Define a reconfigurable module for the user application. Load the user application as the primary reconfigurable module and define a black box variant.
5. Create physical constraints. The reconfigurable partition must have area_group range physical constraints to designate which physical resources are parts of that reconfigurable partition.
6. Run partial reconfiguration design rule checks.
7. Two implementation configurations are created: one for the full design reconfigurable module and one for the black box reconfigurable module.
8. Implement the first configuration for the full design reconfigurable module. Promote the design after completion, then implement the configuration for the black box reconfigurable module.
9. Run BitGen on the first configuration (Config1) to create a full user application (Full_UA) as well as a partial user application (Partial_UA) bitfile, which is loaded during the second stage configuration. The full user application bitfile is not used.
10. Run BitGen on the black box configuration (Config2) to create a partial black box (Partial_BB) and a full black box (Full_BB) bitfile, which is loaded during the first stage configuration. The partial black box bitfile is not used.

Application Software

Designing the PR Loader Drivers

The PR loader drivers are created by the Jungo WinDriver development tool. Two driver files (`wdapi1020.dll` and `PR_PCIe_V6.sys`) are provided in the reference design to support the PR loader application.

Xilinx developed the PR loader driver and GUI application for this reference design using the Jungo WinDriver software. Jungo granted Xilinx a WinDriver license for the purpose of demonstrating the key features of this reference design.

The Jungo WinDriver wizard produces a driver binary (`.sys` file) and C code header files (`.h`) that are used for the software application development. No actual source code for the driver is produced.

The Jungo license agreement explicitly prohibits Xilinx from releasing any files generated by the Jungo software other than the driver binary. Contact Jungo directly for source code availability.

Designing the PR Loader Application

The PR loader application included in the reference design is compiled in Visual Basic 6. A high-level flow diagram is shown in Figure 12. A fixed-size block of the partial reconfiguration file content is copied directly to the target memory range defined in BAR2. This process repeats until the last partial reconfiguration word is written.

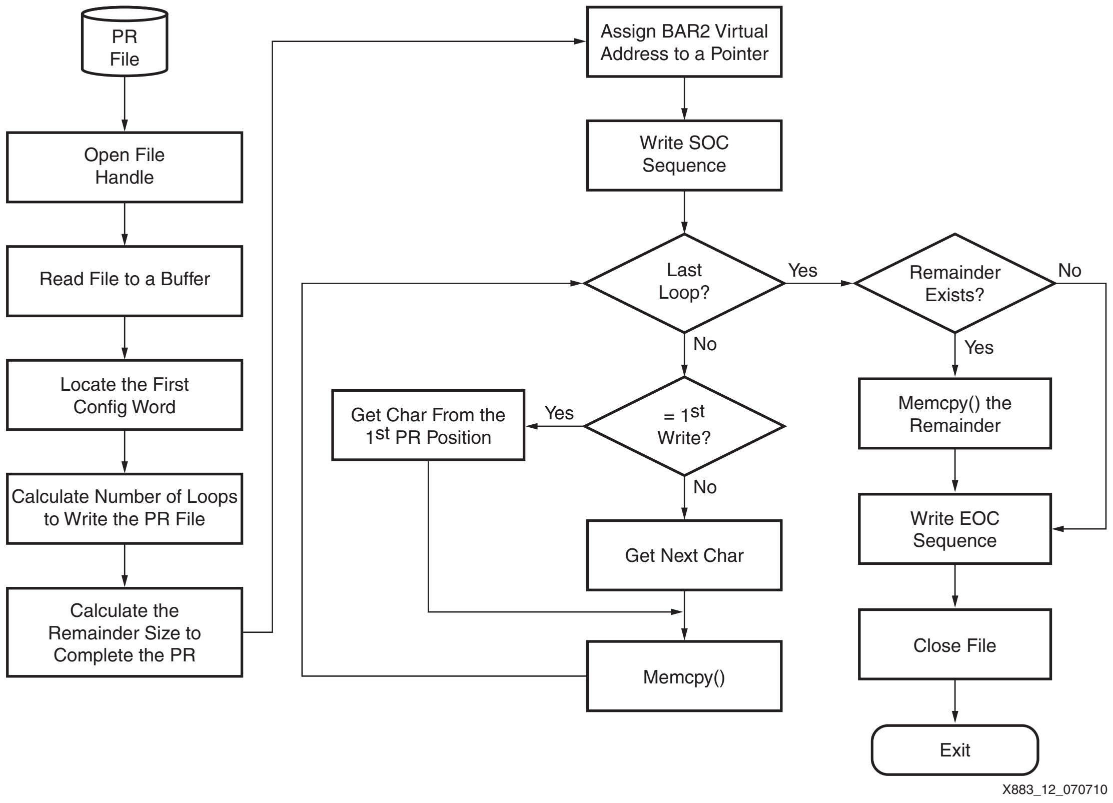


Figure 12: **PR Loader Application Flow**

This is an excerpt of the Visual Basic 6 subroutine responsible for sending the partial reconfiguration file to the FPC module:

```
Private Sub BtnReconfig_Click()
    Dim File_Name As String
    Dim File_Length As Long
    Dim fnum As Integer
    Dim BufferLength As Integer
```

```
'BufferLength defines the size of the write buffer in DWs
'Its size in bytes should not exceed the memory range defined in BAR2
BufferLength = 400

Dim data() As file_buffer
Dim tx_data() As file_buffer
Dim remainder As Integer
Dim LoopNumber As Long

Dim pData As Long
Dim data_length As Long
Dim i As Long
Dim FF_pos As Long
Dim ReadChar As Byte


On Error GoTo ErrHandle

'extract the PR filename from the GUI
With ComDiag
    .Filter = "*.bit|*.bit"
    .CancelError = True
    .ShowOpen
End With
File_Name = ComDiag.FileName

'Find position of first FF and do all calculations based on that character position
fnum = FreeFile 'supply the next available file number
Open File_Name For Binary Access Read As fnum 'Assign file to a file handle
FF_pos = 0 'reset character position
Get #fnum, , ReadChar
While ReadChar <> 255 'read character sequentially until not equal to xFF
    FF_pos = FF_pos + 1 'Keep track of the character position
    Get #fnum, , ReadChar 'Get one character at a time
Wend
FF_pos = FF_pos + 1
Close fnum


File_Length = FileLen(File_Name)
File_Length = File_Length / 4 'Gives the number how many Longs need to be read

fnum = FreeFile


ReDim data(1 To BufferLength)
ReDim tx_data(1 To BufferLength)

remainder = File_Length Mod BufferLength
LoopNumber = File_Length / BufferLength

'Calling the Jungo driver to assign virtual address of BAR2 to pData
pData = (hML507_MK.cardReg.Card.Item(hML605_MK.addrDesc(ML605_MK_AD_BAR2).Index).dw4)
data_length = UBound(data) * (4 * LenB(data(0).bytes(0)))

Dim dbg_file As String
Open File_Name For Binary Access Read As #fnum
i = 1

'Write Start-of-Configuration sequence to FPC module
'data_length=12, 3 words with 4 bytes each
' Start_Word has the SOC sequence definitions
Call memcpy(pData, VarPtr(Start_Word(0)), 12)

While i <= LoopNumber
    ReDim data(1 To BufferLength)
    If i = 1 Then
        Get #fnum, FF_pos, data
    Else
        Get #fnum, , data
    End If
    ' write to the memory mapped range directly
    Call memcpy(pData, VarPtr(data(1)), data_length)
    i = i + 1
Wend
```

```
        If remainder > 0 Then
            ReDim data(1 To remainder)
            Get #fnum, , data
    ' write to the memory mapped range directly
            Call memcpy(pData, VarPtr(data(1)), 4 * remainder)
        End If

        'Write End-of-Configuration sequence to FPC module
        Call memcpy(pData, VarPtr(End_Word(0)), 12)

        Close #fnum

        Exit Sub
    ErrHandle:
        Err.Clear
        On Error GoTo 0
        Exit Sub

    End Sub
```

Simulation

The reference design includes resources to support functional simulation in the ModelSim simulator. See the *Virtex-6 FPGA Integrated Block for PCI Express User Guide* [Ref 1] for further information.

Figure 13 shows the board-level simulation setup for the design. The PCI Express Root Port Model is a test bench environment for the test program interface. It provides a source mechanism for generating downstream TLP traffic to stimulate the target FPC module and a destination mechanism for receiving upstream TLP traffic from the FPC module.

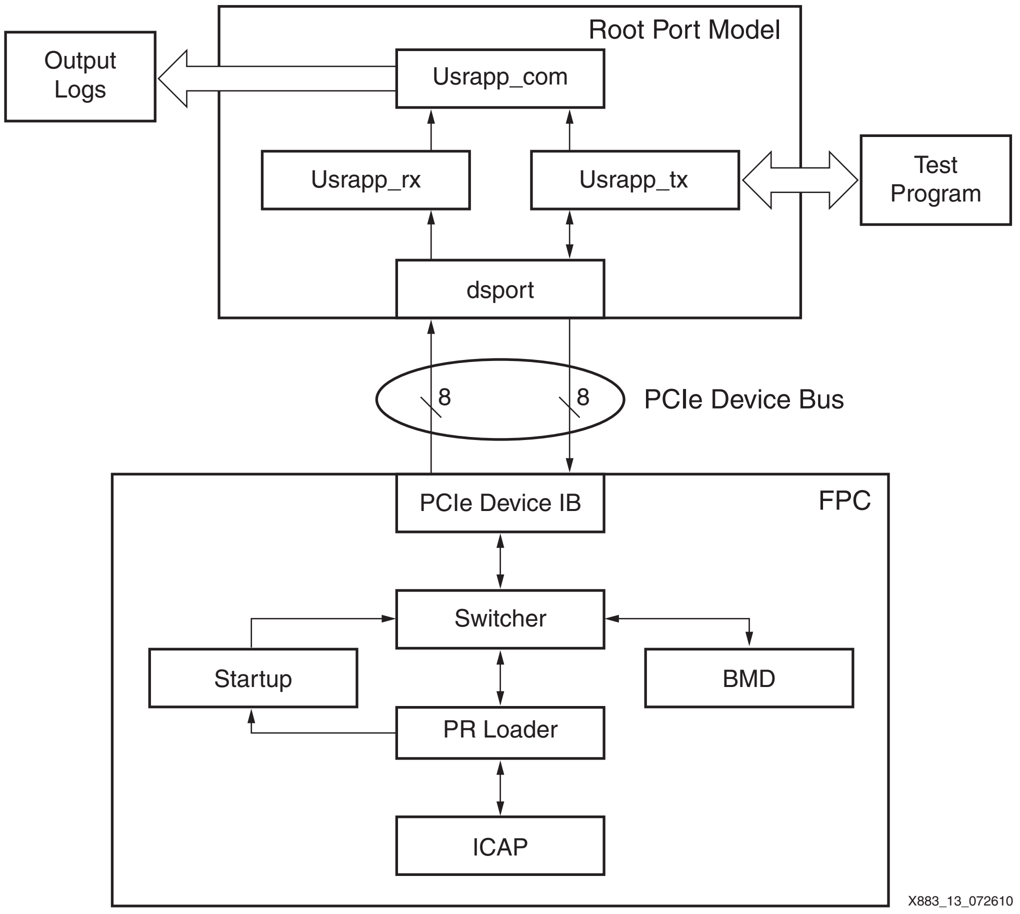


Figure 13: **Root Port Model and FPC Design**

Source code for the Root Port Model is included to provide a starting point for a custom test bench. This code initializes the configuration space of the Integrated Block for PCI Express core, creates TLP transactions, generates TLP logs, and provides an interface for creating and verifying tests.

The Root Port Model consists of these blocks:

- `dsport` (Downstream Root Port)
- `usrapp_tx`
- `usrapp_rx`
- `usrapp_com`

The `usrapp_tx` block sends TLPs to the `dsport` block for transmission across the PCIe system link to the FPC module. In turn, the FPC module transmits TLPs across the PCIe system link to the `dsport` block, which is subsequently passed to the `usrapp_rx` block. The `dsport` and core are responsible for the data link layer and physical link layer processing when communicating across the PCIe device bus. Both the `usrapp_tx` and `usrapp_rx` utilize the `usrapp_com` block for shared functions—for example, TLP processing and log file outputting. Transaction sequences or test programs are initiated by the `usrapp_tx` block to stimulate the endpoint device's bus interface. TLP responses from the endpoint device are received by the `usrapp_rx` block. Communication between the `usrapp_tx` and `usrapp_rx` blocks allows the `usrapp_tx` block to verify correct behavior and act accordingly when the `usrapp_rx` block has received TLPs from the endpoint device.

The Root Port Model creates three output files (`tx.dat`, `rx.dat`, and `error.dat`) during each simulation run. Log files `rx.dat` and `tx.dat` each contain a detailed record of every TLP that was received and transmitted, respectively, by the Root Port Model. With an understanding of the expected TLP transmission during a specific test case, users can easily isolate the failure.

The log file `error.dat` is used in conjunction with the expectation tasks. Test programs that utilize the expectation tasks generate a general error message to standard output.

Running Functional Simulation

1. Change the directory to `<`*`project_directory`*`>/sim`.
2. Modify these logical libraries in `modelsim.ini` to point to the physical library paths compiled in the system:
   - `secureip`
   - `simprims_ver`
   - `unisims_ver`
   - `xilinxcorelib_ver`
   - `unimacro_ver`
3. Launch the ModelSim simulator.
4. In the console, run the script file by entering this command:

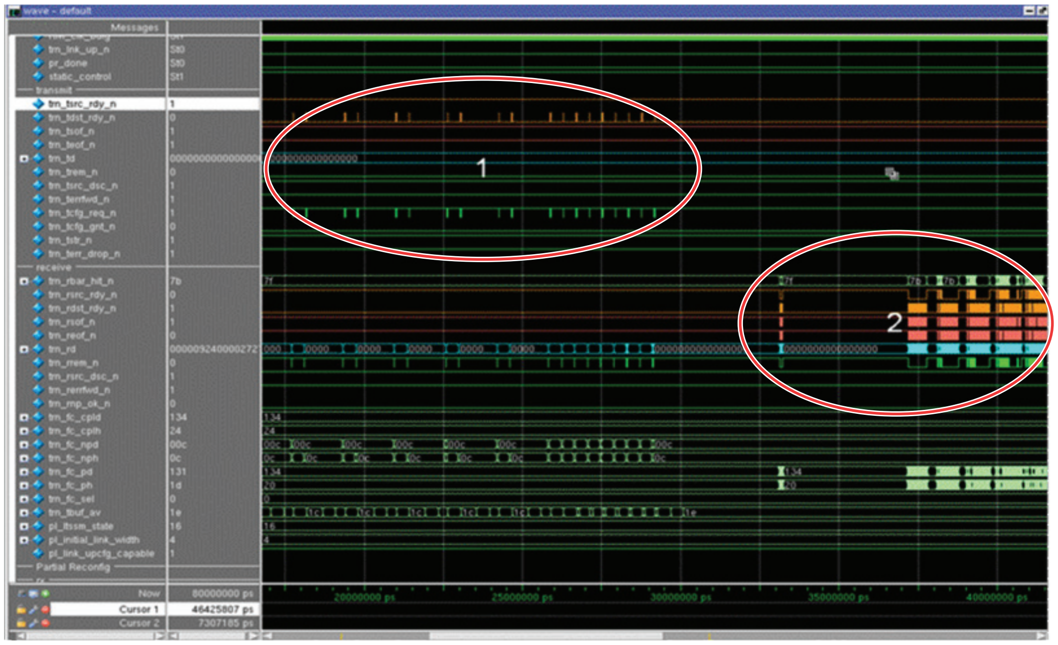
   ```
   do simulate_mti.do
   ```
5. The script compiles source codes and launches the simulator. The simulation runs for about 80 µs. The results are displayed in several waveform windows.

Simulation Description

The simulation mimics an actual FPC module use scenario from PCIe system initialization to DMA read data transfer. The different stages of the simulation are described here:

1. Prior to this state, the static partition has already been configured and is ready for enumeration. When the PCIe system link is up and ready, the Root Port Model issues Type 0 configuration requests to initialize BARs in the Integrated Block for PCI Express.

2. The host begins sending the partial reconfiguration bitstream in the form of packetized memory write request (MWr) TLPs. The first few TLPs contain the SOC sequence followed by the actual partial reconfiguration contents (see Figure 14).



X883_14_070510

Figure 14: **Simulation Waveform (Beginning)**

3. After sending the last partial reconfiguration word, the host releases the EOC sequence and signals the PR loader to terminate the partial reconfiguration process. This is indicated by the pr_done signal. Soon after, the signal static_control, which controls the PCIe system interface connections, switches over to the BMD (see Figure 15).

   Note: The partial reconfiguration bitstream in this simulation is a sample of a real bitstream. It does not contain valid configuration words and its size is also significantly reduced.

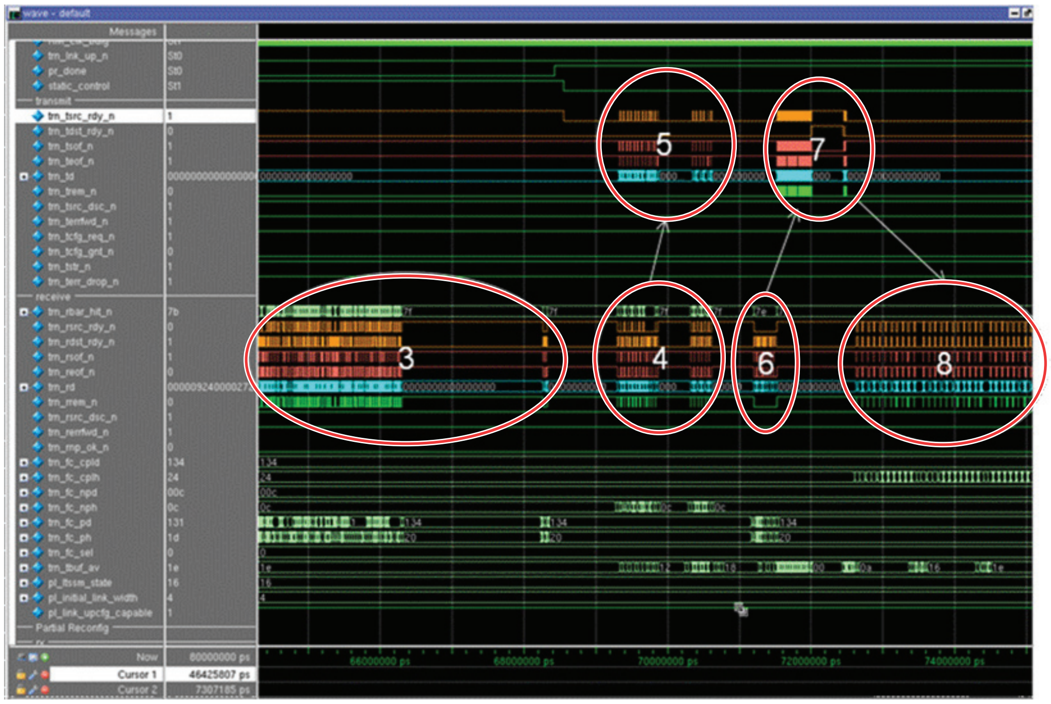


Figure 15: **Simulation Waveform (End)**

4. The host reads the BMD design descriptor registers in the form of memory read request (MRd) TLPs.
5. The BMD responds by completing the read request with the register contents.
6. The host programs the BMD design descriptor registers to set up a DMA read request.
7. The BMD acknowledges the DMA read request and begins sending a sequence of back-to-back MRd TLPs.
8. The Root Port Model responds by sending completion with data (CplD) TLPs.

Note: The simulation validates the FPC module and the user-defined functionality. It does not fully validate the ICAP functionality.

Building the Project

System Requirements

The following are required for the reference design:

- ML605 demonstration platform (XC6VLX240T-FF1156 production device)
- ISE Design Suite 12.3 or newer. Partial reconfiguration is a license-enabled feature
- X86 system with Windows XP Service Pack 3 installed

Step 1 — Extract the Reference Design

1. Extract the contents of the reference design zip file `xapp883_Fast_Config_PCIe.zip` to the desired *`<project_directory>`*

   Note: *`<project_directory>`* is the project root directory under which all directory paths throughout this section reside.

Step 2 — Generate the Integrated Block for PCI Express

1. Launch the CORE Generator software.

2. Select **File** → **New Project**. Create a new CORE Generator software project in `\hw\coregen\XAPP883.cgp`.
3. Select **Project** → **Project Options**. Specify these settings:

   Part:

   - Family: Virtex6
   - Device: xc6vlx240t
   - Package: FF1156
   - Speed Grade: -1

   Generation:

   - Design Entry: Verilog
   - Simulation Files: Behavioral
   - Other Output Products: Uncheck ASY Symbol File
   - Flow Settings
     - Vendor: Other

   Leave the other settings at their default values and click **OK**.
4. Select **Standard Bus Interfaces** → **PCI Express** → **Virtex-6 Integrated Block for PCI Express version 1.6** under the IP catalog pane. This launches the Virtex-6 FPGA Integrated Block for PCI Express wizard.
5. Set up the PCIe system lane width:

   Name the Integrated Block for PCI Express `v6_pcie_v1_6` in the Component Name field. Select **PCI Express Endpoint device** in the Device/Port Type pull-down menu. Select **X8** in the Lane Width pull-down menu. Leave the other settings at their default values.
6. Set up the base address registers:

   Select **BAR0** and specify the following:

   - Type: memory
   - 64 bit: uncheck
   - Size: 1 Kilobytes

   BAR0 is the base address definition for the user application (BMD reference design).

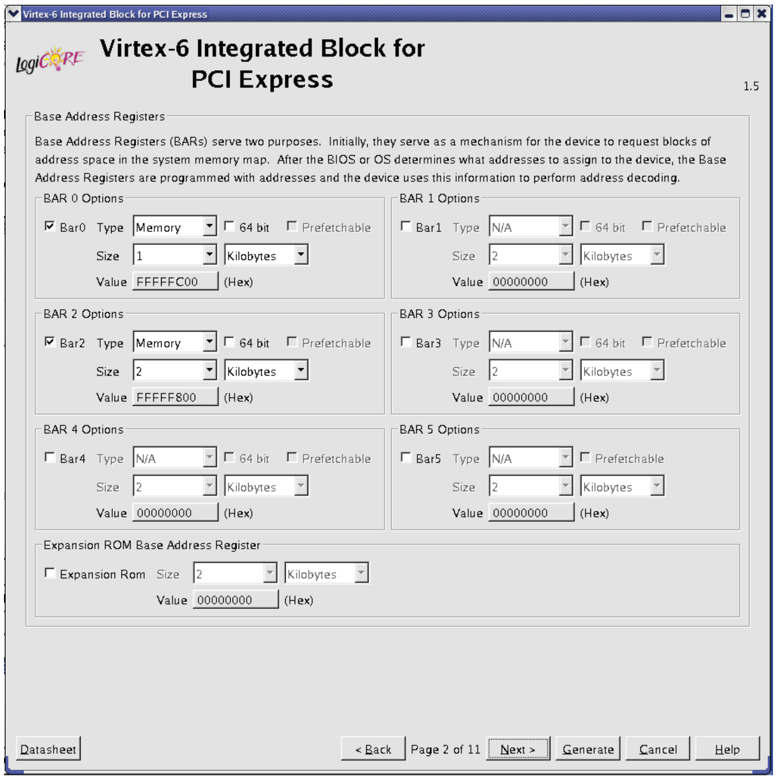
   Select **BAR2** and set the following:

   - Type: memory
   - 64 bit: uncheck
   - Size: 2 Kilobytes

   BAR2 is the base address definition for the PR loader.

   Leave the other settings at their default values.

   The page should look similar to Figure 16. Click **Next** to proceed to the next page.



Virtex-6 Integrated Block for PCI Express 1.5

Base Address Registers

Base Address Registers (BARs) serve two purposes. Initially, they serve as a mechanism for the device to request blocks of address space in the system memory map. After the BIOS or OS determines what addresses to assign to the device, the Base Address Registers are programmed with addresses and the device uses this information to perform address decoding.

BAR 0 Options: [✓ Bar0] Type Memory; 64 bit; Prefetchable; Size 1 Kilobytes; Value FFFFFC00 (Hex)

BAR 1 Options: Bar1 Type N/A; 64 bit; Prefetchable; Size 2 Kilobytes; Value 00000000 (Hex)

BAR 2 Options: [✓ Bar2] Type Memory; 64 bit; Prefetchable; Size 2 Kilobytes; Value FFFFF800 (Hex)

BAR 3 Options: Bar3 Type N/A; 64 bit; Prefetchable; Size 2 Kilobytes; Value 00000000 (Hex)

BAR 4 Options: Bar4 Type N/A; 64 bit; Prefetchable; Size 2 Kilobytes; Value 00000000 (Hex)

BAR 5 Options: Bar5 Type N/A; Prefetchable; Size 2 Kilobytes; Value 00000000 (Hex)

Expansion ROM Base Address Register: Expansion Rom Size 2 Kilobytes; Value 00000000 (Hex)

X883_16_070510

Figure 16: **Setting Up Base Address Registers**

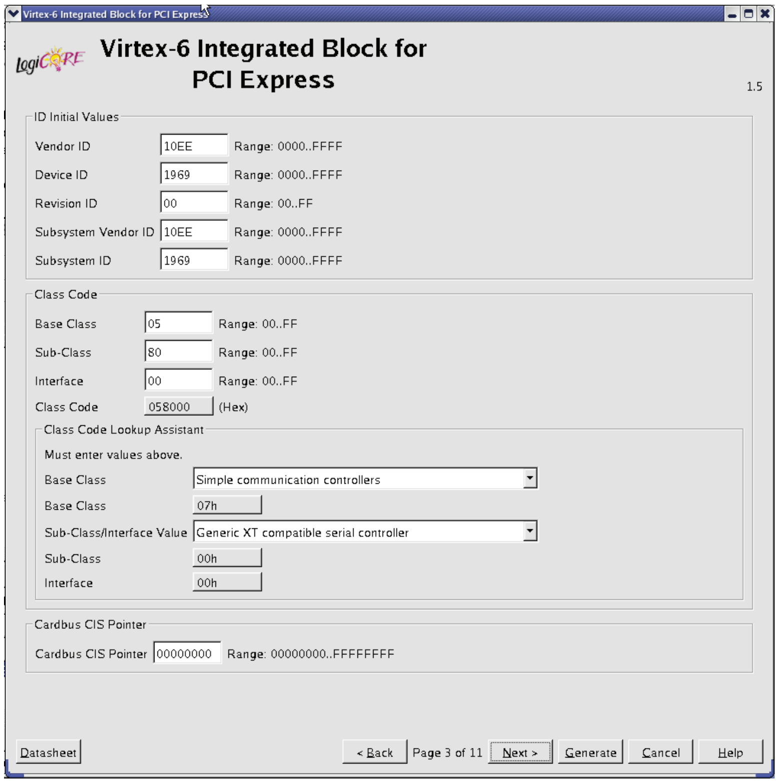
7. Set up the PCIe system ID and class:

   Specify the following on page 3 of the wizard.

   Note: The PR loader driver does not recognize the FPC module if these settings are not correctly matched.

   - Vendor ID: 10EE
   - Device ID: 1969
   - Revision ID: 00
   - Subsystem Vendor ID: 10EE
   - Subsystem ID: 1969
   - Base Class: 05
   - Sub-Class: 80
   - Interface: 00
   - Cardbus CIS Pointer: 00000000

   The result should look like Figure 17. Click **Next** to proceed.



Virtex-6 Integrated Block for PCI Express

ID Initial Values

Vendor ID	10EE	Range: 0000..FFFF
Device ID	1969	Range: 0000..FFFF
Revision ID	00	Range: 00..FF
Subsystem Vendor ID	10EE	Range: 0000..FFFF
Subsystem ID	1969	Range: 0000..FFFF

Class Code

Base Class	05	Range: 00..FF
Sub-Class	80	Range: 00..FF
Interface	00	Range: 00..FF
Class Code	058000	(Hex)

Class Code Lookup Assistant

Must enter values above.

Base Class	Simple communication controllers
Base Class	07h
Sub-Class/Interface Value	Generic XT compatible serial controller
Sub-Class	00h
Interface	00h

Cardbus CIS Pointer

Cardbus CIS Pointer 00000000 Range: 00000000..FFFFFFFF

X883_17_070510

Figure 17: **Setting Up Device Configuration Parameters**

8. Set up the payload size:

   Select **512 bytes** in the Max Payload Size pull-down menu under Device Capabilities Register. Click **Next** until reaching the Pinout Selection page (Page 9).

9. Set up the pinout selection:

   Select **ML 605** in the Xilinx Development Board pull-down menu under Xilinx Development Boards. The PCIe Block Location changes to `X0Y0` automatically. Click **Next** until reaching Advanced Setting 2 (Page 11).

10. Set up the reference clock:

    Select **100MHz** in the Frequency (MHz) pull-down menu under Reference Clock Frequency.

11. Click **Generate** to create the Integrated Block for PCI Express. Close the readme window after reading the information, but keep the CORE Generator software open to generate the next core.

12. Copy the source codes from the `Coregen` directory:

    ```
    Copy hw\Coregen\v6_pcie_v1_6\source\* hw\Source\PCIe_x8_gen1\source
    ```

13. Copy the source codes from the `custom` directory to the `source` directory:

    ```
    Copy /y hw\Source\PCIe_x8_gen1\source\custom\* hw\Source\PCIe_x8_gen1\source
    ```

    The custom codes modify the PCIe system clock module to include extra clocks for the PR loader. The code reduces the use of the MMCM and simplifies the clocking scheme.

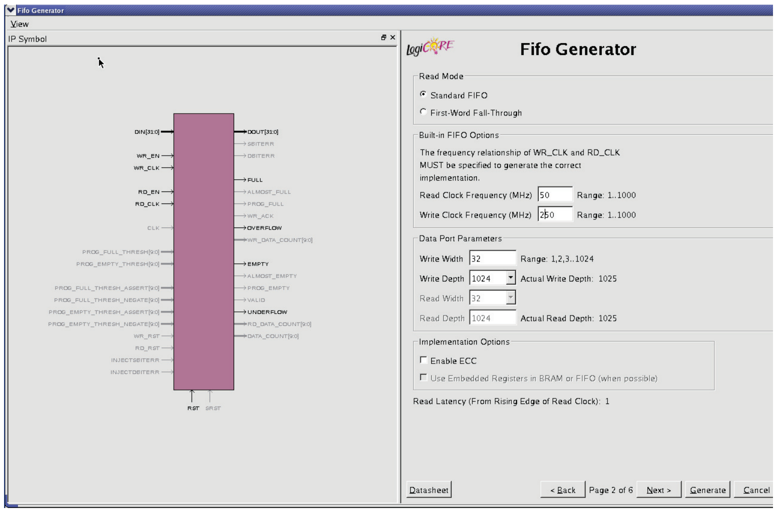
14. Select **Memories & Storage Elements** → **FIFOs** → **FIFO Generator version 6.2** under the IP catalog pane. This brings up the FIFO Generator wizard.

15. Specify the Component Name to be **config_FIFO**. Select **Independent Clocks (RD_CLK, WR_CLK) Built-in FIFO**. Click **Next**.

16. Select **Standard FIFO** under Read Mode.

17. Specify **250** for Write Clock Frequency (MHz) and **50** for Read Clock Frequency (MHz).
18. Under Data Port Parameters, specify the Write Width to be **32** and Write Depth to be **1024**.

    The result should look like Figure 18. Click **Next**.



Fifo Generator

Read Mode
- (●) Standard FIFO
- ( ) First-Word Fall-Through

Built-in FIFO Options

The frequency relationship of WR_CLK and RD_CLK MUST be specified to generate the correct implementation.

Read Clock Frequency (MHz)	50	Range: 1..1000
Write Clock Frequency (MHz)	250	Range: 1..1000

Data Port Parameters

Write Width	32	Range: 1,2,3..1024
Write Depth	1024	Actual Write Depth: 1025
Read Width	32	
Read Depth	1024	Actual Read Depth: 1025

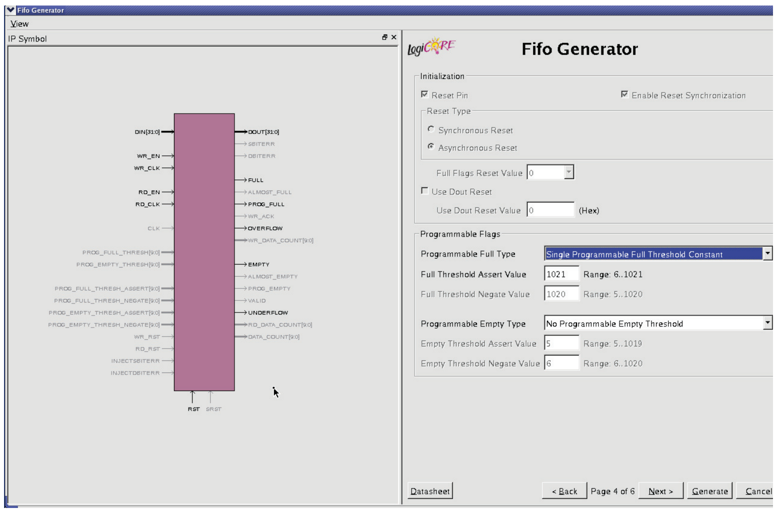
Implementation Options
- [ ] Enable ECC
- [ ] Use Embedded Registers in BRAM or FIFO (when possible)

Read Latency (From Rising Edge of Read Clock): 1

X883_18_070510

Figure 18: **FIFO Generator Wizard, Page 2**

19. Select **Overflow Flag** and **Active High**. Select **Underflow Flag** and **Active High**. Click **Next**.
20. Select **Single Programmable Full Threshold Constant** in the Programmable Full Type pull-down menu. Click **Generate**. The result should look like Figure 19.



Fifo Generator

Initialization
- [✓] Reset Pin
- [✓] Enable Reset Synchronization

Reset Type
- ( ) Synchronous Reset
- (●) Asynchronous Reset

Full Flags Reset Value: 0
- [ ] Use Dout Reset

Use Dout Reset Value: 0 (Hex)

Programmable Flags

Programmable Full Type	Single Programmable Full Threshold Constant	
Full Threshold Assert Value	1021	Range: 6..1021
Full Threshold Negate Value	1020	Range: 5..1020
Programmable Empty Type	No Programmable Empty Threshold	
Empty Threshold Assert Value	5	Range: 5..1019
Empty Threshold Negate Value	6	Range: 6..1020

X883_19_070510

Figure 19: **FIFO Generator Wizard, Page 4**

Step 3 — Synthesize the Design

1. Change directory to `\hw\Tools`.
2. At the command prompt, run this command:

   ```
   xtclsh xpartition.tcl data.tcl
   ```

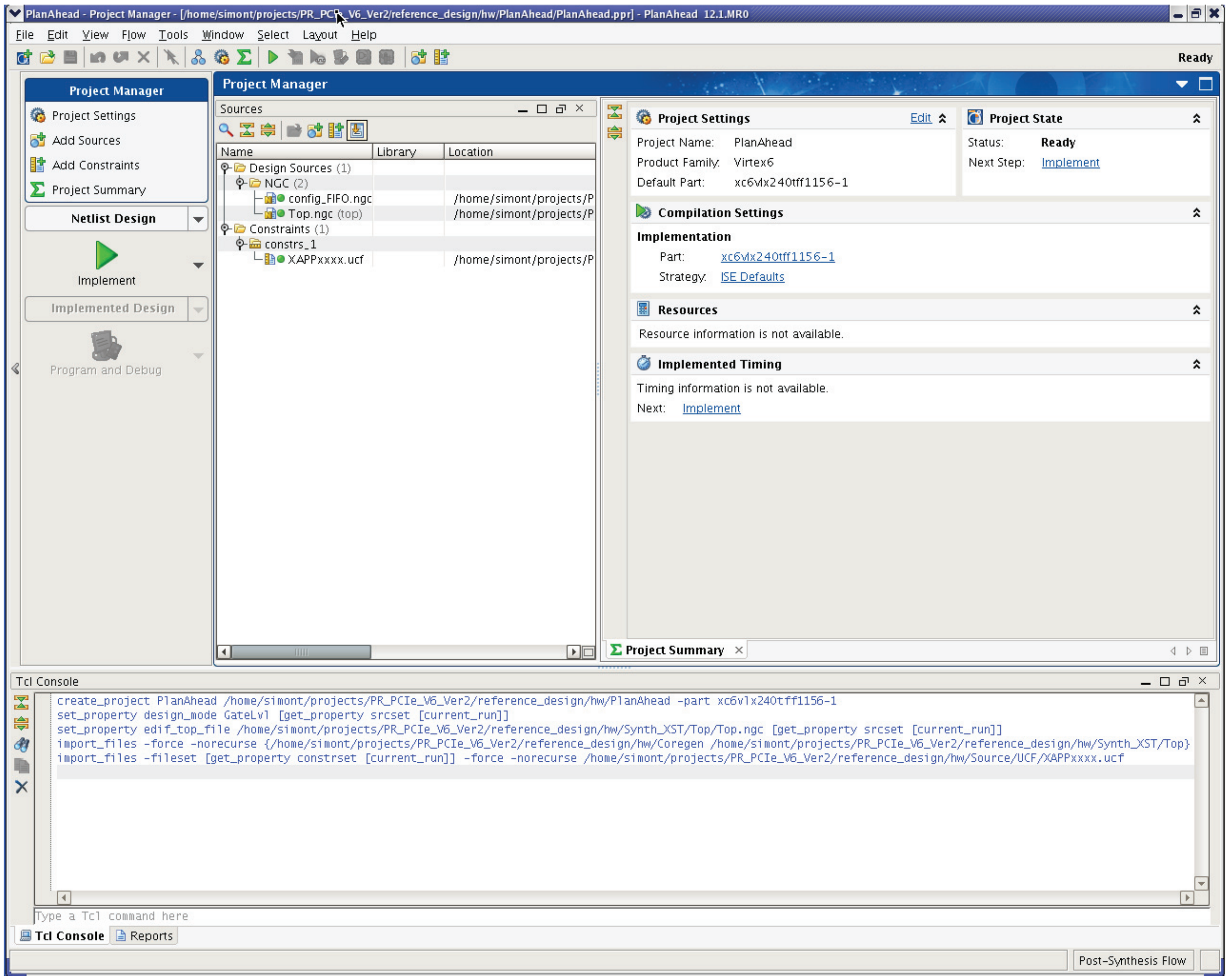
   This launches the Xilinx Synthesis Technology (XST) in batch mode and synthesizes the reference design. These messages appear after a successful run:

   ```
   **** Synthesizing with XST ****
   xst -ifn Top.xst
   xst -ifn UserDesign.xst
   ```

Step 4 — Create the Partial Reconfiguration Project

1. Start the PlanAhead software.
2. Click **Create New Project** and click **Next**. Set a new project name and location; this reference design uses `\hw\PlanAhead`.
3. Select **Specify synthesized (EDIF or NGC) netlist**. Click **Next**.
4. Browse to the netlist and select `\hw\Synth_XST\Top\Top.ngc` in the Top Netlist File field.
5. Click **Add Directories** and select `\hw\Coregen` as the netlist search directory. Click **Select**. Click **Next**.
6. Click **Add Files...** Set the constraints file to `\hw\Source\UCF\XAPP883_ML605.ucf`, and click **Next**.
7. Select the device **xc6vlx240tff1156-1** by scrolling through the list or by using the filters to narrow the part selection. Changing the part in a partial reconfiguration project is not supported.
8. Review the project summary and click **Finish**.

At this point, the project has no modifications for partial reconfiguration; it is a regular FPGA design and PlanAhead project. The PlanAhead environment should look like Figure 20.



PlanAhead - Project Manager - [/home/simont/projects/PR_PCIe_V6_Ver2/reference_design/hw/PlanAhead/PlanAhead.ppr] - PlanAhead 12.1.MR0

Name	Library	Location
Design Sources (1)		
NGC (2)		
config_FIFO.ngc		/home/simont/projects/P
Top.ngc (top)		/home/simont/projects/P
Constraints (1)		
constrs_1		
XAPPxxxx.ucf		/home/simont/projects/P

Project Settings

Project Name: PlanAhead

Product Family: Virtex6

Default Part: xc6vlx240tff1156-1

Project State

Status: Ready

Next Step: Implement

Compilation Settings

Implementation

Part: xc6vlx240tff1156-1

Strategy: ISE Defaults

Resources

Resource information is not available.

Implemented Timing

Timing information is not available.

Next: Implement

```
create_project PlanAhead /home/simont/projects/PR_PCIe_V6_Ver2/reference_design/hw/PlanAhead -part xc6vlx240tff1156-1
set_property design_mode GateLvl [get_property srcset [current_run]]
set_property edif_top_file /home/simont/projects/PR_PCIe_V6_Ver2/reference_design/hw/Synth_XST/Top/Top.ngc [get_property srcset [current_run]]
import_files -force -norecurse {/home/simont/projects/PR_PCIe_V6_Ver2/reference_design/hw/Coregen /home/simont/projects/PR_PCIe_V6_Ver2/reference_design/hw/Synth_XST/Top}
import_files -fileset [get_property constrset [current_run]] -force -norecurse /home/simont/projects/PR_PCIe_V6_Ver2/reference_design/hw/Source/UCF/XAPPxxxx.ucf
```

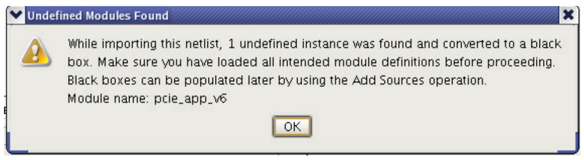
X883_20_070510

Figure 20: **Initial PlanAhead Tool Project**

Step 5 — Load the Netlist Planner

1. Select **Flow** → **Netlist Design**, or click the **Netlist Design** tab on the left side of the window.

   The top-level netlist being loaded into the PlanAhead tool is seen. When it finishes loading, an "Undefined Modules Found" warning message and a "Constraint File Errors" warning message are displayed, as shown in Figure 21 and Figure 22.



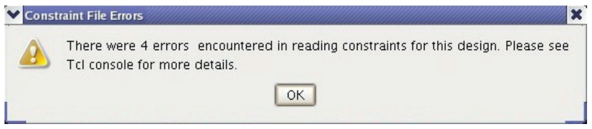
Undefined Modules Found

While importing this netlist, 1 undefined instance was found and converted to a black box. Make sure you have loaded all intended module definitions before proceeding. Black boxes can be populated later by using the Add Sources operation.

Module name: pcie_app_v6

X883_21_070510

Figure 21: **Undefined Instance Warning**


Constraint File Errors

There were 4 errors encountered in reading constraints for this design. Please see Tcl console for more details.

X883_22_111010

Figure 22: **Constraint File Errors Message**

These messages are expected, and are the result of not pointing the New Project wizard to the locations of these files. This is done in the subsequent steps.

2. Click **OK** on the warning message.
3. Set the partial reconfiguration attribute for the project. Select the **File** → **Set PR Project** menu command.

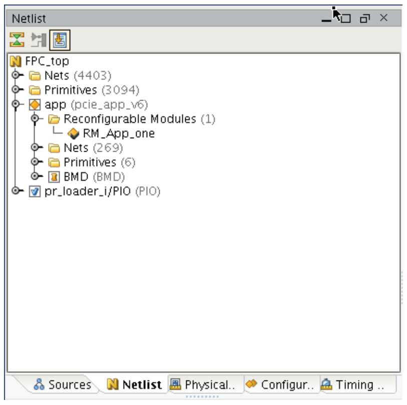
   Note: This menu command is visible only if a valid partial reconfiguration license exists. The bottom-right status bar changes from `Post-Synthesis Flow` to `Partial Reconfiguration Flow`.

Step 6 — Create the Reconfigurable Partition

1. In the Netlist frame, right-click `app(pcie_app_v6)` and select **Set Partition**. This launches the Set Partition wizard.
2. Click **Next** to select whether or not the partition is reconfigurable. Because `app` does not have a netlist associated with it, it is a black box and therefore must be a reconfigurable partition.
3. Click **Next** and name the module `RM_App_one`. Make sure the selection is set to **Netlist already available for this Reconfigurable Module** and click **Next** again.
4. Set the top netlist file to `hw\Synth_XST\UserDesign\UserDesign.ngc`, and click **Next**.

   Note: It is not necessary to set the optional netlist directories. This option is available for reconfigurable modules that have lower-level netlists associated with them.
5. Click **Next** again.

   Note: It is not necessary to set the optional constraint files. This option is available for reconfigurable modules that have module-level constraint files associated with them.
6. This last screen is the Set Partition summary. Click **Finish** to complete this task.
7. The icon for `app` in the Netlist frame changes to an orange diamond in a white box, indicating that the reconfigurable module netlist has been added to the project. The orange diamond indicates that it is a reconfigurable partition. The partial reconfiguration project now has one reconfigurable module and one elaborated netlist, as shown in Figure 23.


Netlist

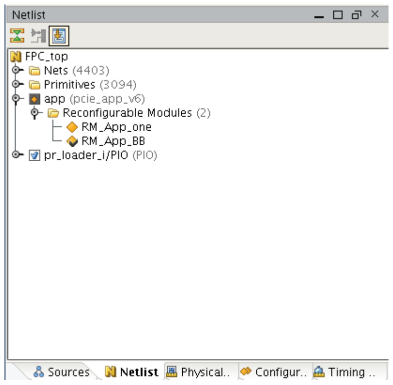
```
FPC_top
  Nets (4403)
  Primitives (3094)
  app (pcie_app_v6)
    Reconfigurable Modules (1)
      RM_App_one
    Nets (269)
    Primitives (6)
    BMD (BMD)
  pr_loader_i/PIO (PIO)
```

X883_23_111010

Figure 23: **Adding Reconfigurable Modules**

Step 7 — Create the Black Box Reconfigurable Module

1. Right-click `app` in the netlist frame and select **Add Reconfigurable Module...** This brings up the Add Reconfigurable Module wizard.
2. Click **Next** and name the module `RM_App_BB`.
3. Select **Add this Reconfigurable module as a black box without a netlist** and click **Next**.
4. Click **Finish** (see Figure 24).



```
FPC_top
  Nets (4403)
  Primitives (3094)
  app (pcie_app_v6)
    Reconfigurable Modules (2)
      RM_App_one
      RM_App_BB
  pr_loader_i/PIO (PIO)
```

X883_24_111010

Figure 24: **Adding the Black Box**

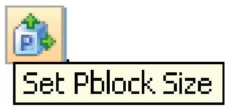
5. Right-click on `RM_App_one` and select **Set as Active Reconfigurable Module**. These actions load the netlists into the PlanAhead software and prepare the design for the first configuration.

Step 8 — Create the Physical Constraints

The reconfigurable partition `app` must have area_group range physical constraints to designate which physical resources are parts of that reconfigurable partition. All physical resources *not* part of the area_group range of a reconfigurable partition are part of the static logic. (Static logic is unaffected by partial reconfiguration and remains operational during the reconfiguration process.) The area_group range constraints should not be created until the reconfigurable partition has been created with the **Set Reconfigurable Partition...** command, as described in Step 6 — Create the Reconfigurable Partition, page 24.

The area_group range constraints can be added to the user constraints file (UCF) with a text editor or by using a graphical floorplanning tool such as the PlanAhead software.

1. In the Physical Constraints frame, select the reconfigurable partition `pblock_app`.
2. To the left of the floorplan frame, click the **Set Pblock Size** icon (see Figure 25).

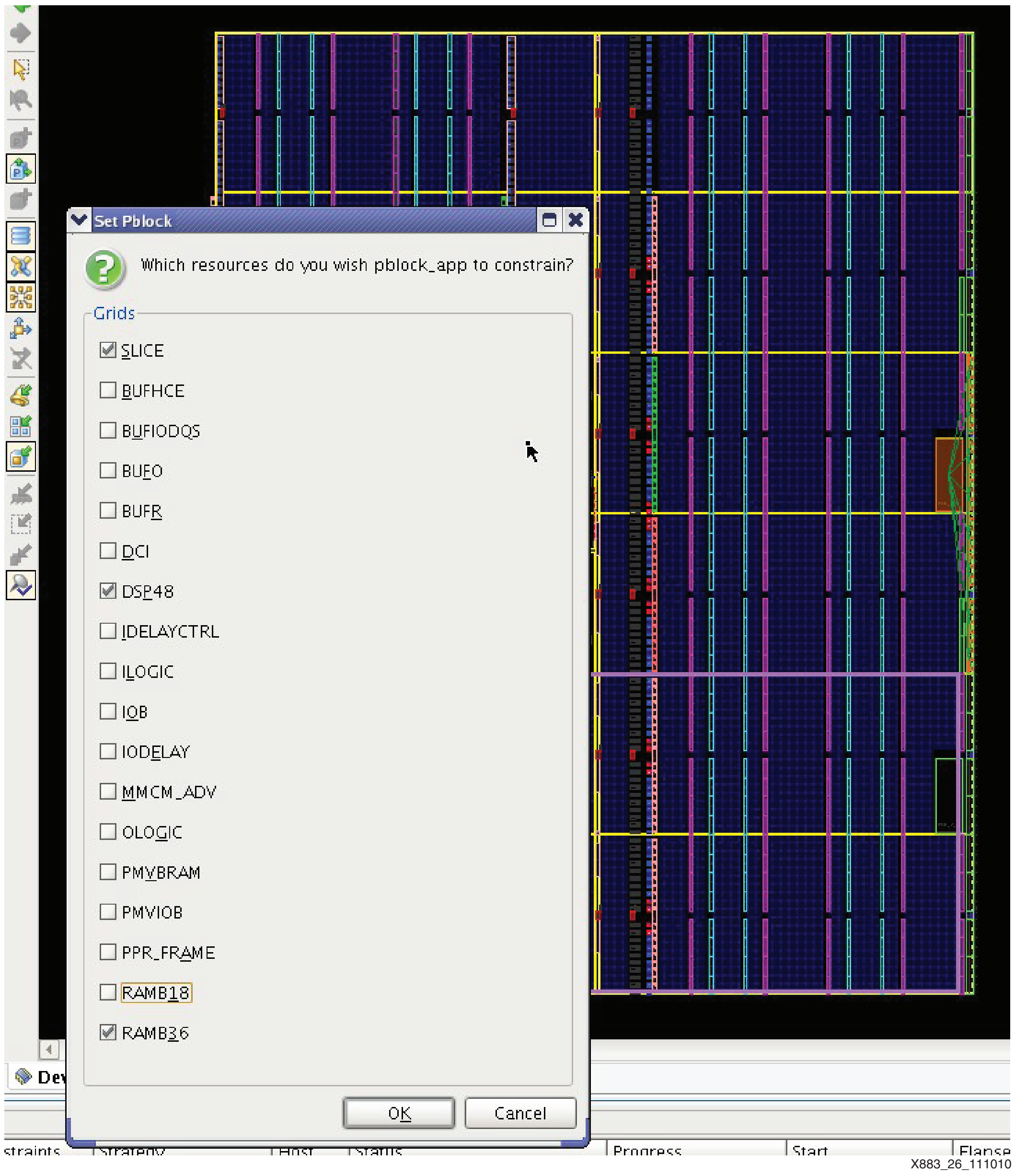


X883_25_093010

Figure 25: **Set Pblock Size**

Note: Ensure that the **Draw Pblock** function is *not* clicked; its icon looks similar to **Set Pblock Size**. As an alternative to clicking the icon, the user can right-click `pblock_app` and then select **Set Pblock Size** from the menu.

3. Drag a box that encompasses SLICE_X0Y0 through SLICE_X123Y239.
4. After drawing the bounding box, select only the **SLICE**, **DSP48**, and **RAMB36** resources for the area group (see Figure 26).


Set Pblock

Which resources do you wish pblock_app to constrain?

Grids

- [x] SLICE
- [ ] BUFHCE
- [ ] BUFIODQS
- [ ] BUFO
- [ ] BUFR
- [ ] DCI
- [x] DSP48
- [ ] IDELAYCTRL
- [ ] ILOGIC
- [ ] IOB
- [ ] IODELAY
- [ ] MMCM_ADV
- [ ] OLOGIC
- [ ] PMVBRAM
- [ ] PMVIOB
- [ ] PPR_FRAME
- [ ] RAMB18
- [x] RAMB36

Figure 26: **Setting Pblock Resources**

5. Click the **Add Pblock Rectangle** button and draw two more boxes from Slice_X124Y0 to Slice_X161Y78 and from Slice_X124Y200 to X161Y239. Select only the **SLICE**, **DSP48**, and **RAMB36** resources.

   Xilinx recommends aligning the partition boundary to the clock region boundary. See the Known Issues section for more details. The floorplan of the entire design should resemble Figure 27.

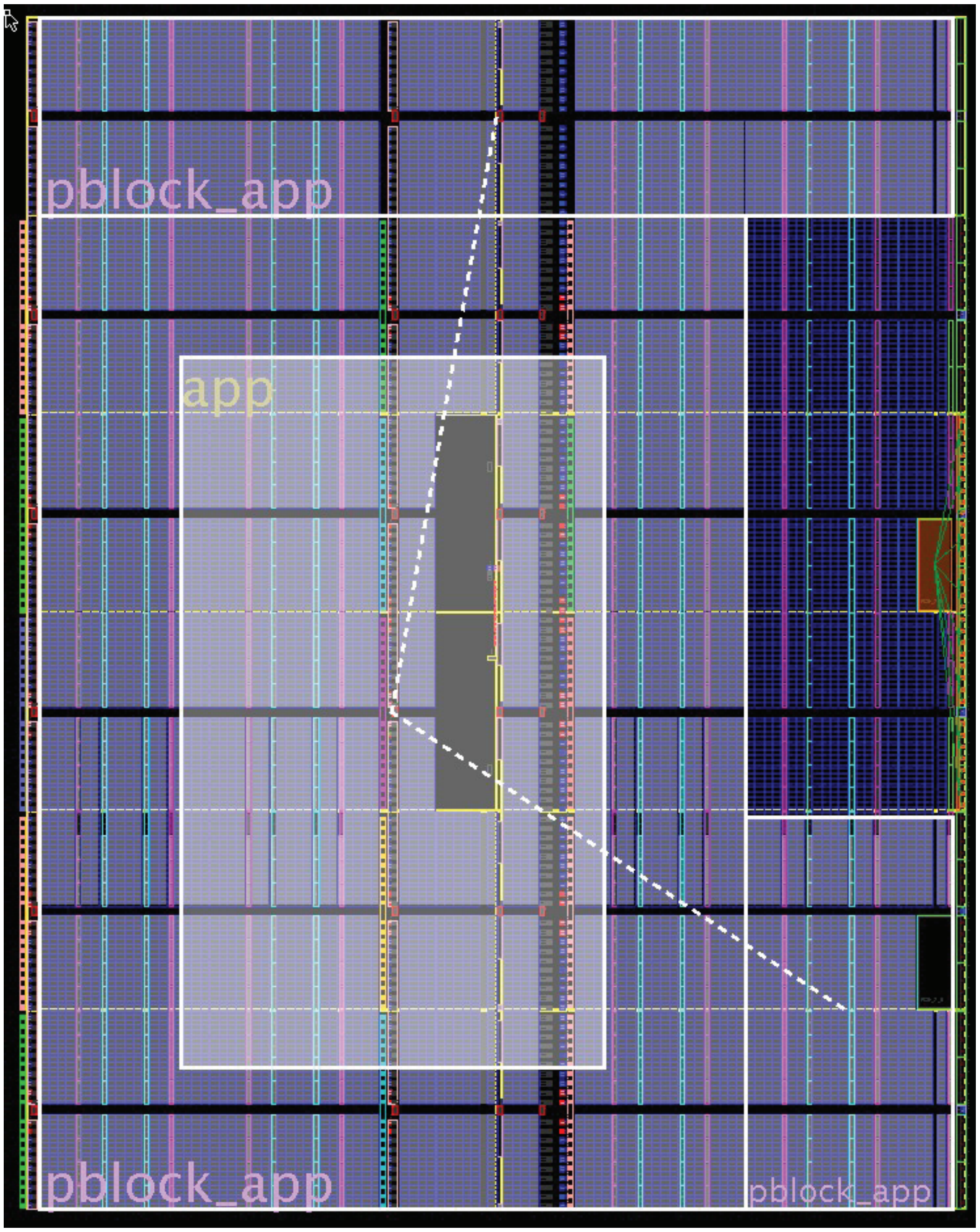


Figure 27: **Floorplan of the Static and Reconfigurable Partitions**

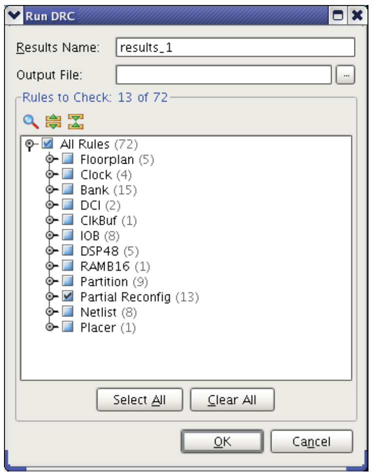
6. Save the floorplan back to the original UCF with **File** → **Export Constraints…** and save to `hw\Source\UCF\XAPP883.ucf`, overwriting the original UCF. After saving, click `XAPP883.ucf` in the Sources frame to view the new constraints:

```
INST "app" AREA_GROUP = "pblock_app";
AREA_GROUP "pblock_app" RANGE=SLICE_X124Y200:SLICE_X161Y239,
SLICE_X124Y0:SLICE_X161Y78, SLICE_X0Y0:SLICE_X123Y239;
AREA_GROUP "pblock_app" RANGE=DSP48_X6Y80:DSP48_X7Y95,
DSP48_X0Y0:DSP48_X7Y29, DSP48_X0Y32:DSP48_X5Y95;
AREA_GROUP "pblock_app" RANGE=RAMB36_X6Y40:RAMB36_X8Y47,
RAMB36_X0Y0:RAMB36_X8Y14, RAMB36_X0Y16:RAMB36_X5Y47;
```

Step 9 — Partial Reconfiguration Design Rule Checks

There are many partial reconfiguration specific design rules that must be followed to implement a valid design. Some of these rules have been incorporated in the design rule checker of the PlanAhead tool. These rules should be run on the partial reconfiguration design before implementing configurations and generating bit files.

1. Select **Tools** → **Run DRC…** or click **Run DRC** on the left side of the GUI under the netlist planner. Mark only the **Partial Reconfig** checkbox and click **OK** (see Figure 28).


Run DRC

Results Name: results_1

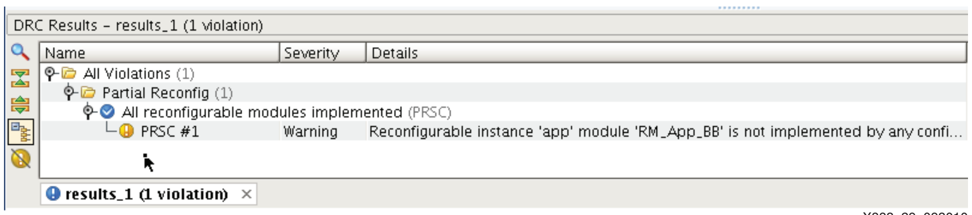
Output File:

Rules to Check: 13 of 72

- All Rules (72)
  - Floorplan (5)
  - Clock (4)
  - Bank (15)
  - DCI (2)
  - ClkBuf (1)
  - IOB (8)
  - DSP48 (5)
  - RAMB16 (1)
  - Partition (9)
  - ✓ Partial Reconfig (13)
  - Netlist (8)
  - Placer (1)

Figure 28: **Run DRC Frame**

The console shows the DRC output. The reference design should pass all DRCs but should display a warning (Figure 29) that the black_box module is not in a configuration. (A reconfigurable module must be in a configuration, because it is the configuration that is ultimately implemented.)


DRC Results - results_1 (1 violation)

Name	Severity	Details
All Violations (1)		
Partial Reconfig (1)		
All reconfigurable modules implemented (PRSC)		
PRSC #1	Warning	Reconfigurable instance 'app' module 'RM_App_BB' is not implemented by any confi…

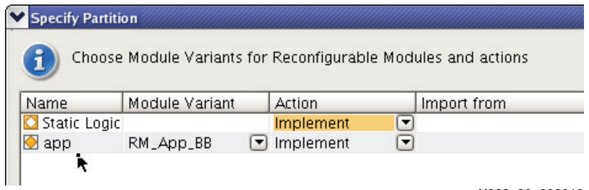
results_1 (1 violation)

Figure 29: **Warning Message in DRC Results**

The warning can be ignored for now because it is resolved before the end of this tutorial.

Step 10 — Creating the Configurations

1. Select **Tools** → **Create Multiple Runs…**
2. Click **Next**, confirm **Part**, and click **Next** again.
3. Name the new run `config_2`. Click the ellipsis (**…**) under Partition Action. Specify the partition to be RM_App_BB, as shown in Figure 30, then click **OK**.



Specify Partition

Choose Module Variants for Reconfigurable Modules and actions

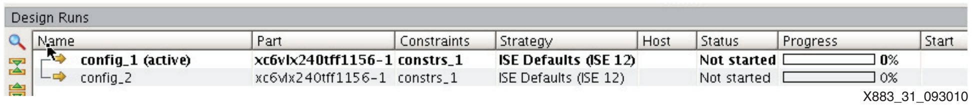
Name	Module Variant	Action	Import from
Static Logic		Implement	
app	RM_App_BB	Implement	

X883_30_093010

Figure 30: **Specify Partition Frame**

4. Click **Next**.
5. Select the radio button **Do not launch now**, and click **Next**.
6. Click **Finish**.

The Design Runs window should now look like Figure 31.



Design Runs

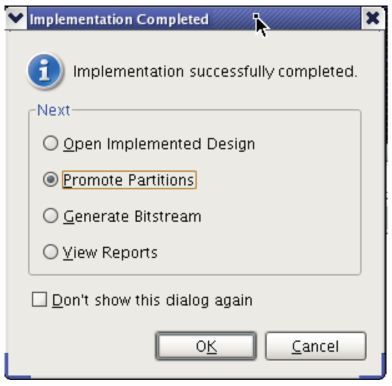
Name	Part	Constraints	Strategy	Host	Status	Progress	Start
config_1 (active)	**xc6vlx240tff1156-1**	**constrs_1**	**ISE Defaults (ISE 12)**		**Not started**	**0%**	
config_2	xc6vlx240tff1156-1	constrs_1	ISE Defaults (ISE 12)		Not started	0%	

X883_31_093010

Figure 31: **Design Runs Frame**

Step 11 — Implementing the Configurations

1. Right-click `config_1` in the Design Runs window and select **Launch Runs…** Click **OK** to accept the launch options and start implementation.
2. After the configuration finishes, review the timing report to check for timing errors.
3. Select **Promote Partitions** and click **OK** in the Implementation Completed window (see Figure 32).



Implementation Completed

Implementation successfully completed.

Next

- ○ Open Implemented Design
- [✓ Promote Partitions]
- ○ Generate Bitstream
- ○ View Reports

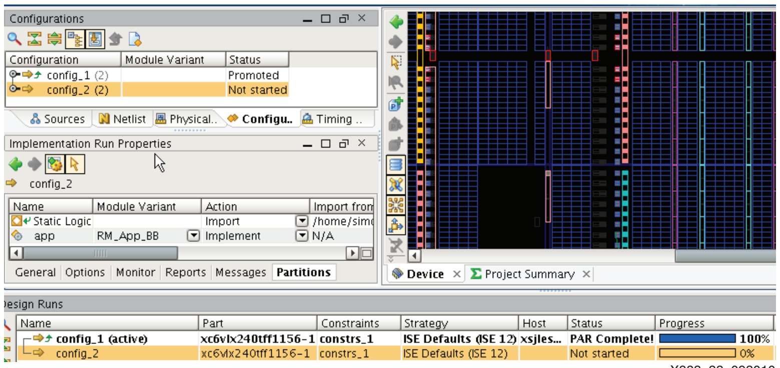
☐ Don't show this dialog again

X883_32_093010

Figure 32: **Implementation Completed Window**

4. Click **Select Implemented** and click **OK** in the Promote Partitions window.

   In the Configurations frame, `config_1` has a status of Promoted and the Static Logic for `config_2` is updated to Not Started (see Figure 33).



Configuration	Module Variant	Status
config_1 (2)		Promoted
config_2 (2)		Not started

config_2

Name	Module Variant	Action	Import from
Static Logic		Import	/home/sim
app	RM_App_BB	Implement	N/A

Name	Part	Constraints	Strategy	Host	Status	Progress
config_1 (active)	xc6vlx240tff1156-1	constrs_1	ISE Defaults (ISE 12)	xsjles...	PAR Complete!	100%
config_2	xc6vlx240tff1156-1	constrs_1	ISE Defaults (ISE 12)		Not started	0%

X883_33_093010

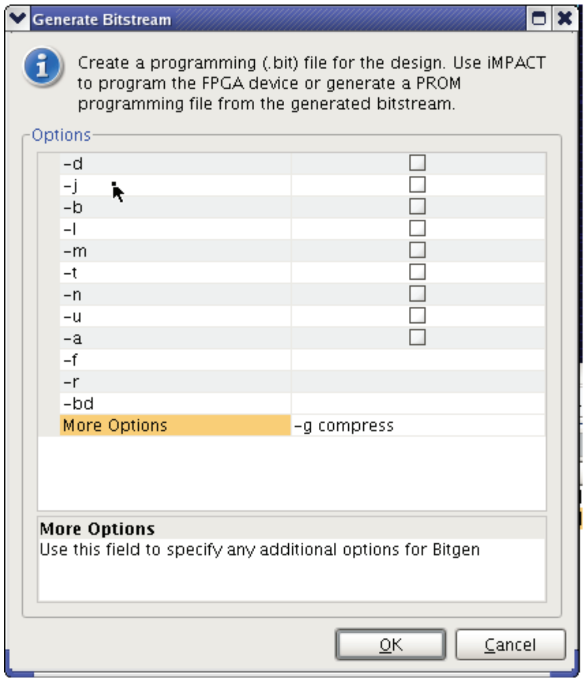
Figure 33: **Configurations and Design Runs Frames**

5. Right-click `config_2` in the Design Runs window and select **Launch Runs…** Click **OK** to start implementation.

Step 12 — Creating the Bitfiles

1. Right-click `config_1` in the Design Runs window and select **Generate Bitstream**. Click **OK** to start creating the bitfile.

   BitGen creates two bitfiles. `Config_1.bit` is the full-configuration bitfile that includes the static and reconfigurable partitions. `Config_1_app_RM_APP_one_partial.bit` contains only the user application implementation and is used in the partial reconfiguration stage.
2. Right-click `config_2` in the Design Runs window and select **Generate Bitstream**.
3. Specify `-g compress` in the More Options field, as shown in Figure 34.



Create a programming (.bit) file for the design. Use iMPACT to program the FPGA device or generate a PROM programming file from the generated bitstream.

-d	☐
-j	☐
-b	☐
-l	☐
-m	☐
-t	☐
-n	☐
-u	☐
-a	☐
-f	
-r	
-bd	
More Options	-g compress

More Options
Use this field to specify any additional options for Bitgen

X883_34_093010

Figure 34: **BitGen Option Selection Window**

BitGen again creates two bitfiles. `Config_2_app_RM_APP_BB_partial.bit` is the black box implementation of the user application plus the PR loader. `Config_2.bit` is the

PR loader implementation. It is used to bring up the PCIe system link and support the subsequent partial reconfiguration.

Note: Use of the compression option is crucial to meet the PCIe system configuration time requirement. It reduces the regular bitfile size of an XC6VLX240T device from 9 MB to about 2 MB. Because of the smaller bitfile size, it is possible to complete the initial configuration in much less time.

4. Click **OK** to start creating bitfile.

Creating a User Project

This reference design can be used as a template to apply to a user design because of the modular nature of the reference design and of the partial reconfiguration flow. The steps to create a user project are:

1. Customize the Integrated Block for PCI Express core settings to suit the user design requirements, especially the BAR, PCIe device ID, and class. In this reference design, BAR2 and BAR0 are the address ranges for the PR loader and the user application, respectively. Refer to Step 2 — Generate the Integrated Block for PCI Express, page 17 in the Building the Project section.
2. Replace the instance `app` in the top-level file `Fast_PCIe_config_top.v` with the user design. Preserve the existing top-level PCIe device connections.
3. Replace all source file paths in `\hw\Synth_XST\UserDesign\UserDesign.prj` with the source files in the user design.
4. Refer to Step 3 — Synthesize the Design, page 22 in the Building the Project section to synthesize the user design.
5. Right-click `app` in the Netlist frame and select **Update Reconfigurable Module** (see Figure 35). Click **Next**.

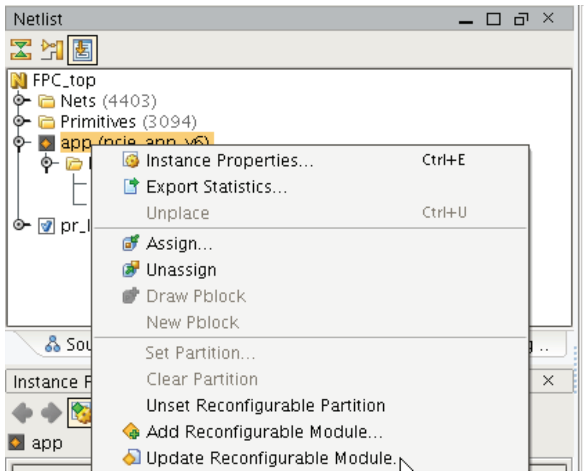
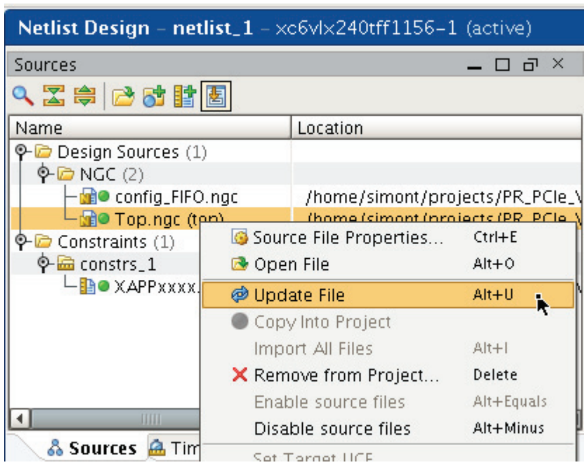


Figure 35: **Update Reconfigurable Module**

6. Set the top netlist file to `hw\Synth_XST\UserDesign\UserDesign.ngc` and click **Next**.
7. Click **Next** again, then click **Finish**.

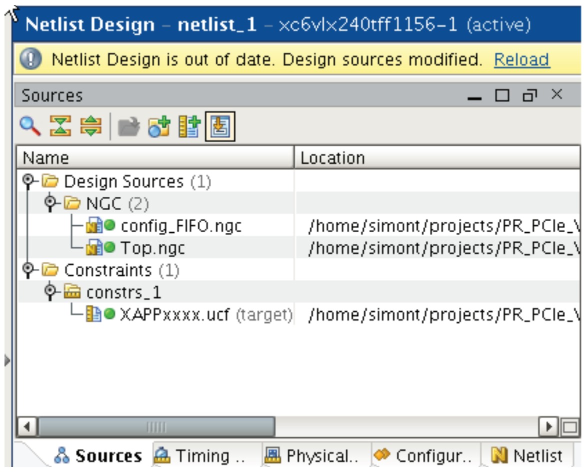
   This updates the user design netlist. The new design takes effect in the next implementation.
8. If the static partition netlist is updated and the netlist is not copied into the project, the PlanAhead tool automatically updates it. Otherwise, right-click the top-level netlist in the Sources frame and select **Update File** to refresh the netlist (see Figure 36).


Netlist Design – netlist_1 – xc6vlx240tff1156-1 (active)
Sources
Name
Location
Design Sources (1)
NGC (2)
config_FIFO.ngc
/home/simont/projects/PR_PCIe_\
Top.ngc (top)
Constraints (1)
constrs_1
XAPPxxxx.
Source File Properties... Ctrl+E
Open File Alt+O
Update File Alt+U
Copy Into Project
Import All Files Alt+I
Remove from Project... Delete
Enable source files Alt+Equals
Disable source files Alt+Minus
Set Target UCF
Sources
Tim

X883_36_093010

Figure 36: **Update the Top-Level Netlist**

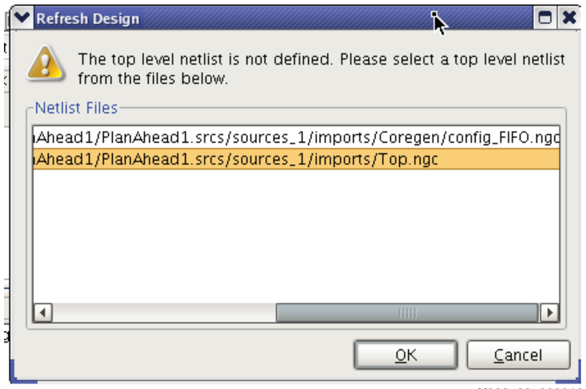
9. Select the new top-level netlist and click **Open**. The PlanAhead software asks the user to reload the netlist. Click **Reload** (see Figure 37).


Netlist Design – netlist_1 – xc6vlx240tff1156-1 (active)
Netlist Design is out of date. Design sources modified. Reload
Sources
Name
Location
Design Sources (1)
NGC (2)
config_FIFO.ngc
/home/simont/projects/PR_PCIe_\
Top.ngc
/home/simont/projects/PR_PCIe_\
Constraints (1)
constrs_1
XAPPxxxx.ucf (target)
/home/simont/projects/PR_PCIe_\
Sources
Timing ..
Physical..
Configur..
Netlist

X883_37_093010

Figure 37: **Reload the Top-Level Netlist**

10. When asked which source netlist is the top level, select the target netlist (see Figure 38). Click **OK**.


Refresh Design
The top level netlist is not defined. Please select a top level netlist from the files below.
Netlist Files
Ahead1/PlanAhead1.srcs/sources_1/imports/Coregen/config_FIFO.ngc
Ahead1/PlanAhead1.srcs/sources_1/imports/Top.ngc
OK
Cancel

X883_38_093010

Figure 38: **Select the Target Netlist**

The user can now start from Step 9 — Partial Reconfiguration Design Rule Checks, page 28 in the Building the Project section and proceed to build the project.

Reference Design Demonstration

The steps in this section guide the user through setting up and running the demonstration. The demonstration can be run without building the reference design first. Pre-built bitfiles are available.

1. Extract the zip file `FPC_release.zip` to the <*ref_design_dir*> directory.

   Note: <*ref_design_dir*> is the project root directory to all directory paths throughout this section.
2. Copy all files in `\sw\Jungo` to <*WindowsXP_install_path*>`\system32`.

   This step installs the Jungo WinDriver libraries and driver (rev. 10.2) in Windows XP. `COMDLD32.OCX` should be backed up if the file already exists.
3. Run `\sw\XAPP1052\win32_application\setup.exe` to install the Bus Master DMA demonstration executable.
4. Connect the ML605 board to an available PCIe device slot in the host system. Connect the ATX power cable to J25. Set switch SW2 to the ON position.

The demonstration first needs to configure the static partition in the FPGA. This initiates the PCIe system link for the host to download partial reconfiguration later. The following steps guide the user through the process of programming the PR loader in the on-board Xilinx Platform Flash:

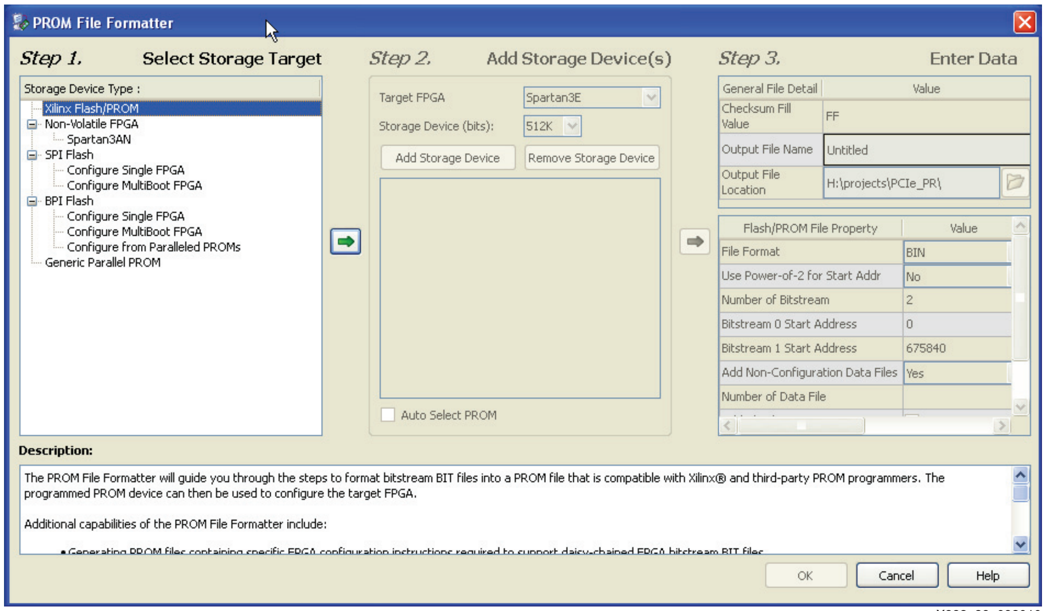
5. Connect the mini USB cable connector to the J22 connector on the ML605 board. Connect the other end to a USB connector to the host system.
6. To enable Platform Flash programming, set DIP switch S2 on the ML605 as follows:

```
S2-1 ON
S2-2 OFF
S2-3 OFF
S2-4 ON
S2-5 ON
S2-6 OFF
```

7. Set DIP switch S1 as follows to avoid loading from the CompactFlash card:

```
S1-1 OFF
S1-2 OFF
S1-3 OFF
S1-4 OFF
```

8. Launch iMPACT and select the **Create PROM File** flow. The PROM File Formatter window appears, as shown in Figure 39.



PROM File Formatter

Step 1. Select Storage Target

Storage Device Type :
- Xilinx Flash/PROM
- Non-Volatile FPGA
  - Spartan3AN
- SPI Flash
  - Configure Single FPGA
  - Configure MultiBoot FPGA
- BPI Flash
  - Configure Single FPGA
  - Configure MultiBoot FPGA
  - Configure from Paralleled PROMs
- Generic Parallel PROM

Step 2. Add Storage Device(s)

Target FPGA: Spartan3E

Storage Device (bits): 512K

Add Storage Device | Remove Storage Device

Auto Select PROM

Step 3. Enter Data

General File Detail	Value
Checksum Fill Value	FF
Output File Name	Untitled
Output File Location	H:\projects\PCIe_PR\

Flash/PROM File Property	Value
File Format	BIN
Use Power-of-2 for Start Addr	No
Number of Bitstream	2
Bitstream 0 Start Address	0
Bitstream 1 Start Address	675840
Add Non-Configuration Data Files	Yes
Number of Data File	

Description:

The PROM File Formatter will guide you through the steps to format bitstream BIT files into a PROM file that is compatible with Xilinx® and third-party PROM programmers. The programmed PROM device can then be used to configure the target FPGA.

Additional capabilities of the PROM File Formatter include:

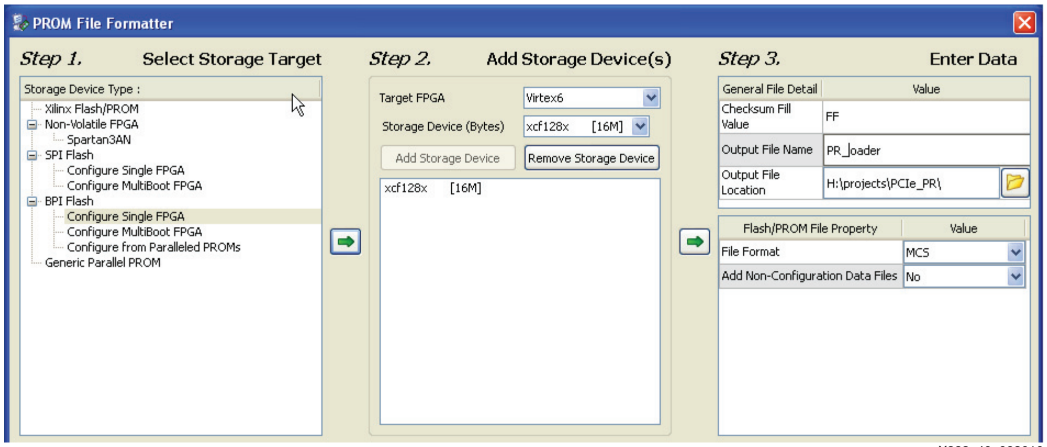
• Generating PROM files containing specific FPGA configuration instructions required to support daisy-chained FPGA bitstream BIT files

OK | Cancel | Help

X883_39_093010

Figure 39: **PROM File Formatter**

9. Select **Configure Single FPGA** under BPI Flash in "Step 1. Select Storage Target." Click the green right arrow and proceed to "Step 2. Add Storage Device(s)."
10. Select **Virtex6** as the Target FPGA. Select **xcf128x [16M]** for Storage Device (Bytes). Click **Add Storage Device**. Click the green right arrow and proceed to "Step 3. Enter Data."
11. Specify `PR_Loader` in the Output File Name field and select a directory in the Output File Location field. Leave the default entries in the other fields. The Formatter should look like Figure 40. Click **OK** to proceed.



PROM File Formatter

Step 1. Select Storage Target

Storage Device Type :
- Xilinx Flash/PROM
- Non-Volatile FPGA
  - Spartan3AN
- SPI Flash
  - Configure Single FPGA
  - Configure MultiBoot FPGA
- BPI Flash
  - Configure Single FPGA
  - Configure MultiBoot FPGA
  - Configure from Paralleled PROMs
- Generic Parallel PROM

Step 2. Add Storage Device(s)

Target FPGA: Virtex6

Storage Device (Bytes): xcf128x [16M]

Add Storage Device | Remove Storage Device

xcf128x [16M]

Step 3. Enter Data

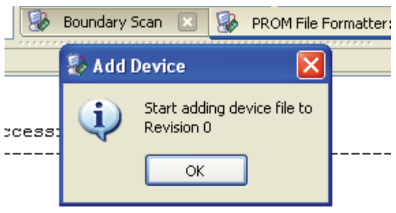
General File Detail	Value
Checksum Fill Value	FF
Output File Name	PR_loader
Output File Location	H:\projects\PCIe_PR\

Flash/PROM File Property	Value
File Format	MCS
Add Non-Configuration Data Files	No

X883_40_093010

Figure 40: **Setting PROM Parameters**

12. The window shown in Figure 41 appears. Click **OK**.



Start adding device file to Revision 0

X883_41_093010

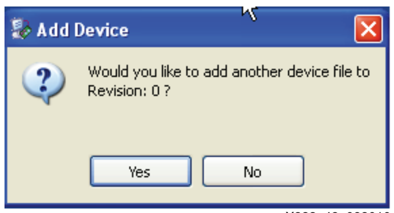
Figure 41: **Add Device Start**

13. Select `\hw\PlanAhead\PlanAhead.runs\config_2\config_2.bit` in the Add Device window. Click **Open**.

    If the project has not been built, the bitfile can be replaced with the pre-built version `\config\bitfiles\PR_Loader.bit`.

    This is the bitfile of the first stage of configuration. It includes the logic (PCIe device link, switcher, and loader) to support partial reconfiguration later. Because it is a compressed bitfile, its size is about 2 MB.

14. Click **No** when asked to add another device file (see Figure 42).

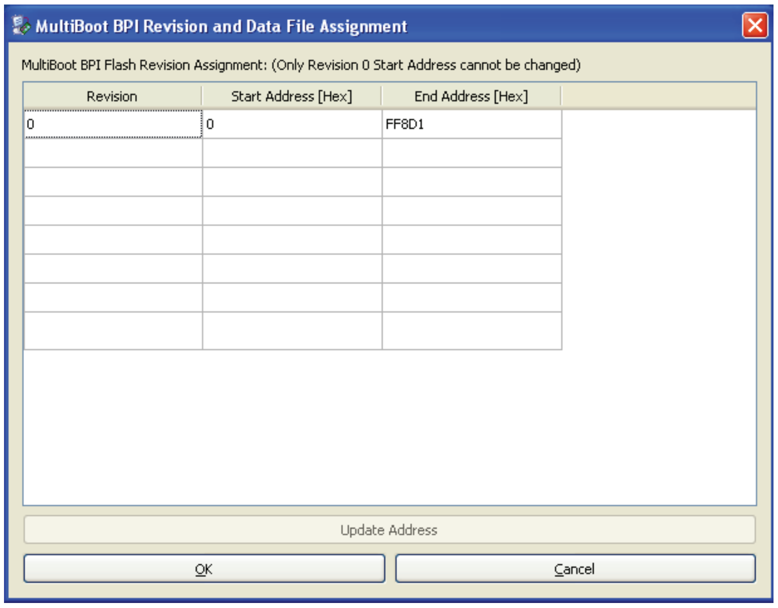


Would you like to add another device file to Revision: 0 ?

X883_42_093010

Figure 42: **Add Another Device**

15. Use the default values in the MultiBoot BPI Revision and Data File Assignment window and click **OK** (see Figure 43).



MultiBoot BPI Flash Revision Assignment: (Only Revision 0 Start Address cannot be changed)

Revision	Start Address [Hex]	End Address [Hex]
0	0	FF8D1

X883_43_093010

Figure 43: **MultiBoot Settings**

16. Click **Generate File…** under iMPACT processes. The PROM file is created after a short moment (see Figure 44).

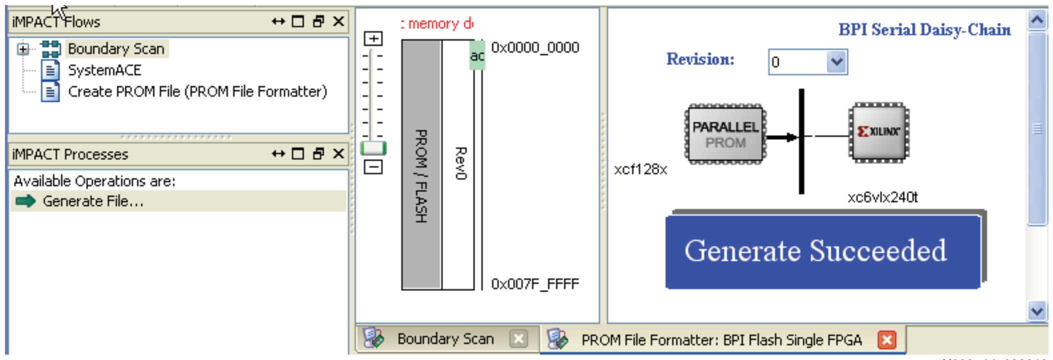


Figure 44: **Successful PROM File Generation**

17. Click on **Boundary Scan** in the iMPACT Flows pane. Right-click the FPGA and select **Add SPI/BPI Flash…** (see Figure 45).

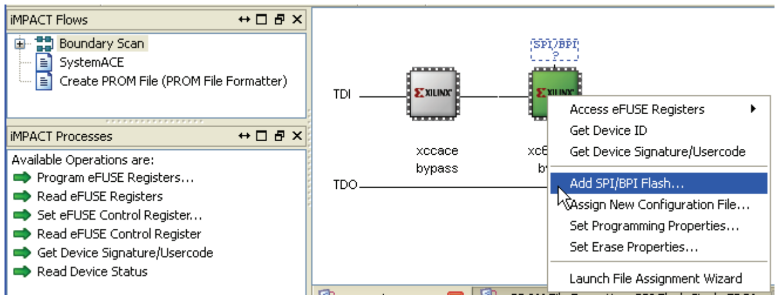
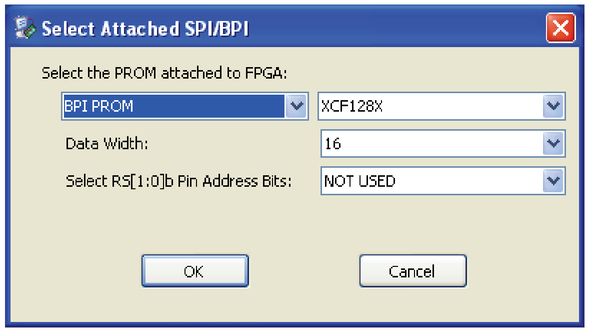

iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Formatter)
iMPACT Processes
Available Operations are:
Program eFUSE Registers…
Read eFUSE Registers
Set eFUSE Control Register…
Read eFUSE Control Register
Get Device Signature/Usercode
Read Device Status
TDI
TDO
xccace
bypass
SPI/BPI
Access eFUSE Registers
Get Device ID
Get Device Signature/Usercode
Add SPI/BPI Flash…
Assign New Configuration File…
Set Programming Properties…
Set Erase Properties…
Launch File Assignment Wizard
X883_45_093010

Figure 45: **Add SPI/BPI Flash**

18. Select `PR_Loader.mcs` from the user-selected directory and click **Open**.

    A pre-built version of this file can be found at `\config\PROM\Fast_PCIe_Loader.mcs`.

19. Use the default values shown in Figure 46 and click **OK**.



Select Attached SPI/BPI

Select the PROM attached to FPGA: BPI PROM, XCF128X

Data Width: 16

Select RS[1:0]b Pin Address Bits: NOT USED

Figure 46: **Select Attached BPI PROM**

20. Right-click the green FLASH icon and select **Program** (see Figure 47).

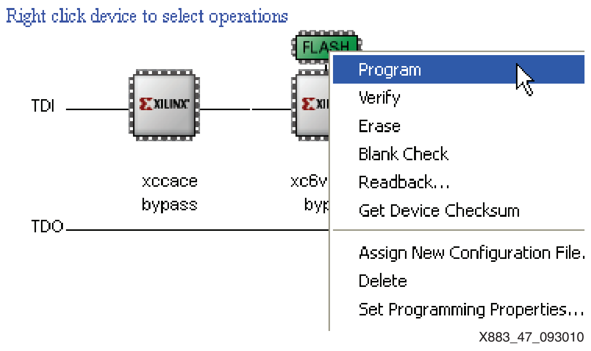
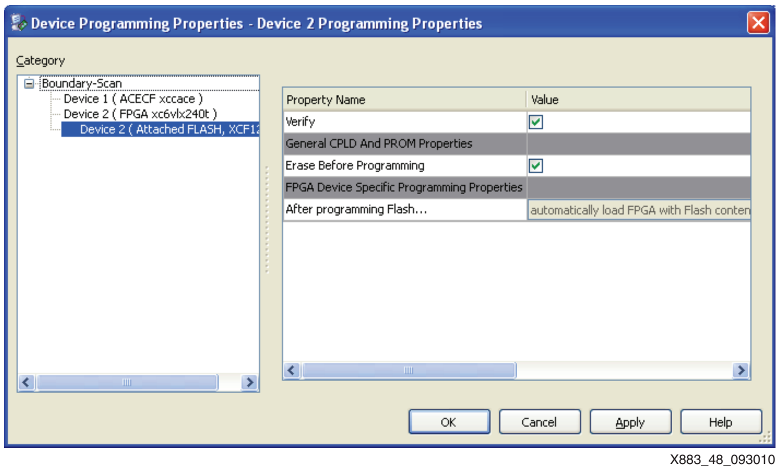


Figure 47: **Program the PROM**

21. Use the default settings and click OK (see Figure 48).


Device Programming Properties - Device 2 Programming Properties
Category
Boundary-Scan
Device 1 (ACECF xccace)
Device 2 (FPGA xc6vlx240t)
Device 2 (Attached FLASH, XCF1
OK
Cancel
Apply
Help
X883_48_093010

Property Name	Value
Verify	✓
General CPLD And PROM Properties	
Erase Before Programming	✓
FPGA Device Specific Programming Properties	
After programming Flash...	automatically load FPGA with Flash conten

Figure 48: **Device 2 Programming Properties**

iMPACT finishes programming the Platform Flash in several minutes.

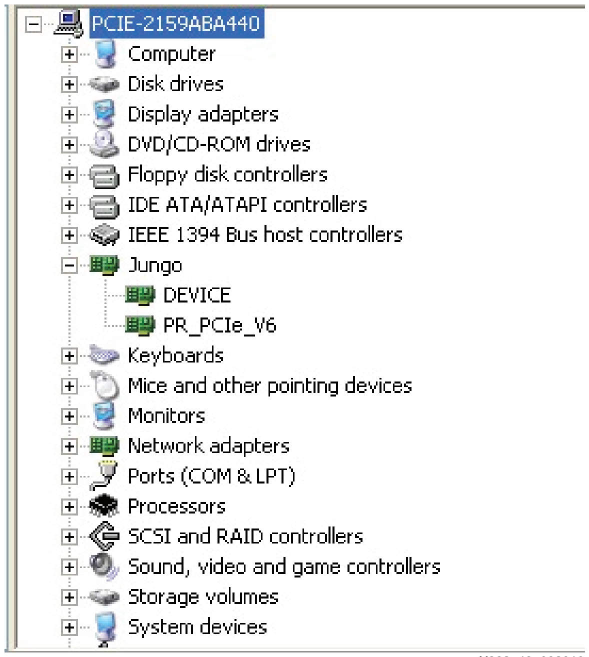
Note: After the Platform Flash has been programmed, the PR loader is automatically configured upon power-up. Because the PR loader bitfile is highly compressed (< 2 MB) and utilizes only a small amount of FPGA resources, its configuration time (from power-on to PCIe device link ready) is typically around 20 ms.

22. Turn off and restart the system. This allows the FPGA to be configured from the Platform Flash, which now contains the first stage PR loader. The MMCM Locked (DS21) and Link Ready (DS22) LEDs are turned on.

23. After Windows starts running, install the PR loader drivers by running the batch file:

    `sw\FPC_Loader\drivers\install_PR_Loader_driver.bat`

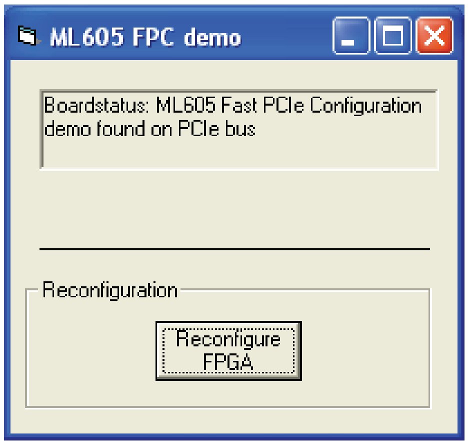
The batch file installs two PR loader drivers, `DEVICE` and `PR_PCIe_V6`. After completion, the two drivers appear under Jungo in the Windows Device Manager, as shown in Figure 49.



X883_49_093010

Figure 49: **Device Manager Listing**

24. Launch the Loader application in `\sw\FPC_Loader\app\FPC_Loader.exe`. The window shown in Figure 50 appears.



Boardstatus: ML605 Fast PCIe Configuration demo found on PCIe bus

X883_50_093010

Figure 50: **PR Loader Application**

Note: The board status indicates that the reference design has been found on the ML605 demonstration board. If the GUI does not find the board, make sure the ML605 board is powered and the drivers were loaded successfully.

25. Click the **Reconfigure FPGA** button. The window shown in Figure 51 appears.

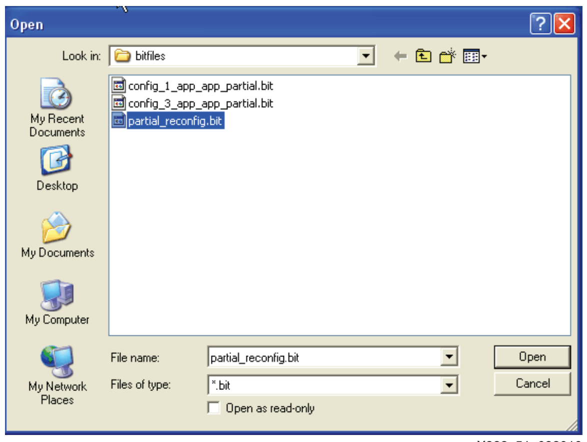


Figure 51: **Select User Application Partial Bitfile**

26. Select `\hw\PlanAhead\PlanAhead.runs\config_1\Config_1_app_RM_APP_one_partial.bit` and click **Open**. This is the configuration file for the user application, which is the BMD reference design from *Bus Master DMA Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions*. [Ref 5]

    If the project has not been built, the bitfile can be replaced with the pre-built version: `\config\bitfiles\Partial_Reconfig.bit`.

    After the last step, the PR Completed LED (DS14) turns on, indicating that the FPGA has been partially reconfigured successfully. At this point, the BMD has control of the PCIe device bus. The PR loader and the application no longer work.

27. Close the demonstration application. Uninstall the PR loader drivers by running the command:

    `\sw\FPC_Loader\drivers\unstall_PR_Loader_driver.bat`

    The results of this operation should look like the window shown in Figure 52.

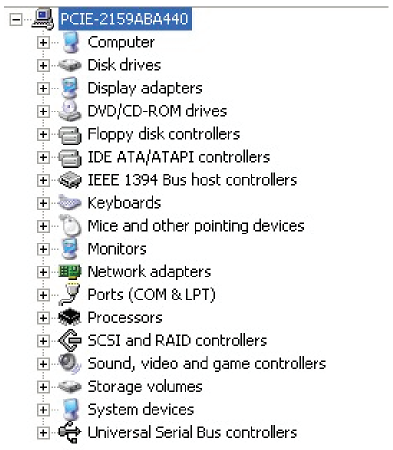
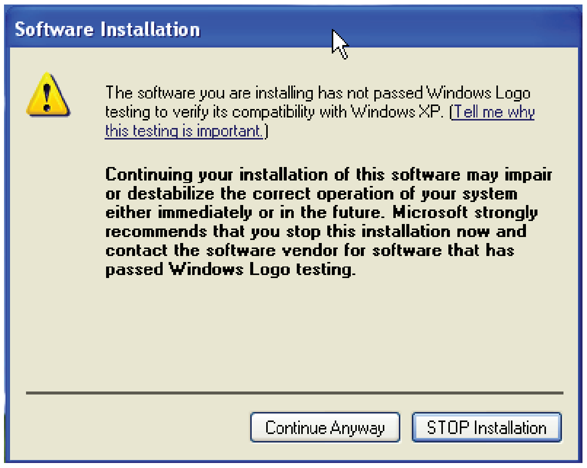


Figure 52: **PR Loader Device Drivers Removed**

28. Install the BMD driver by running the command:

    `sw\XAPP1052\win32_driver\install_BMD_driver.bat`

    On some systems, software and hardware installation warning messages might be seen. Examples are shown in Figure 53 and Figure 54. Click **Continue Anyway** to proceed.

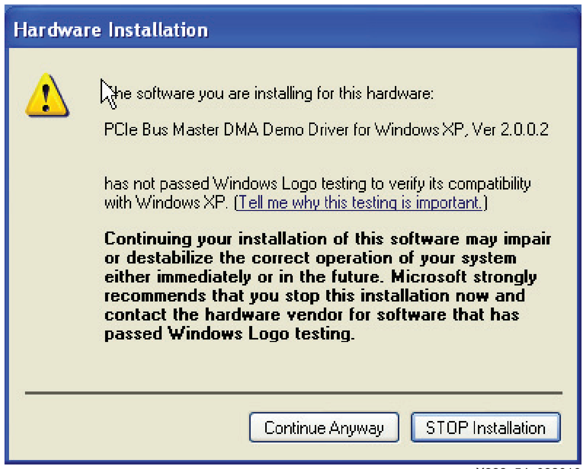


Software Installation

The software you are installing has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the software vendor for software that has passed Windows Logo testing.

Figure 53: **Software Compatibility Warning**



Hardware Installation

The software you are installing for this hardware:

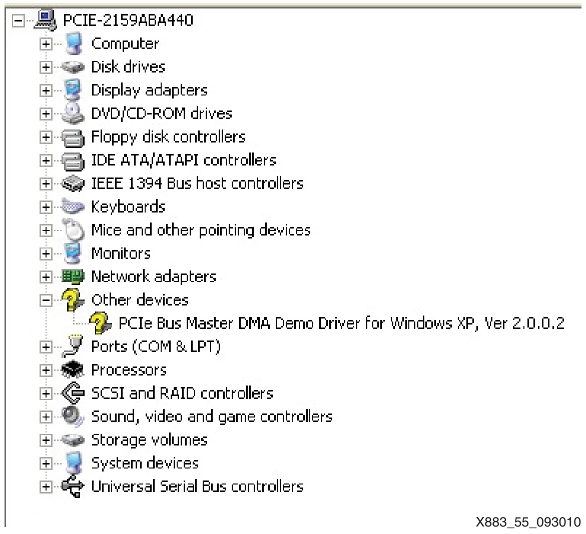
PCIe Bus Master DMA Demo Driver for Windows XP, Ver 2.0.0.2

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

Figure 54: **Hardware Compatibility Warning**

After the installation, the driver `PCIe Bus Master DMA Demo Driver for Windows XP, Ver 2.0.0.2` appears under Other Devices in Device Manager, as shown in Figure 55.


PCIE-2159ABA440
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
Floppy disk controllers
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Other devices
PCIe Bus Master DMA Demo Driver for Windows XP, Ver 2.0.0.2
Ports (COM & LPT)
Processors
SCSI and RAID controllers
Sound, video and game controllers
Storage volumes
System devices
Universal Serial Bus controllers

X883_55_093010

Figure 55: **Bus Master DMA Demo Driver in Windows Device Manager**

29. Launch the **Performance Demo for PCIe** application from the Windows **Start** menu, as shown in Figure 56.

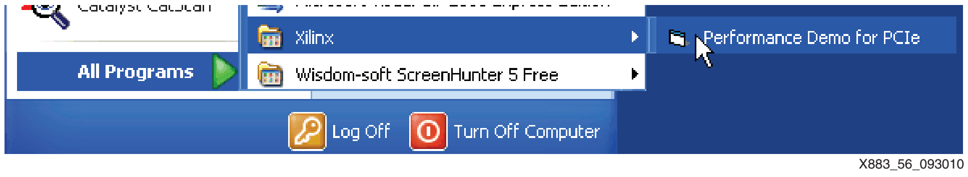
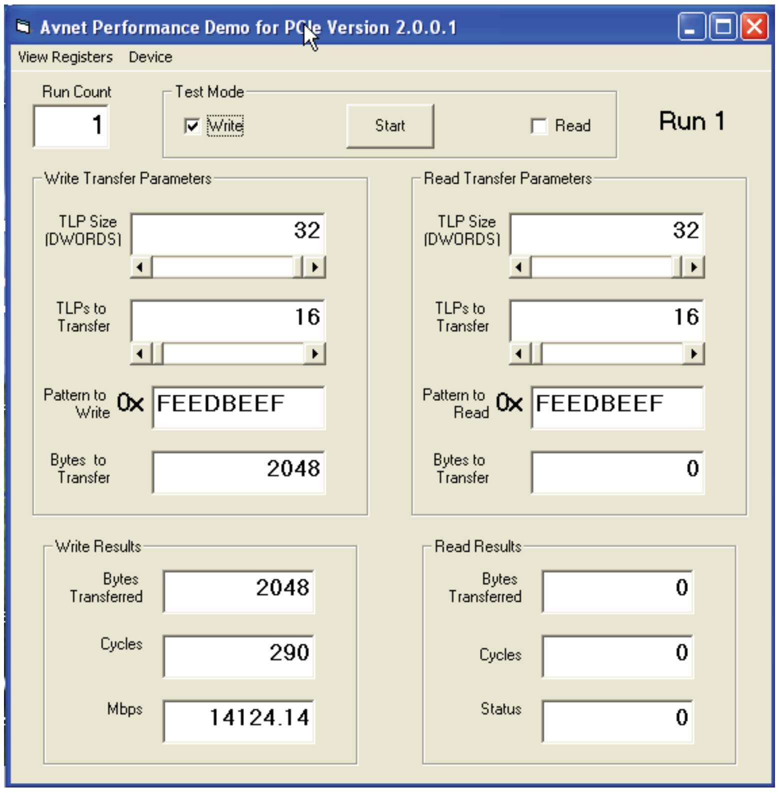


Figure 56: **Launch Performance Demo for PCIe**

30. The application GUI shown in Figure 57 appears.


Avnet Performance Demo for PCIe Version 2.0.0.1

View Registers Device

Run Count: 1

Test Mode: [✓ Write] Start Read

Run 1

Write Transfer Parameters
TLP Size (DWORDS): 32
TLPs to Transfer: 16
Pattern to Write: 0x FEEDBEEF
Bytes to Transfer: 2048

Read Transfer Parameters
TLP Size (DWORDS): 32
TLPs to Transfer: 16
Pattern to Read: 0x FEEDBEEF
Bytes to Transfer: 0

Write Results
Bytes Transferred: 2048
Cycles: 290
Mbps: 14124.14

Read Results
Bytes Transferred: 0
Cycles: 0
Status: 0

X883_57_093010

Figure 57: **BMD Application GUI**

31. Select **Write** or **Read** in Test Mode, and click the **Start** button to initiate the DMA transactions. The performance results appear in the Result boxes. Vary the **TLP Size** and **TLPs to Transfer** settings and observe the changes in performance.

Other Design Considerations

This section describes additional considerations in generating the design:

- The partial reconfiguration flow does not support bidirectional pins to the reconfigurable partition. Any bidirectional functionality must remain in the static partition.
- The ChipScope™ analyzer ILA and ICON cores cannot be instantiated in the reconfigurable partition because they contain global logic. These cores must remain in the static partition.
- The reconfigurable partition should not overlap the MMCM feedback path. The clock feedback path resides in the HCLK region directly adjacent to the MMCM. Overlap could disrupt MMCM operation during reconfiguration.
- This reference design does not support bitstream encryption.
- The reconfiguration data written to the ICAP is loaded in the configuration memory cells directly without error screening (although error screening can be added by the user). Therefore, corrupted data can potentially disrupt the correct operation of the user design or cause contentions. However, the PCIe system has error checking and handling capability built into the packets, guaranteeing packet integrity across the link. This can greatly reduce the potential risks. However, the user should ensure that the reconfiguration data is sent in the correct order from the host application. Any mistakes made prior to writing to the target address range (before packet CRC generation) are *not* detected by the PCIe system error detection mechanism.

- There are two ICAP_VIRTEX6 resources in each device, but only the top ICAP (X0Y1) can support partial reconfiguration. The ISE tool automatically places the ICAP_VIRTEX6 in the correct location.

Reference Design

The reference design files for this application note can be downloaded at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=149163

The reference design matrix is shown in Table 3.

Table 3: **Reference Design Matrix**

Parameter	Description
General	
Developer Name	Xilinx
Target Devices (Stepping Level, ES, Production, Speed Grades)	Virtex-6 FPGA (XC6VLX240T-FF1156-1)
Source Code Provided?	Yes
Source Code Format	Verilog
Design Uses Code or IP from Existing Reference Design, Application Note, 3rd party, or CORE Generator Software?	Yes
Simulation	
Functional Simulation Performed?	Yes
Timing Simulation Performed?	No
Testbench Provided for Functional and Timing Simulations?	Yes
Testbench Format	Verilog
Simulator Software and Version	ModelSim SE 6.4b
SPICE/IBIS Simulations?	No
Implementation	
Synthesis Software Tools and Version	XST 12.2
Implementation Software Tools and Version	ISE Design Suite 12.2
Static Timing Analysis Performed?	Yes
Hardware Verification	
Hardware Verified?	Yes
Hardware Platform Used for Verification	ML605 evaluation board

Resource Utilization

The FPGA resources utilized by the design are summarized in Table 4.

Table 4: **Resources Usage Between Static and Reconfigurable Partitions**

Resources	Static Partition	Reconfigurable Partition (BMD Design)
LUTs	1697	1589
Flip-Flops	1649	1127
Block RAMs	9	0

Software Support

The reference design supports these software and tool environments:

- Verilog HDL source codes
- Functional simulation in ModelSim 6.4b
- XST
- PlanAhead tool v12.1/12.2
- ISE Design Suite 12.1/12.2
- Windows XP Service Pack 3
- Windows .NET Framework 3.0 or above

Tested Systems

The reference design has been tested in these host system environments:

- ASUS p5B-VM
    - CPU: Pentium 4 531 (3.0 GHz)
    - Chipset: Intel P965/G965
    - Southbridge: Intel 828011HB/HR (ICH8/R)
- Intel desktop D975XBX
    - CPU: Pentium D (3.0 GHz)
    - Chipset: Intel 975 Express Chipset
    - Southbridge: Intel 82801GR (ICH7R)

Known Issues

Readback verification after partial reconfiguration shows fewer than 250 mismatches. This is caused by an error in the BitGen mask file and does not impact the intended functionality of the reference design.

The current implementation tools can mistakenly put block RAMs belonging to a static partition in the same configuration frame with block RAMs from a reconfigurable partition. The contents of the block RAMs in the static partition might get corrupted during partial reconfiguration of the affected configuration frame. Therefore, it is recommended to align the partition boundary to the clock region boundary. Because the configuration frame boundary lines up with the clock region boundary, doing so has the effect of preventing block RAMs from the static partition combining with block RAMs from the reconfigurable partition in the same configuration frame.

References

This application note uses these references:

1. UG517, *Virtex-6 FPGA Integrated Block for PCI Express User Guide.*
2. UG702, *Partial Reconfiguration User Guide.*
3. UG360, *Virtex-6 FPGA Configuration User Guide.*
4. DS152, *Virtex-6 FPGA Data Sheet: DC and Switching Characteristics.*
5. XAPP1052, *Bus Master DMA Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions.*

Appendix

The reference design hierarchical structure `[module name/(instance name)]` is shown here:

```
FPC_top
    v6_pcie_v1_6 (core)
        trn_lnk_up_n (trn_lnk_up_n_i)
        trn_lnk_up_n_int (trn_lnk_up_n_int_i)
        trn_reset_n (trn_reset_n_i)
        trn_reset_n_int (trn_reset_n_int_i)
        trn_reset_delay (trn_reset_delay_i)
        pcie_clocking (pcie_clocking_i)
        pcie_2_0_v6 (pcie_2_0_i)
            pcie_gtx_v6 (pcie_gt_i)
                gtx_wrapper_v6 (gtx_v6_i)
                    GTXE1 (GTXD[i])
    Switcher (Switcher_i)
    RM_startup (RM_startup_i)
    PR_Loader (PR_Loader_i)
        PIO (PIO)
            PIO_EP (PIO_EP)
                ICAP_access (ICAP_ACC)
                    ICAP_mod(ICAP)
                        ICAP_VIRTEX6 (ICAP_VIRTEX6_i)
                data_transfer (data_transfer_i)
                    config_FIFO (config_FIFO_i)
                PIO_EP_MEM_ACCESS (EP_MEM)
                PIO_64_RX_ENGINE (EP_RX)
                PIO_64_TX_ENGINE (EP_TX)
            PIO_TO_CTRL (PIO_TO)
    pcie_app_v6 (app)
        BMD (BMD)
            BMD_EP (BMD_EP)
                BMD_EP_MEM_ACCESS(EP_MEM)
                BMD_RX_ENGINE (EP_RX)
                BMD_TX_ENGINE (EP_TX)
                BMD_RD_THROTTLE (RD_THR)
            BMD_TO_CTRL (BMD_TO)
            BMD_CFG_CTRL (BMD_CF)
```

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
11/19/10	1.0	Initial Xilinx release.

Notice of Disclaimer