XAPP493 (v1.0) July 21, 2010

Implementing a DisplayPort Source Policy Maker Using a MicroBlaze Embedded Processor

Author: Tom Strader and Matt Ouellette

Summary

This application note describes the implementation of a DisplayPort™ Source Policy Maker targeted specifically for the Spartan®-6 FPGA Consumer Video Kit (CVK) [Ref 1], as detailed in Table 1. Although the implementation described here targets the Spartan-6 FPGA, it has been designed to be architecture independent.

Table 1: **Target Platforms and Software**

Platform	Tokyo Electron Device (TED) Spartan-6 FPGA Consumer Video Kit (TB-6S-LX150-IMG) with DisplayPort Mezzanine Card
Device	XC6SLX150T-FGG676-3 (Spartan-6 FPGA, XC6SLX150T device, FGG676 package, -3 speed grade)
ISE® Software Version	12.1 (M.53d)
EDK Version	12.1 (EDK_MS1.53d)
DisplayPort LogiCORE™ IP Version	displayport_v1_2

Introduction

The purpose of this reference design is to implement the DisplayPort Source Policy Maker in a MicroBlaze™ processor. The DisplayPort Source Policy Maker communicates to both the transmitter (source) and the receiver (sink) to perform several tasks such as initialization of GTP transceiver links, probing of registers, and other features useful for bring-up and use of the core. In this specific reference system, the DisplayPort Source Policy Maker is implemented at the source side. The mechanism for communication to the sink side is over the auxiliary channel, as described in the DisplayPort standard specification. This standard is available from the Video Electronics Standards Association (VESA) website. [Ref 2]

The reference design included with the application note encompasses the DisplayPort Source Policy Maker, a DisplayPort source core generated using the CORE Generator™ tool, and a video pattern generator to create video data for transmission over the DisplayPort link. The focus of this application note is the DisplayPort Source Policy Maker implementation. The block diagram of the test system is shown in Figure 1.

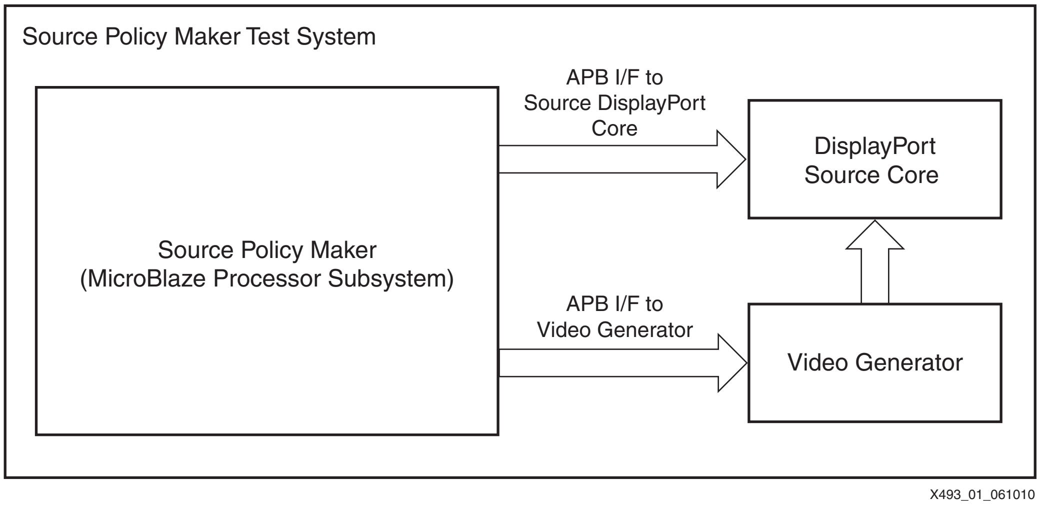


Figure 1: **Top-Level Block Diagram**

Hardware Implementation

The DisplayPort Source Policy Maker is a minimal MicroBlaze processor. The logic for the Source Policy Maker is implemented in stand-alone C code running on the MicroBlaze processor. The block diagram for the MicroBlaze processor system is shown in Figure 2. Included in the design are a MicroBlaze processor, a Processor Local Bus (PLB) v4.6, a Xilinx Platform Studio (XPS) timer, an XPS MicroBlaze Debug Module (MDM) debug core, an XPS UART, and two instantiations of the PLB2APB bridge.

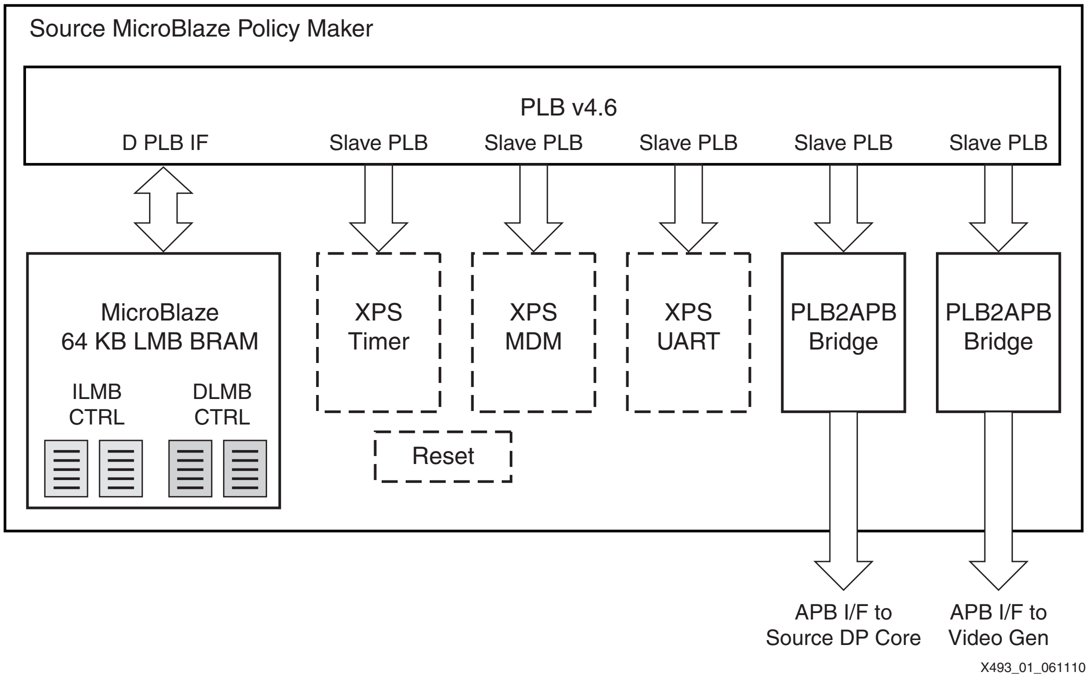


Figure 2: **EDK System Block Diagram**

The communications interface between the MicroBlaze processor and the DisplayPort source core is a simple PLB-Advanced Peripheral Bus (APB) bridge interface, a subset of the ARM® AMBA® 2 specification. [Ref 3] The bridge converts PLB read/write transactions to APB. Other types of APB such as bursting are not supported and are not required in this design.

Software Implementation

The software of the DisplayPort Source Policy Maker is the primary mechanism for controlling the DisplayPort link. The software provides a flexible terminal interface for testing and debugging the link interface. A state diagram of the DisplayPort Source Policy Maker software is shown in Figure 3. The default software included with this application note runs through the link training and hot-plug detection, and provides a UART console interface to provide debug capabilities, such as read/write of DisplayPort AUX Registers and mode selection. The software is stand-alone polled C code running on the MicroBlaze processor.

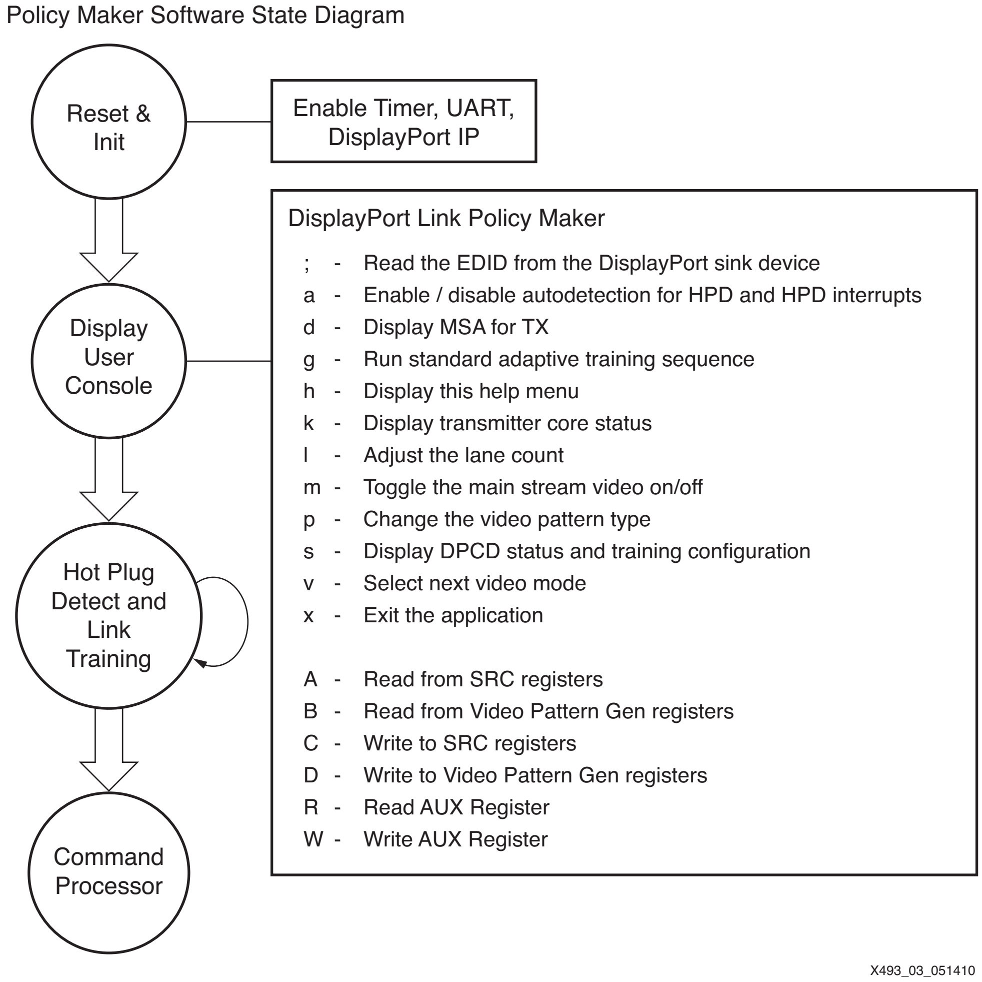

Policy Maker Software State Diagram
Reset & Init
Enable Timer, UART, DisplayPort IP
Display User Console
Hot Plug Detect and Link Training
Command Processor
DisplayPort Link Policy Maker
; - Read the EDID from the DisplayPort sink device
a - Enable / disable autodetection for HPD and HPD interrupts
d - Display MSA for TX
g - Run standard adaptive training sequence
h - Display this help menu
k - Display transmitter core status
l - Adjust the lane count
m - Toggle the main stream video on/off
p - Change the video pattern type
s - Display DPCD status and training configuration
v - Select next video mode
x - Exit the application
A - Read from SRC registers
B - Read from Video Pattern Gen registers
C - Write to SRC registers
D - Write to Video Pattern Gen registers
R - Read AUX Register
W - Write AUX Register
X493_03_051410

Figure 3: **Software Flow Diagram**

Software Flow

The state diagram shown in Figure 3 shows the basic structure of the software. This section describes the startup procedure and terminal options in more detail.

All software for the reference design is targeted at a single DisplayPort core. The address of this DisplayPort core is referenced multiple times throughout the software as:

XILINX_DISPLAYPORT_TX_BASE_ ADDRESS

which is in turn referenced to:

XPAR_PLBV46_TO_APB_BRIDGE_DP_SOURCE_BASEADDR

as defined in `sys_defs.h`.

Reset and Init

When the bitstream is downloaded to the FPGA, the MicroBlaze processor begins executing. The MicroBlaze processor first initializes its peripherals. It runs a self test on the timer and then checks the DisplayPort core in the system to ensure that it is a DisplayPort Source core. The DisplayPort core type is read from address `0xFC` of the DisplayPort core; the value `0xAxxxx` indicates that the core is a source core.

If the DisplayPort core is identified to be a source core, this sequence is executed (`displayport_tx_drv.c`):

1. Disable the transmitter
2. Put the physical layer (PHY) into reset
3. Set the clock divider
4. Set DisplayPort clock speed
5. Bring the PHY out of reset
6. Wait for the PHY to be ready
7. Enable the transmitter
8. Unmask all interrupts

The code then initializes the voltage swing and preemphasis values, but they are not yet transferred to the DisplayPort core. At this point, the init_platform function in `main.c` is complete.

Next, the xilcccAppInit (`xil_ccc_app.c`) function is called from `main.c`. Inside xilcccAppInit, the dplpmInitLinkPolicyMaker (`displayport_lpm.c`) function, which is responsible for setting the hardware capabilities of the transmitter, is called. The parameters passed to the dplpmInitLinkPolicyMaker function are link_rate and lane_count, but many settings are statically assigned within the function. The settings in dplpmInitLinkPolicyMaker control the startup functionality of the transmitter before the terminal is active.

Display User Console

The user console is displayed after the system is initialized (see Table 2); however, the DisplayPort link is not yet active. The hot-plug detect and link training take place after the console is displayed, and the terminal input command processor cannot begin until link training is complete. In other words, if the DisplayPort cable is not plugged in, or if hot-plug detect did not get established, pressing keys in the terminal does not execute commands.

> ***Caution!*** Keys that are pressed before the hot-plug detect is established are stored in the UART FIFO. When the cable is plugged in, the commands execute.

The functionality of each terminal command is described in Command Processor.

Table 2: **Terminal Display after Bitstream Download**

Displayed on Terminal

```
*** Link Policy Maker Active ***
   TX Autodetection  is enabled
   RX Autodetection  is disabled
   MSA Filter Bypass is disabled
*** Default Configuration ***
   Link Rate  : High (2.7Gbps)
   Lane Count : 4

Press 'h' for help

- - - - - - - - - - - - - - - - - - - -
-   Displayport Link Policy Maker   -
- - - - - - - - - - - - - - - - - - - -
; - Read the EDID from the DisplayPort sink device
a - Enable / disable autodetection for HPD and HPD interrupts
d - Display MSA for TX
g - Run standard adaptive training sequence
h - Display this help menu
k - Display transmitter core status
l - Adjust the lane count
m - Toggle the main stream video on/off
p - Change the video pattern type
s - Display DPCD status and training configuration
v - Select next video mode
x - Exit the application
1 - Adjust TX Voltage Swing
2 - Adjust TX Preemphasis

A - Read from SRC registers
B - Read from Video Pattern Gen registers
C - Write to SRC registers
D - Write to Video Pattern Gen registers
R - Read AUX Register
W - Write AUX Register
```

Hot-Plug Detect and Link Training

Hot-plug detect occurs every 100 ms when the user terminal is inactive or after a terminal function completes. The 100 ms time-out for checking hot-plug detect is set in `xil_ccc_app.c` at the xil_getc function call **`app_ctrl →cmd_key = xil_getc(100);`**. To change the time-out value, the time-out parameter passed to xil_getc should be changed. To see this hot-plug detect in action, the DisplayPort cable, which is attached to the CVK board, should be connected and disconnected. See Setup and Usage for details on cable connectivity setup.

Command Processor

The command processor receives the input from the terminal and executes the desired transaction, as described in this section.

; — Read the EDID from the DisplayPort Sink Device

This function reads the extended display identification data (EDID) from the sink device through the AUX channel and displays relevant information. Table 3 shows an example of what is displayed when the **`;`** command is executed.

Table 3: **EDID Status Display**

Displayed on Terminal	Description
`Displayport Configuration Data`	
`DPCD Revision               : 1.1`	The revision number of the DisplayPort sink
`Max Link Rate               : 2.7 Gbps`	The maximum capable link rate of the sink device (2.7 Gb/s or 1.62 Gb/s)
`Max Lane Count              : 4`	
`Enhanced Framing            : Yes`	
`Max Downspread              : 0.5%`	
`Require AUX Handshake       : Yes`	
`Number of RX Ports          : 1`	
`Main Link ANSI 8B/10B       : No`	
`Downstream Port Count       : 0`	
`Format Conversion Support   : No`	
`OUI Support                 : No`	
`Receiver Port 0`	
`Has EDID                    : No`	
`Uses Previous Port          : No`	
`Buffer Size                 : 0`	

a — Enable/Disable Autodetection for HPD and HPD Interrupts

This function enables or disables the ability to autodetect hot-plug detect. By default, autodetection is enabled. This can be seen by connecting and disconnecting the DisplayPort cable while the application is running. As mentioned in Hot-Plug Detect and Link Training, hot-plug detect is checked periodically.

Table 4 is an example of what is displayed when the **a** command is executed. Each time the command is executed, autodetection toggles ON or OFF.

Table 4: **TX Autodetection Status Display**

Displayed on Terminal	Description
`TX Autodetection is disabled`	The code does not dynamically detect when the cable is connected or disconnected

`TX Autodetection is enabled`	The code dynamically detects when the cable is connected or disconnected.
	When the cable is disconnected, this output is displayed:
	`##############################################`
	`######  Detected Dis-Connection Event!  ######`
	`##############################################`
	When the cable is reconnected, this output is displayed:
	`##########################################`
	`######  Detected Connection Event!  ######`
	`##########################################`

d — Display MSA for TX

This command displays the multi-source agreement (MSA) values for the DisplayPort Source LogiCORE IP and the video pattern generator settings, as described in Table 5.

Table 5: **MSA Status Display**

Displayed on Terminal	Description
`Main Stream Attributes TX`	
`Clocks, H Total : 1056`	Total number of clock cycles for Horizontal Front Porch + HSync + Back Porch + Active Video
`Clocks, V Total : 628`	Total number of clock cycles for Vertical Front Porch + HSync + Back Porch + Active Video
`Polarity (V / H) : 0`	Polarity of VSYNC and HSYNC, 1 = High, 0 = Low
`HSync Width : 128`	Number of clock cycles HSYNC is asserted
`VSync Width : 4`	Number of HSYNCS that the VSYNC is asserted
`Horz Resolution : 800`	Active-video horizontal resolution
`Vert Resolution : 600`	Active-video vertical resolution (line count)
`Horz Start : 216`	Number of clocks between start of HSYNC and start of active video. The front porch is determined by this number: Front porch = H_{Total} – Horz Resolution – Horz Start
`Vert Start : 28`	Number of clocks between start of VSYNC and start of active video. The front porch is determined by this number: Front porch = V_{Total} – Vert Resolution – Vert Start
`Misc0 : 0x00000021`	
`Misc1 : 0x00000000`	
`User Pixel Width : 1`	
`M Vid : 40000`	PLL multiplier for stream clock recovery
`N Vid : 270000`	PLL divider for stream clock recovery
`Transfer Unit Size : 64`	TRANSFER_UNIT_SIZE. This sets the size of a transfer unit in the transmitter framing logic. This number should be in the range of 32 to 64 and is set to a fixed value that depends on the inbound video mode.
`User Data Count : 1199`	This register is used to translate the number of pixels per line to the native internal 16-bit datapath. It should be set to (HRES * bits per pixel / 16) – 1.
`Video Pattern Generator Config`	
`VPOL : 1`	VSYNC polarity for the pattern generator
`HPOL : 1`	HSYNC polarity for the pattern generator
`DEPOL : 1`	Data Enable polarity for the pattern generator
`VSWIDTH : 4`	VSYNC width
`VB : 23`	Vertical back porch
`VF : 1`	Vertical front porch
`VRES : 600`	Vertical resolution
`HSWIDTH : 128`	HSYNC width
`HB : 88`	Horizontal back porch
`HF : 40`	Horizontal front porch
`HRES : 800`	Horizontal resolution

g — Run Standard Adaptive Training Sequence

This command runs a full training sequence on the link, including hot-plug detect, link setup, and link training.

h — Display This Help Menu

This command displays the help menu, as shown in Table 6.

Table 6: **Help Menu Display**

Displayed on Terminal

```
- - - - - - - - - - - - - - - - - - -
-   Displayport Link Policy Maker   -
- - - - - - - - - - - - - - - - - - -
; - Read the EDID from the DisplayPort sink device
a - Enable / disable autodetection for HPD and HPD interrupts
d - Display MSA for TX
g - Run standard adaptive training sequence
h - Display this help menu
k - Display transmitter core status
l - Adjust the lane count
m - Toggle the main stream video on/off
p - Change the video pattern type
s - Display DPCD status and training configuration
v - Select next video mode
x - Exit the application
1 - Adjust TX Voltage Swing
2 - Adjust TX Preemphasis

A - Read from SRC registers
B - Read from Video Pattern Gen registers
C - Write to SRC registers
D - Write to Video Pattern Gen registers
R - Read AUX Register
W - Write AUX Register
```

k — Display Transmitter Core Status

The transmitter core status shows information about the DisplayPort transmitter LogiCORE IP. These registers are described in Table 7. More information can be found in the *LogiCORE IP DisplayPort Source Core v1.2 User Guide*. [Ref 4]

Table 7: **Transmitter Core Status Display**

Displayed on Terminal	Description
`DisplayPort Transmitter Registers:`	
`TX Core ID                  : 0x000A0101`	Bits 19:16 = Core type `0xA` = DisplayPort source (TX) core Bits 15:0 = Core revision
`Link BW                     : 0x0000000A`	Bits 3:0 = Link bandwidth `0x6` = 1.62 Gb/s `0xA` = 2.7 Gb/s
`Lane Count                  : 0x00000004`	Bits 3:0 = Number of lanes `0x1` = 1 Lane `0x2` = 2 Lanes `0x4` = 4 Lanes
`Enhanced Framing            : 0x00000001`	Bit 1 = enhanced framing enabled `0` = Disabled `1` = Enabled

Table 7: **Transmitter Core Status Display *(Cont'd)***

Displayed on Terminal	Description
`Training Pattern : 0x00000000`	
`Link Qual Pattern : 0x00000000`	
`Scrambling Disable : 0x00000000`	
`Downspread Ctrl : 0x00000000`	
`TX Enable : 0x00000001`	
`Main Link Enable : 0x00000001`	Bit 1 = Main link enabled `0` = Disabled `1` = Enabled
`Sec Stream Enable : 0x00000000`	Bit 1 = Secondary stream enabled `0` = Disabled `1` = Enabled
`TX Command Reg : 0x00000907`	
`TX Address Reg : 0x00000200`	
`Interrupt Sig : 0x00000005`	
`Interrupt : 0x00000000`	
`Interrupt Mask : 0x00000000`	
`TX Clk Divider : 0x00000028`	
`TX Status : 0x00000005`	
`TX Reply Reg : 0x00000000`	
`TX Reply Count : 0x00000059`	
`PHY Registers:`	
`Reset : 0x00000000`	
`Preemphasis, Lane 0 : 0x00000003`	Bits 2:0 = Preemphasis `0x01` = 0 dB `0x02` = 3.5 dB `0x03` = 6 dB `0x04` = 9.5 dB
`Preemphasis, Lane 1 : 0x00000003`	Same as preemphasis lane 0
`Preemphasis, Lane 2 : 0x00000003`	Same as preemphasis lane 0
`Preemphasis, Lane 3 : 0x00000003`	Same as preemphasis lane 0
`Voltage Diff, Lane 0 : 0x00000006`	Bits 2:0 = Voltage swing `0x6` = 1200 mV `0x5` = 800 mV `0x4` = 600 mV `0x0` = 400 mV
`Voltage Diff, Lane 1 : 0x00000006`	Same as voltage swing lane 0
`Voltage Diff, Lane 2 : 0x00000006`	Same as voltage swing lane 0
`Voltage Diff, Lane 3 : 0x00000006`	Same as voltage swing lane 0
`Transmit PRBS7 : 0x00000000`	

Table 7: **Transmitter Core Status Display *(Cont'd)***

Displayed on Terminal	Description
`PHY Clock Feedback Set      : 0x00000003`	
`PHY Status                  : 0x1111007F`	

l — Adjust the Lane Count

This command toggles the transmitter between 1, 2, or 4 lanes each time `l` (lower-case L) is pressed. The possible outputs are shown in Table 8.

Table 8: **- Lane Count Status Display**

Displayed on Terminal	Description

`Lane count set to 1`
`Lane count set to 2`
`Lane count set to 4`	

m — Toggle the Main Stream Video ON / OFF

This command toggles the main link on or off each time **`m`** is pressed.

p — Change the Video Pattern Type

This command changes the type of color bars displayed on the screen. Each time the **`p`** command is executed, the pattern type increments. There are eight video pattern types, but some appear identical on the monitor.

s — Display DPCD Status and Training Configuration

This command displays the DisplayPort configuration data of the DisplayPort sink as described in the DisplayPort specification. [Ref 2]

Table 9: **DPCD Status Display**

Displayed on Terminal	Description
`Lane 0/1 Status              = 0x77`	LANE0_1_STATUS: Lane0 and Lane1 status Bit 0 = LANE0_CR_DONE Bit 1 = LANE0_CHANNEL_EQ_DONE Bit 2 = LANE0_SYMBOL_LOCKED Bit 3 = RESERVED. Read 0. Bit 4 = LANE1_CR_DONE Bit 5 = LANE1_CHANNEL_EQ_DONE Bit 6 = LANE1_SYMBOL_LOCKED Bit 7 = RESERVED. Read 0.
`Lane 2/3 Status               = 0x77`	LANE2_3_STATUS (Bit definitions identical to that of LANE0_1_STATUS)
`Lane Align Status             = 0x81`	LANE_ALIGN__STATUS_UPDATED Bit 0 = INTERLANE_ALIGN_DONE Bits 5:1 = RESERVED. Read all 0s. Bit 6 = DOWNSTREAM_PORT_STATUS_CHANGED Bit 6 is set when any of the downstream ports change status. Bit 7 = LINK_STATUS_UPDATED Link Status and Adjust Request are updated after the last read. Bit 7 is set when updated and cleared after read.

Table 9: **DPCD Status Display *(Cont'd)***

Displayed on Terminal	Description
`Sink Status                  = 0x00`	SINK_STATUS Bit 0 = RECEIVE_PORT_0_STATUS 0 = SINK out of synchronization 1 = SINK in synchronization Bit 1 = RECEIVE_PORT_1_STATUS 0 = SINK out of synchronization 1 = SINK in synchronization These status bits are set only when the sink device determines that the received streams are properly regenerated and within the supported stream format range. Bits 7:2 = RESERVED. Read all 0s
`Adjustment Request 0/1       = 0x88`	ADJUST_REQUEST_LANE0_1: Voltage swing and equalization setting adjust Request for Lane0 and Lane1 Bits 1:0 = VOLTAGE_SWING_LANE0 00 = Level 0 01 = Level 1 10 = Level 2 11 = Level 3 Bits 3:2 = PRE-EMPHASIS_LANE0 00 = Level 0 01 = Level 1 10 = Level 2 11 = Level 3 Bits 5:4 = VOLTAGE_SWING_LANE1 00 = Level 0 01 = Level 1 10 = Level 2 11 = Level 3 Bits 7:6 = PRE-EMPHASIS_LANE1 00 = Level 0 01 = Level 1 10 = Level 2 11 = Level 3
`Adjustment Request 2/3       = 0x88`	ADJUST_REQUEST_LANE2_3 (Bit definitions identical to that of ADJUST_REQUEST_LANE0_1)
`Training Config:`	
`(0x0100) Link Bandwidth Setup  : 0x0A`	LINK_BW_SET: Main link bandwidth setting = value x 0.27 Gb/s per lane Bits 7:0 = LINK_BW_SET For DisplayPort version 1, revision 1a, only two values are supported. All other values are reserved. 0x06 = 1.62 Gb/s per lane 0x0A = 2.7 Gb/s per lane The source can choose either of the two link bandwidths as long as it does not exceed the capability of the DisplayPort receiver, as indicated in the receiver capability field.

Table 9: **DPCD Status Display *(Cont'd)***

Displayed on Terminal	Description
`(0x0101) Lane Count Set        : 0x84`	LANE_COUNT_SET: Main link lane count = value Bits 4:0 = LANE_COUNT_SET For DisplayPort version 1, revision 1a, all values are reserved, except for: 0x1 = One lane 0x2 = Two lanes 0x4 = Four lanes For one-lane configuration, Lane0 is used. For two-lane configuration, Lane0 and Lane1 are used. The source can choose any lane count as long as it does not exceed the capability of the DisplayPort receiver, as indicated in the receiver capability field. For DPCD Ver.1.0: Bits 7:5 = RESERVED. Read all 0s. For DPCD Ver.1.1: Bits 6:5 = RESERVED. Read all 0s. Bit 7 = ENHANCED_FRAME_EN 0 = Enhanced framing symbol sequence is not enabled. 1 = Enhanced framing symbol sequence for BS, SR, CPBS, and CPSR is enabled.

`(0x0102) Training Pattern Set  : 0x00`	TRAINING_PATTERN_SET
	Bits 1:0 = TRAINING_PATTERN_SET: Link training pattern setting
	00 =Training not in progress (or disabled)
	01 =Training Pattern 1
	10 =Training Pattern 2
	11 = RESERVED
	Bits 3:2 = LINK_QUAL_PATTERN_SET
	00 = Link quality test pattern not transmitted
	01 = D10.2 test pattern (unscrambled) transmitted (same as Training Pattern 1)
	10 = Symbol Error Rate measurement pattern transmitted
	11 = PRBS7 transmitted
	The PRBS7 bit sequence must be:
	`----- direction -----`
	`0010000011000010100`
	`011110010001011001110101001`
	`111101000011100010010011011`
	`010110111101100011010010111`
	`011100110010101011111110000`
	The upper left is transmitted first and lower right is transmitted last.
	Bit 4 = RECOVERED_CLOCK_OUT_EN
	0 = Recovered clock output from a test pad of DisplayPort Rx not enabled
	1 = Recovered clock output from a test pad of DisplayPort Rx enabled.
	Bit 5 = SCRAMBLING_DISABLE
	0 = DisplayPort transmitter scrambles data symbols before transmission
	1 = DisplayPort transmitter disables scrambler and transmits all symbols without scrambling
	For DPCD Version 1.0:
	Bits 7:6 = Reserved, read all 0s.
	For DPCD Version 1.1
	Bits 7:6 = SYMBOL ERROR COUNT SEL
	00 = Disparity error and illegal symbol error
	01 = Disparity error
	10 = Illegal symbol error
	11 = Reserved

Table 9: **DPCD Status Display *(Cont'd)***

Displayed on Terminal	Description
`(0x0103) Training Lane 0 Set   : 0x10`	TRAINING_LANE0_SET: Link Training Control_Lane0 Bits 1:0 = VOLTAGE SWING SET `00` = Training Pattern 1 with voltage swing level 0 `01` = Training Pattern 1 with voltage swing level 1 `10` = Training Pattern 1 with voltage swing level 2 `11` = Training Pattern 1 with voltage swing level 3 Bit 2 = MAX_SWING_REACHED Set to `1` when the maximum driven current setting is reached. The transmitter must support at least three levels of voltage swing, 400, 600, and 800 mV_diff_pp. If only three levels of voltage swing are supported, program bit 2 must be set to `1` when bits 1:0 are set to `10`. Bits 4:3 = PRE-EMPHASIS_SET `00` = Training Pattern 2 without pre-emphasis `01` = Training Pattern 2 with pre-emphasis level 1 `10` = Training Pattern 2 with pre-emphasis level 2 `11` = Training Pattern 2 with pre-emphasis level 3 Bit 5 = MAX_PRE-EMPHASIS_REACHED Set to `1` when the maximum drive current setting is reached. The transmitter must support at least two levels of pre-emphasis (3.5 dB and 6 dB) in addition to no pre-emphasis (0 dB). Support of additional pre-emphasis levels is optional. If only 0 dB, 3.5 dB, and 6 dB are supported, the transmitter must set bit 5 when it sets bits 4:3 to `0x2` (level2) to indicate to the receiver that the maximum preemphasis level has been reached. Support of independent preemphasis level control for each lane is also optional. Bits 7:6 = RESERVED. Read all 0s.
`(0x0104) Training Lane 1 Set   : 0x10`	TRAINING_LANE1_SET (Bit definition identical to that of TRAINING_LANE0_SET.)
`(0x0105) Training Lane 2 Set   : 0x10`	TRAINING_LANE2_SET (Bit definition identical to that of TRAINING_LANE0_SET.)
`(0x0106) Training Lane 3 Set   : 0x10`	TRAINING_LANE3_SET (Bit definition identical to that of TRAINING_LANE0_SET.)
`(0x0107) Downspread Ctrl       : 0x00`	DOWNSPREAD_CTRL: Down-spreading control Bits 3:0 = RESERVED. Read all 0s Bit 4 = SPREAD_AMP Spreading amplitude: `0` = No downspread `1` = Equal to or less than 0.5% downspread Bits 7:5 = RESERVED. Read all 0s. Write `0x00` to declare to the receiver that there is no down-spreading. The modulation frequency must be in the range of 30 kHz – 33 kHz.

v — Select Next Video Mode

This command initializes the video pattern generator and toggles the video mode between 800 x 600 and 1024 x 768 each time it is executed, as shown in Table 10.

Table 10: **Video Mode Status Display**

Displayed on Terminal	Description
Setting mode to VIDEO_MODE_800_600_60_32	Sets the video mode to 800 x 600, 60 Hz, 32-bit color. (The pattern generator only transmits 24-bit color.)
Setting mode to VIDEO_MODE_1024_768_60_32	Sets the video mode to 1024 x 768, 60 Hz, 32-bit color. (The pattern generator only transmits 24-bit color.)

x — Exit the Application

This function exits the application loop and returns to main. The processor remains in an infinite loop in main and does nothing more at this point.

A — Read from SRC Registers

This command allows direct read access to the registers inside the DisplayPort Source LogiCORE IP. A description of each register can be found in Chapter 5 of the *LogiCORE IP DisplayPort Source Core v1.2 User Guide*. [Ref 4]

B — Read from Video Pattern Gen Registers

This command allows direct read access to the registers inside the video pattern generator core. The registers are described inTable 11.

C — Write to SRC Registers

This command allows direct write access to the registers inside the DisplayPort Source LogiCORE IP. A description of each register can be found in Chapter 5 of the *LogiCORE IP DisplayPort Source Core v1.2 User Guide*.

D — Write to Video Pattern Gen Registers

This command allows direct write access to the registers inside the video pattern generator core. The registers are described in Table 11.

Table 11: **Video Pattern Generator Registers**

Register Address	Read/Write	Description
`0x000`	R/W	Bit 0 = Enable video output Bit 1 = SW reset of the pattern generator
`0x004`	R/W	Bit 0 = VSYNC polarity
`0x008`	R/W	Bit 0 = HSYNC polarity
`0x00C`	R/W	Bit 0 = DE polarity
`0x010`	R/W	Bits 8:0 = VSYNC width
`0x014`	R/W	Bits 8:0 = Vertical back porch
`0x018`	R/W	Bits 8:0 = Vertical front porch
`0x01C`	R/W	Bits 10:0 = Vertical resolution
`0x020`	R/W	Bits 8:0 = HSYNC width
`0x024`	R/W	Bits 8:0 = Horizontal back porch
`0x028`	R/W	Bits 8:0 = Horizontal front porch

Table 11: **Video Pattern Generator Registers *(Cont'd)***

Register Address	Read/Write	Description
0x02C	R/W	Bits 10:0 = Horizontal resolution
0x034	R/W	Bits 21:0 = Framelock delay Bit 31 = Framelock enable
0x03C	R/W	Bits 10:0 = Framelock line fraction Bit 16 = Framelock align HSYNC
0x040	R/W	Bit 0 = Bar mode 0 = Bars run full screen height 1 = Bars run 3/4 screen height with additional bars on top Bit 1 = HD mode 0 = 720 1 = 1080 Bit 2 = Interlace 0 = Progressive 1 = Interlaced
0x100	R/W	Bit 0 = Video clock select 0 = 40 MHz 1 = 65 MHz
0x200	R	Bits 11:0 = VSYNC counter current count
0x204	R	Bits 11:0 = HSYNC counter current count
0x208	R	Bits 11:0 = Data enable counter current count

R — Read AUX Register

This command allows read access to the DisplayPort configuration data register space of the DisplayPort sink over the AUX channel. Detailed descriptions of the registers can be found in the DisplayPort specification [Ref 2] in the “Address Mapping for the DPCD (DisplayPort Configuration Data)” table.

W — Write AUX Register

This command allows write access to the DisplayPort configuration data register space of the DisplayPort sink over the AUX channel. Detailed descriptions of the registers can be found in the DisplayPort specification in the “Address Mapping for the DPCD (DisplayPort Configuration Data)” table.

Software Source Files

Table 12 lists all the included software source files for the DisplayPort Source Policy Maker design.

Table 12: **Description of Software Source Files**

File Name	Description
main.c	Main routine
displayport_lpm.c	Link policy maker functions for link management
xil_displayport.c	Low-level DisplayPort physical layer communication and training functions
displayport_tx_drv.c	Driver layer for TX (source) side of a DisplayPort link for accessing registers in the core
xil_ccc_app.c	User interface terminal and continuous link status code

Table 12: **Description of Software Source Files *(Cont'd)***

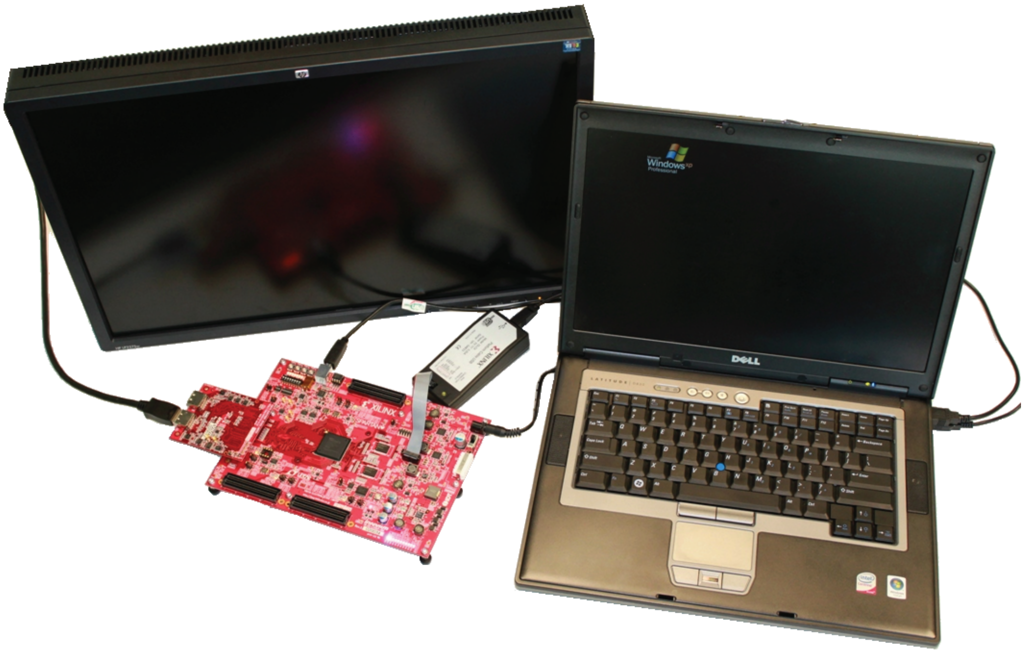
File Name	Description
`displayport_defs.h`	Various definitions for core status and AUX channel communication
`edid_drv.h`	EDID structures for managing the EDID
`mode_table.h`	Defines the display characteristics such as resolution and frame rate
`sys_defs.h`	Generic constant and structure definitions
`xlib_string.c/.h`	String functions for terminal handling. Also includes a DEBUG_LEVEL define for terminal verbosity

Setup and Usage

This reference design is targeted at the Spartan-6 FPGA Consumer Video Kit. [Ref 1] To bring up this reference design, the following setup is needed:

- PC with at least two USB 1.1 or USB 2.0 ports with ISE Design Suite 12.1 and EDK 12.1 installed.
- Spartan-6 FPGA CVK board and power supply.
- DisplayPort-compliant receiver device, such as a monitor. For this example, an HP LP2275w is used.
- Platform cable USB JTAG programmer.
- DisplayPort cable and two USB cables.

The setup is shown in Figure 4.



X493_04_051510

Figure 4: **Hardware Setup**

All necessary cables and jumpers should be set correctly on the board, as shown in Table 13 and Figure 5.

Table 13: **CVK Jumper Settings**

Jumper Name	Jumper Position
JP2	1-2
JP3	2-3
JP4	1-2
JP5	1-2
JP6	1-2
JP7	1-2
JP8	1-2
JP9	1-2
JP10	1-2
JP11	1-2
JP12	1-2

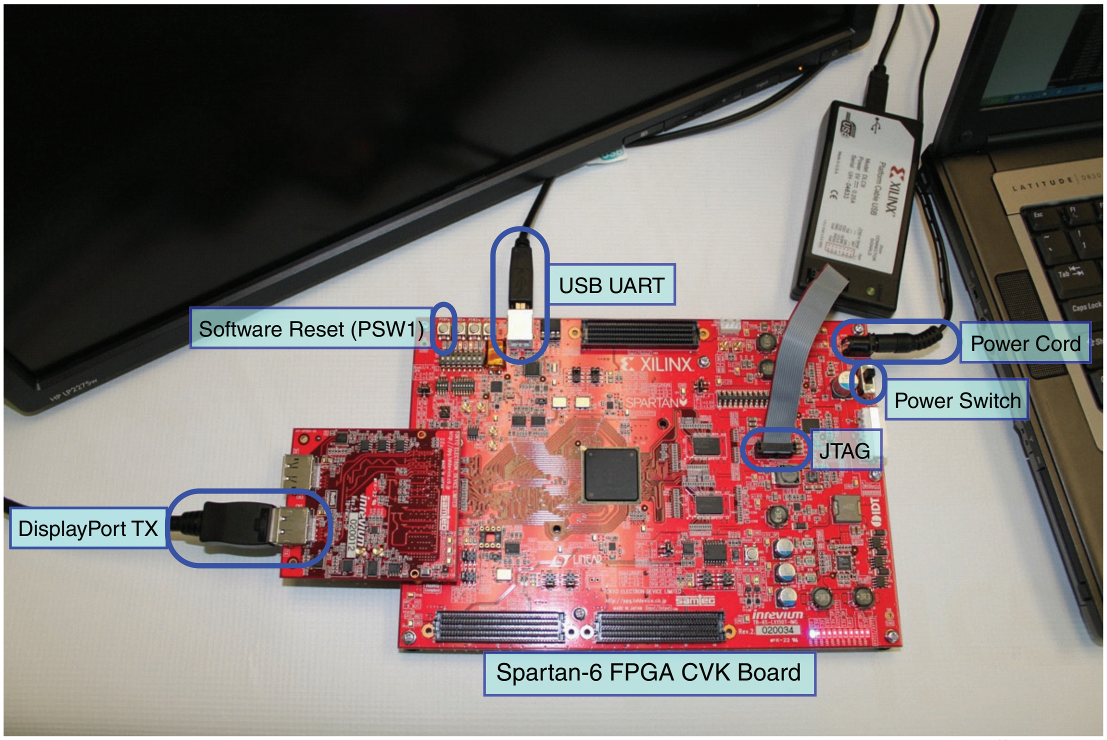


Figure 5: **CVK Board Setup**

The USB UART is detected and installed automatically onto a COM port. The device manager should be checked to see which COM port it has been assigned to. On the host computer, a terminal application such as HyperTerminal or PuTTy should be opened in serial mode and connected to the COM port with these settings:

Baud: 115200
Parity: None

When the board is connected, this bitstream should be downloaded:

`XAPP493/sdk_workspace/hw_platform_0/download.bit.`

The terminal display should be as shown in Table 14.

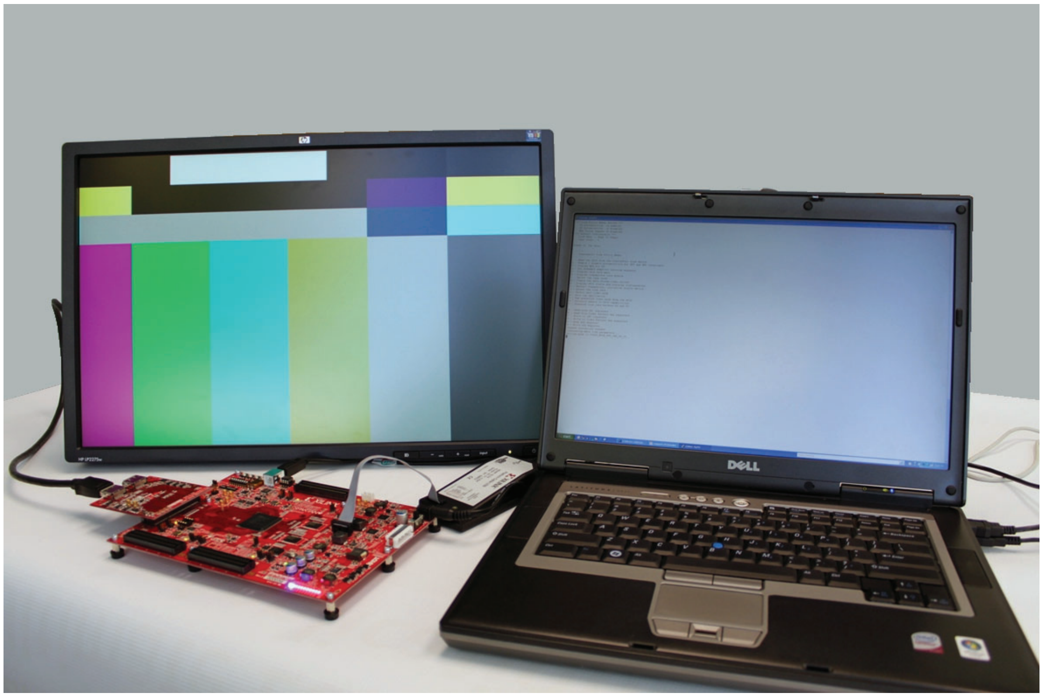
Table 14: **Terminal Display after Bitstream Download**

Displayed on Terminal

```
*** Link Policy Maker Active ***
   TX Autodetection  is enabled
   RX Autodetection  is disabled
   MSA Filter Bypass is disabled
*** Default Configuration ***
   Link Rate  : High (2.7Gbps)
   Lane Count : 4

Press 'h' for help

- - - - - - - - - - - - - - - - - - - -
-   Displayport Link Policy Maker   -
- - - - - - - - - - - - - - - - - - - -
; - Read the EDID from the DisplayPort sink device
a - Enable / disable autodetection for HPD and HPD interrupts
d - Display MSA for TX
g - Run standard adaptive training sequence
h - Display this help menu
k - Display transmitter core status
l - Adjust the lane count
m - Toggle the main stream video on/off
p - Change the video pattern type
s - Display DPCD status and training configuration
v - Select next video mode
x - Exit the application
1 - Adjust TX Voltage Swing
2 - Adjust TX Preemphasis

A - Read from SRC registers
B - Read from Video Pattern Gen registers
C - Write to SRC registers
D - Write to Video Pattern Gen registers
R - Read AUX Register
W - Write AUX Register
```

With the terminal now active, pressing **v** enables the first video mode (800 x 600) with the color bars, as shown in Figure 6. The system is now running. For more information on the different terminal options or to modify the terminal options, refer to Software Implementation, page 3.

> ***Note:*** If the DisplayPort monitor does not display the color bars, double-check the cable connections. Also check whether the monitor is set to the correct DisplayPort input source. If the terminal does not appear on the PC, double-check whether terminal is on the correct COM port and that the terminal baud rate is set to 115200 with parity disabled.



X493_06_051510

Figure 6: **Working System**

Files

The reference design is organized into five directories:

- `doc`: Contains documentation relevant to the reference design
- `ise_top_level`: Contains the top-level ISE tools project
- `display_port_source_policy_maker`: Contains the DisplayPort Source Policy Maker, which is an EDK system
- `design_files`: Contains additional design files not generated by the ISE tools or EDK.
- `sdk_workspace`: Contains the SDK workspace files

Figure 7 shows the directory hierarchy and the most important files in the directories.

Folder / File	Description
XAPP493	
README.txt	
doc	
XAPP493.pdf	
ise_top_level	
dport_source_ref_design.xise	Top-level ISE project. This should be the first design file opened.
ipcore_dir	
displayport_v1_2	
dport_link.ngc	Netlist for DisplayPort link layer. The Macro Search path for Translate should point to this directory.
source	
displayport_v1_2.v	DisplayPort transmitter generated by the CORE Generator tool
dport_link.v	Black-box module for the DisplayPort link layer. This is replaced by dport_linkngc during translate.
dport_tx_defs.v	
dport_tx_phy.v	PHY layer for the DisplayPort transmitter, including AUX channel and GTP transceivers
display_port_source_policy_maker	
display_port_source_policy_maker.xmp	EDK project. This is managed by the ISE tools project listed above
SDK	
SDK_Export	
hw	
display_port_source_policy_maker.xml	XML file for import into SDK. Generated using Export Hardware Design To SDK from within the ISE tools.
pcores	
plbv46_to_apb_bridge_v1_00_a	The PLB-to-APB bridge PCORE
design_files	
displayport_v1_2_exdes.v	Top-level Verilog for the reference design
displayport_v1_2.ucf	UCF file for the reference design
dp_pll.v	PLL and global clock buffer instantiations. Takes in a 200 MHz clock and produces a 40 MHz and 65 MHz clock.
patgen	
video_pat_gen.v	Wrapper for the Color Bar pattern generator, Display Timing Controller, and APB registers
hdclrbar.v	Color Bar pattern generator
disp_timing_controller.vhd	Display Timing Controller
disp_defs_pkg.vhd	Definitions for the Display Timing Controller
regs.v	Pattern Generator control registers that are accessed via the APB interface
sdk_workspace	
dp_source_policy_maker_00	
src	C source code for the reference design. Descriptions can be found in SOFTWARE IMPLEMENTATION.
Debug	
dp_source_policy_maker_00.elf	ELF file for download to the processor via XMD or for updating the bitstream
hw_platform_0	Low-level hardware platform SDK project. This contains the imported system XML file, BMM file and BIT files.
download.bit	BIT file for download to the FPGA
standalone_bsp_0	Contains the Microblaze processor includes and standard Xilinx libraries. This folder is kept up to date by SDK.

X493_07_061810

Figure 7: **Design File Hierarchy and Descriptions**

Recreating the System

This section discusses how to recreate the system from the beginning using the ISE tools, the CORE Generator software, EDK, and SDK. The basic steps to recreate the project from the beginning are:

Step 1: Set Up the Directory Structure

Step 2: Create the ISE Tools System

Step 3: Generate and Integrate the Cores

Step 4: Create an EDK System

Step 5: Generate the Bitstream

Step 6: Create an SDK Project

Step 7: Update the Bitstream

Step 1: Set Up the Directory Structure

Setting up the directory structure is very important to maintain readability of the directories and to avoid duplicate files. Create a directory structure as shown in Figure 8. Remember the location of the XAPP folder, because it is referenced multiple times throughout the use of this application note.

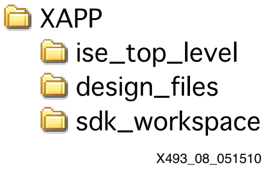


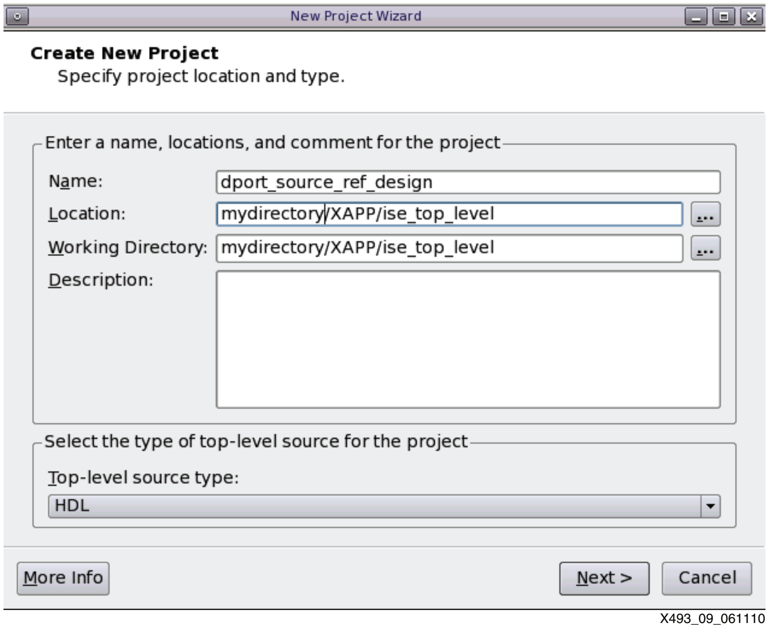
Figure 8: **Directory Structure Setup**

After creating the directories, copy the contents of the original `XAPP493/design_files` directory to the newly created `design_files` directory.

Step 2: Create the ISE Tools System

All hardware source files are managed by the ISE tools project to facilitate implementation. Open the ISE tools and create a new project by clicking **File** → **New Project…** and enter the project **Name: `dport_source_ref_design`**.

Set **Location:** and **Working Directory:** to **`<mydirectory>/XAPP/ise_top_level`**, where *`<mydirectory>`* is the location where the application note folder is placed. Set the **Top-level source type** to **HDL**, as shown in Figure 9.



Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name: dport_source_ref_design
Location: mydirectory/XAPP/ise_top_level
Working Directory: mydirectory/XAPP/ise_top_level
Description:

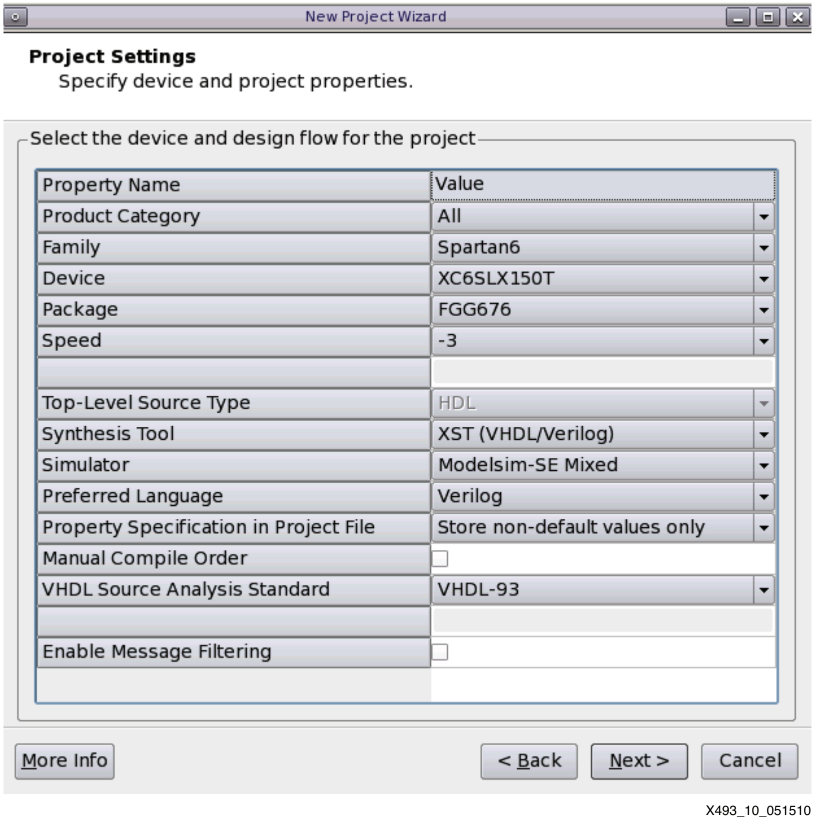
Select the type of top-level source for the project

Top-level source type: HDL

Figure 9: **ISE Tools New Project Wizard**

Set the device and project properties to select the correct device for the CVK board (Figure 10):

- **Family**: **Spartan6**
- **Product**: **XC6SLX150T**
- **Package**: **FGG676**
- **Speed**: **-3**



Project Settings
Specify device and project properties.

Select the device and design flow for the project

Property Name	Value
Product Category	All
Family	Spartan6
Device	XC6SLX150T
Package	FGG676
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE Mixed
Preferred Language	Verilog
Property Specification in Project File	Store non-default values only
Manual Compile Order	□
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	□

Figure 10: **ISE Tools Project Settings**

Click **Next**, and then **Finish** to create the ISE tools system.

The system is now ready to have source files added to it. The first files to be added are the DisplayPort transmitter files. These files are created using the CORE Generator software, as described in the next step.

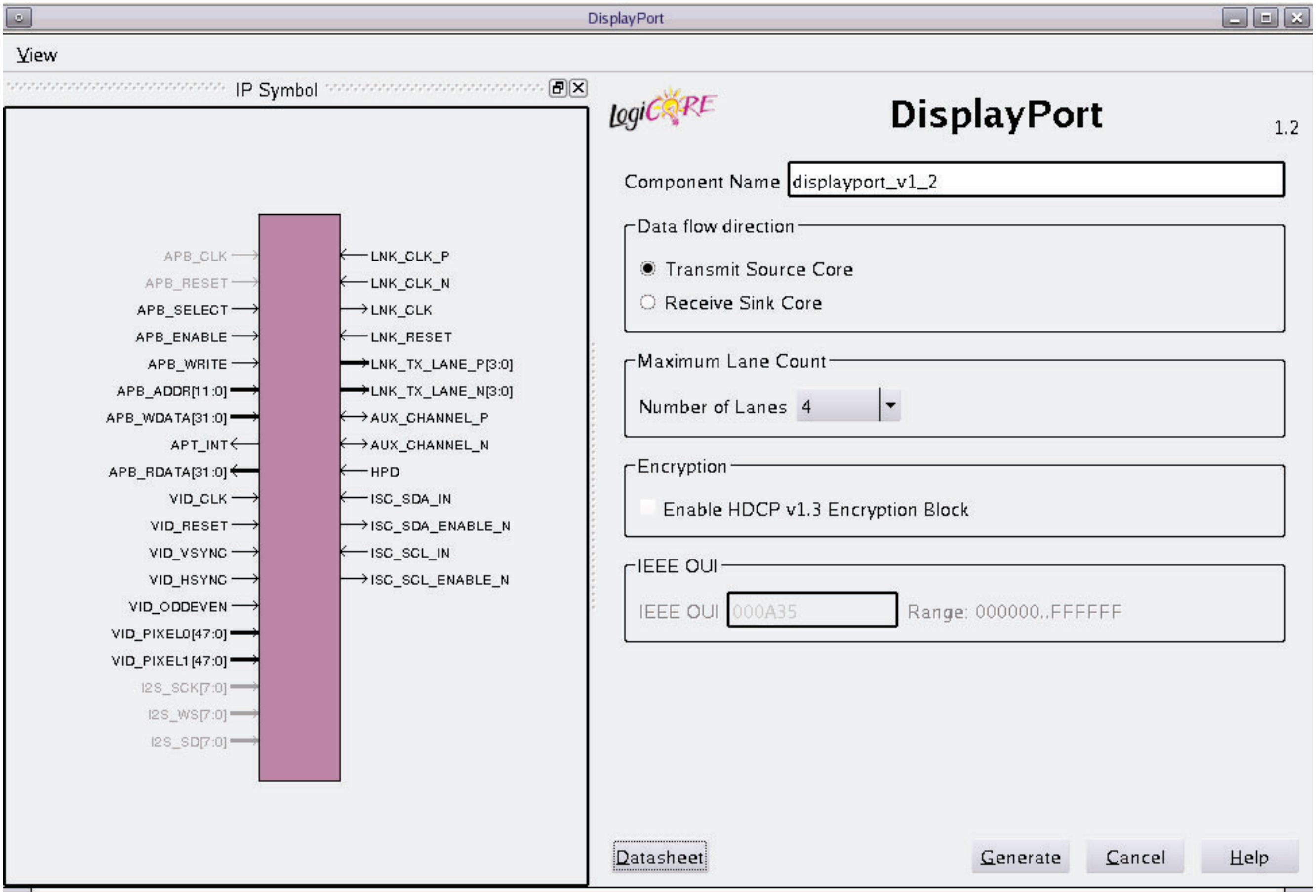
Step 3: Generate and Integrate the Cores

Open the CORE Generator software from the ISE tools by clicking on **Tools** → **CORE Generator…**. When the CORE Generator software opens, navigate to the DisplayPort version 1.2 core found in the **Standard Bus Interfaces** → **DisplayPort** directory, and double-click the entry.

> ***Note:*** If the core is not available, a license might be required. Information on obtaining a license can be found at http://www.xilinx.com/products/ipcenter/ipaccess_fee.htm.

When the customization window is open, set the options as shown below:

- Enter the Component Name as **`displayport_v1_2`**
- Select the **Transmit Source Core** radio button
- Set **Number of Lanes** to **4**
- Uncheck **Enable HDCP v1.3 Encryption Block**
- Click **Generate**



DisplayPort 1.2

Component Name displayport_v1_2

Data flow direction

● Transmit Source Core

○ Receive Sink Core

Maximum Lane Count

Number of Lanes 4

Encryption

□ Enable HDCP v1.3 Encryption Block

IEEE OUI

IEEE OUI 000A35 Range: 000000..FFFFFF

X493_11_062410

Figure 11: **DisplayPort LogiCORE IP Generation**

After the core is generated, it can be integrated into the ISE tools system. The generated core is in the `<mydirectory>/XAPP/ise_top_level/ipcore_dir/displayport_v1_2` directory. As part of the DisplayPort LogiCORE IP, an example design was created, which includes a simple source policy maker. The example design provided with the LogiCORE IP is not used in the current application; instead, it is replaced by the example design provided at `XAPP493/design_files/displayport_v1_2_exdes.v`. The new example design contains the MicroBlaze processor-based policy maker from this application note.

Right-click in the Hierarchy window within the ISE tools and select **Add Source**. Add these sources to the project:

```
<mydirectory>/XAPP/design_files/displayport_v1_2_exdes.v
<mydirectory>/XAPP/design_files/dp_pll.v
<mydirectory>/XAPP/design_files/displayport_v1_2.ucf
<mydirectory>/XAPP/design_files/patgen/video_pat_gen.v
<mydirectory>/XAPP/design_files/patgen/hdclrbar.v
<mydirectory>/XAPP/design_files/patgen/disp_timing_controller.vhd
<mydirectory>/XAPP/design_files/patgen/disp_defs_pkg.vhd
<mydirectory>/XAPP/design_files/patgen/regs.v
<mydirectory>/XAPP/ise_top_level/ipcore_dir/displayport_v1_2/source/displ
ayport_v1_2.v
<mydirectory>/XAPP/ise_top_level/ipcore_dir/displayport_v1_2/source/dport
_link.v
<mydirectory>/XAPP/ise_top_level/ipcore_dir/displayport_v1_2/source/dport
_tx_phy.v
```

These two files should automatically be included after importing the source files, as shown in Figure 12:

```
<mydirectory>/XAPP/ise_top_level/ipcore_dir/displayport_v1_2/source/dport
_tx_defs.v
<mydirectory>/XAPP/design_files/patgen/def.v
```

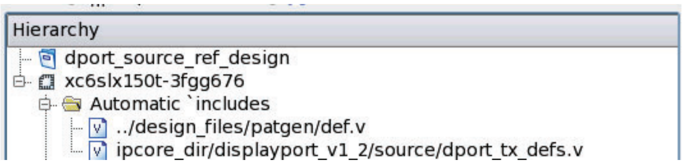
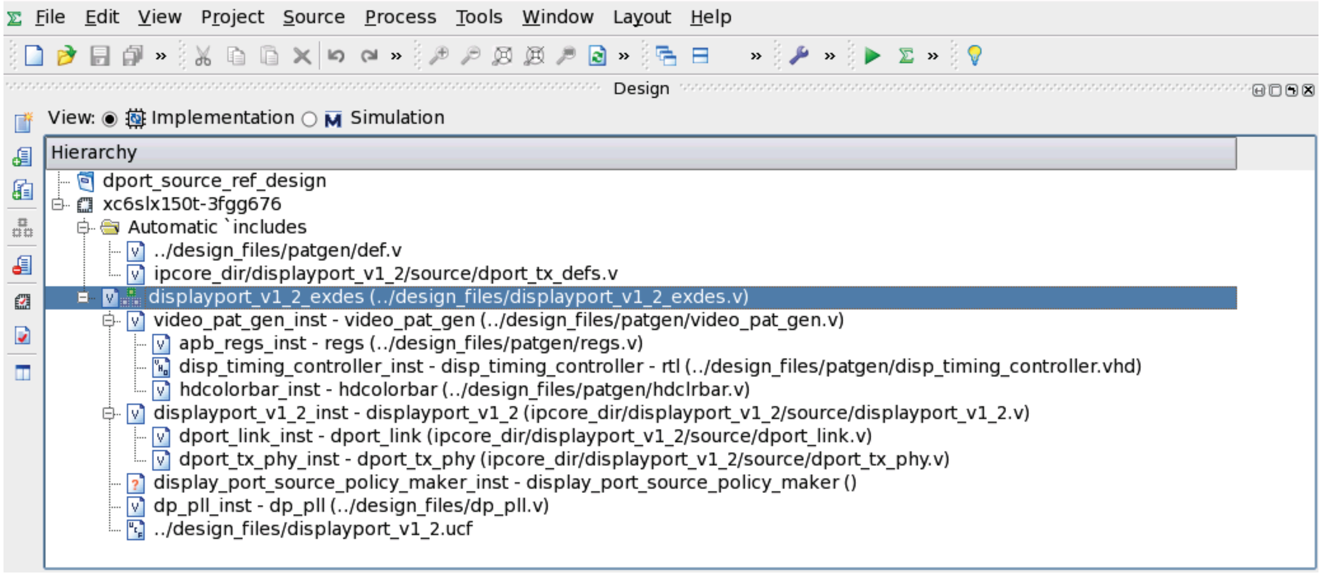


Figure 12: **Automatic Includes**

At this stage, the project should appear as shown in Figure 13. All necessary sources except the EDK subsystem have been added to the project.


File Edit View Project Source Process Tools Window Layout Help
Design
View: ● Implementation ○ Simulation
Hierarchy

```
dport_source_ref_design
xc6slx150t-3fgg676
  Automatic `includes
    ../design_files/patgen/def.v
    ipcore_dir/displayport_v1_2/source/dport_tx_defs.v
  displayport_v1_2_exdes (../design_files/displayport_v1_2_exdes.v)
    video_pat_gen_inst - video_pat_gen (../design_files/patgen/video_pat_gen.v)
      apb_regs_inst - regs (../design_files/patgen/regs.v)
      disp_timing_controller_inst - disp_timing_controller - rtl (../design_files/patgen/disp_timing_controller.vhd)
      hdcolorbar_inst - hdcolorbar (../design_files/patgen/hdclrbar.v)
    displayport_v1_2_inst - displayport_v1_2 (ipcore_dir/displayport_v1_2/source/displayport_v1_2.v)
      dport_link_inst - dport_link (ipcore_dir/displayport_v1_2/source/dport_link.v)
      dport_tx_phy_inst - dport_tx_phy (ipcore_dir/displayport_v1_2/source/dport_tx_phy.v)
    display_port_source_policy_maker_inst - display_port_source_policy_maker ()
    dp_pll_inst - dp_pll (../design_files/dp_pll.v)
    ../design_files/displayport_v1_2.ucf
```

X493_13_051510

Figure 13: **Partial ISE Tools System**

Now that most of the source files are in the ISE tools project, a few modifications need to be made to the DisplayPort LogiCORE IP to take advantage of some Spartan-6 FPGA features and to tailor the LogiCORE IP for the Spartan-6 FPGA CVK board.

The DisplayPort AUX channel is first modified to take advantage of the capabilities of the Spartan-6 FPGA I/Os. The AUX channel that is part of the LogiCORE IP is comprised of two sets of unidirectional differential pairs, one for transmitting and one for receiving. Spartan-6 FPGAs support bidirectional differential I/Os for the DisplayPort AUX channel.

Open `displayport_v1_2_inst` by double-clicking it in the Hierarchy window. Around line 95–98, the AUX channel ports (`aux_tx_out_channel_p`, `aux_tx_out_channel_n`, `aux_tx_in_channel_p`, `aux_tx_in_channel_n`) are declared. To accommodate the bidirectional differential channel, ports `aux_tx_io_p` and `aux_tx_io_n` need to be added. Add these to the port list near line 99 of `displayport_v1_2.v` as follows:

```
inout wire aux_tx_io_p,
inout wire aux_tx_io_n,
```

Near line 227 of `displayport_v1_2.v`, add these wires to connect to `dport_tx_phy`:

```
.aux_tx_io_p          (aux_tx_io_p),
.aux_tx_io_n          (aux_tx_io_n),
```

To add the ports to `dport_tx_phy`, double-click `dport_tx_phy_inst` in the Hierarchy window. Near line 112 of `dport_tx_phy.v`, uncomment or add these lines:

```
inout wire aux_tx_io_p,
inout wire aux_tx_io_n,
```

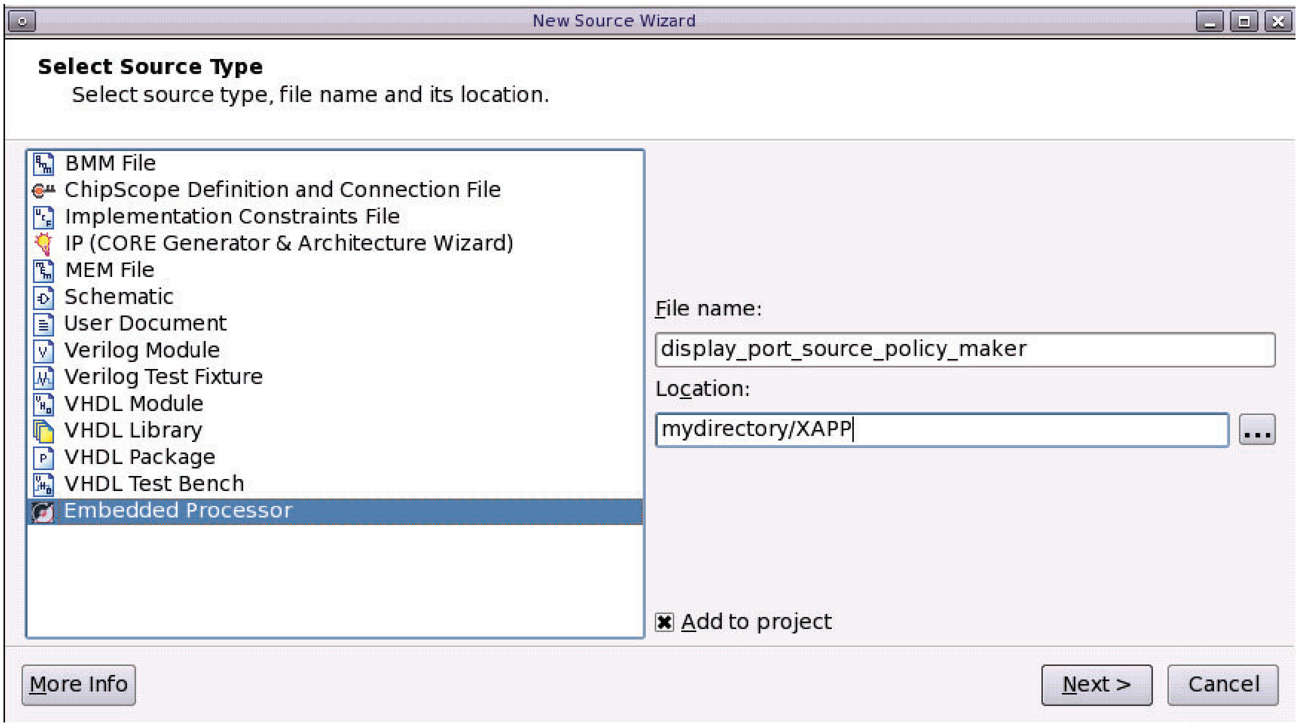
The bidirectional I/Os now need to be enabled. Near line 593 of `dport_tx_phy.v`, the AUX channel is defined. However, the IOBUFDS for the Spartan-6 FPGA is commented out. Replace the old AUX channel definition with the new AUX channel definition, as shown in Table 15.

Table 15: **AUX Channel Verilog**

Old AUX Channel	New AUX Channel
`// ---------------------------------------------`	`// ---------------------------------------------`
`// AUX Channel`	`// AUX Channel`
`// ---------------------------------------------`	`// ---------------------------------------------`
`//    generate if (pDEVICE_FAMILY == "virtex5") begin`	`    generate if ( pDEVICE_FAMILY == "virtex5" || pDEVICE_FAMILY == "virtex6") begin`
`    IBUFDS`	`    IBUFDS`
`    #(  .DIFF_TERM ("TRUE"),`	`    #(  .DIFF_TERM ("TRUE"),`
`        .IOSTANDARD ("LVDS_25")`	`        .IOSTANDARD ("LVDS_25")`
`    ) aux_tx_in_channel_inst (`	`    ) aux_tx_in_channel_inst (`
`        .O  (aux_data_in),`	`        .O  (aux_data_in),`
`        .I  (aux_tx_in_channel_p),`	`        .I  (aux_tx_in_channel_p),`
`        .IB (aux_tx_in_channel_n)`	`        .IB (aux_tx_in_channel_n)`
`    );`	`    );`
`    OBUFTDS_LVDS_25 aux_tx_out_channel_inst (`	`    OBUFTDS_LVDS_25 aux_tx_out_channel_inst (`
`        .O   (aux_tx_out_channel_p),`	`        .O   (aux_tx_out_channel_p),`
`        .OB  (aux_tx_out_channel_n),`	`        .OB  (aux_tx_out_channel_n),`
`        .I   (aux_data_out),`	`        .I   (aux_data_out),`
`        .T   (aux_data_enable_n)`	`        .T   (aux_data_enable_n)`
`    );`	`    );`
`//    end`	`    end`
`//    endgenerate`	`    endgenerate`
`//    generate if (pDEVICE_FAMILY == "spartan6") begin`	`    generate if ( pDEVICE_FAMILY == "spartan6") begin`
`//     IOBUFDS`	`     IOBUFDS`
`//     # (.IOSTANDARD ("DISPLAY_PORT"))`	`     # (.IOSTANDARD ("DISPLAY_PORT"))`
`//     aux_rx_channel_io_inst`	`     aux_rx_channel_io_inst`
`//         (.O    (aux_data_in),`	`         (.O    (aux_data_in),`
`//          .IO   (aux_tx_io_p),`	`          .IO   (aux_tx_io_p),`
`//          .IOB  (aux_tx_io_n),`	`          .IOB  (aux_tx_io_n),`
`//          .I    (aux_data_out),`	`          .I    (aux_data_out),`
`//          .T    (aux_data_enable_n));`	`          .T    (aux_data_enable_n));`
`//`	
`//     assign aux_tx_out_channel_p = 1'b0;`	`     assign aux_tx_out_channel_p = 1'b 0;`
`//     assign aux_tx_out_channel_n = 1'b0;`	`     assign aux_tx_out_channel_n = 1'b 0;`
`//    end`	`    end`
`//    endgenerate`	`    endgenerate`

Step 4: Create an EDK System

To add an EDK system to the ISE tools project, right-click in the Hierarchy window within the ISE tools and select **New Source**. Select **Embedded Processor** as the source type. Set the File name to `display_port_source_policy_maker` and the Location to *<mydirectory>*/XAPP, and ensure the **Add to project** box is checked, as shown in Figure 14.



New Source Wizard

Select Source Type

Select source type, file name and its location.

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: display_port_source_policy_maker

Location: mydirectory/XAPP

☒ Add to project

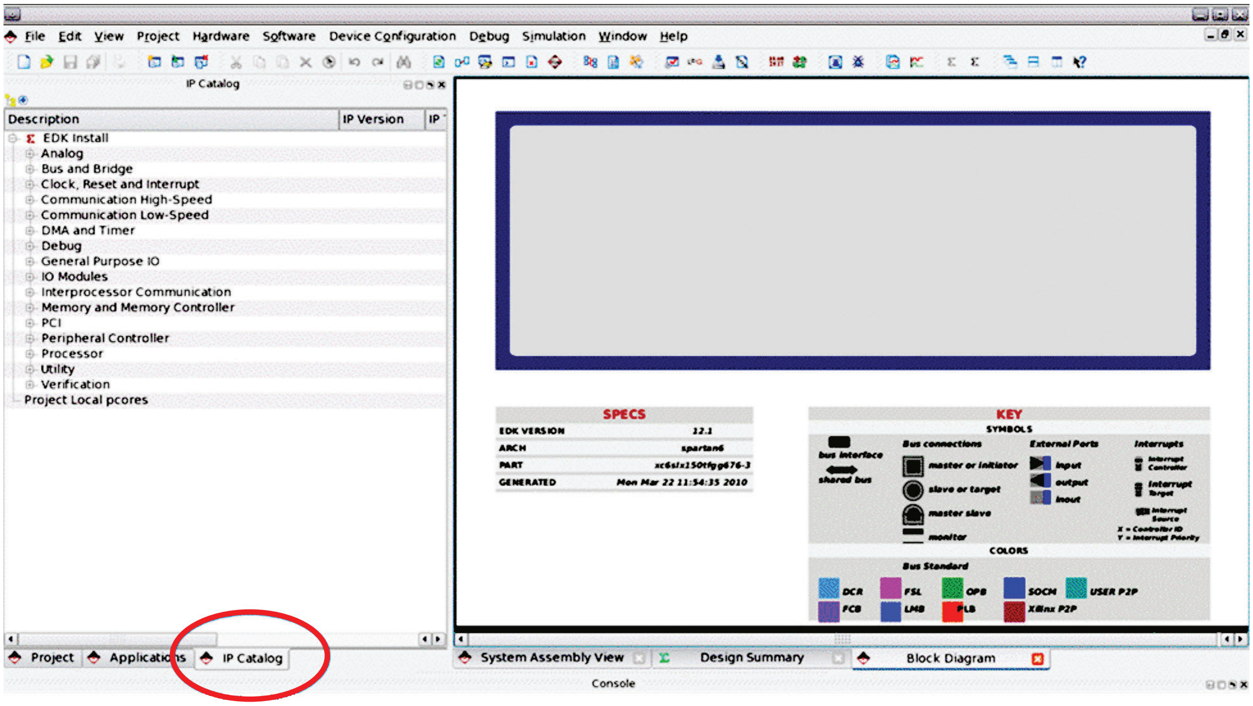
X493_14_061110

Figure 14: **ISE Tools New Source Wizard for EDK Insertion**

Select **Next**, and then **Finish**. This launches XPS.

Note: If a prompt appears asking whether to create a system using Base System Builder, select **No**.

XPS should now be active, and an empty system should be shown. Open the IP Catalog, as shown in Figure 15. From this screen, all required IP is added to the system.



SPECS	
EDK VERSION	12.1
ARCH	spartan6
PART	xc6slx150tfgg676-3
GENERATED	Mon Mar 22 11:54:35 2010

X493_15_051510

Figure 15: **EDK IP Catalog Selection**

Project Local pcores is empty, and the plb-to-apb bridge IP must be added. Copy the `plbv46_to_apb_bridge_v1_00_a` folder from the reference design files in `XAPP493/display_port_source_policy_maker/pcores` to the current design `<mydirectory>/XAPP/display_port_source_policy_maker/pcores`. After the directory has been copied, click **Project** → **Rescan User Repositories** to refresh the IP Catalog.

Note: If a prompt appears asking whether to create a system using Base System Builder, select **No**.

Add the IP

Before adding the IP, click the System Assembly View tab to be able to see the IP as it is added.

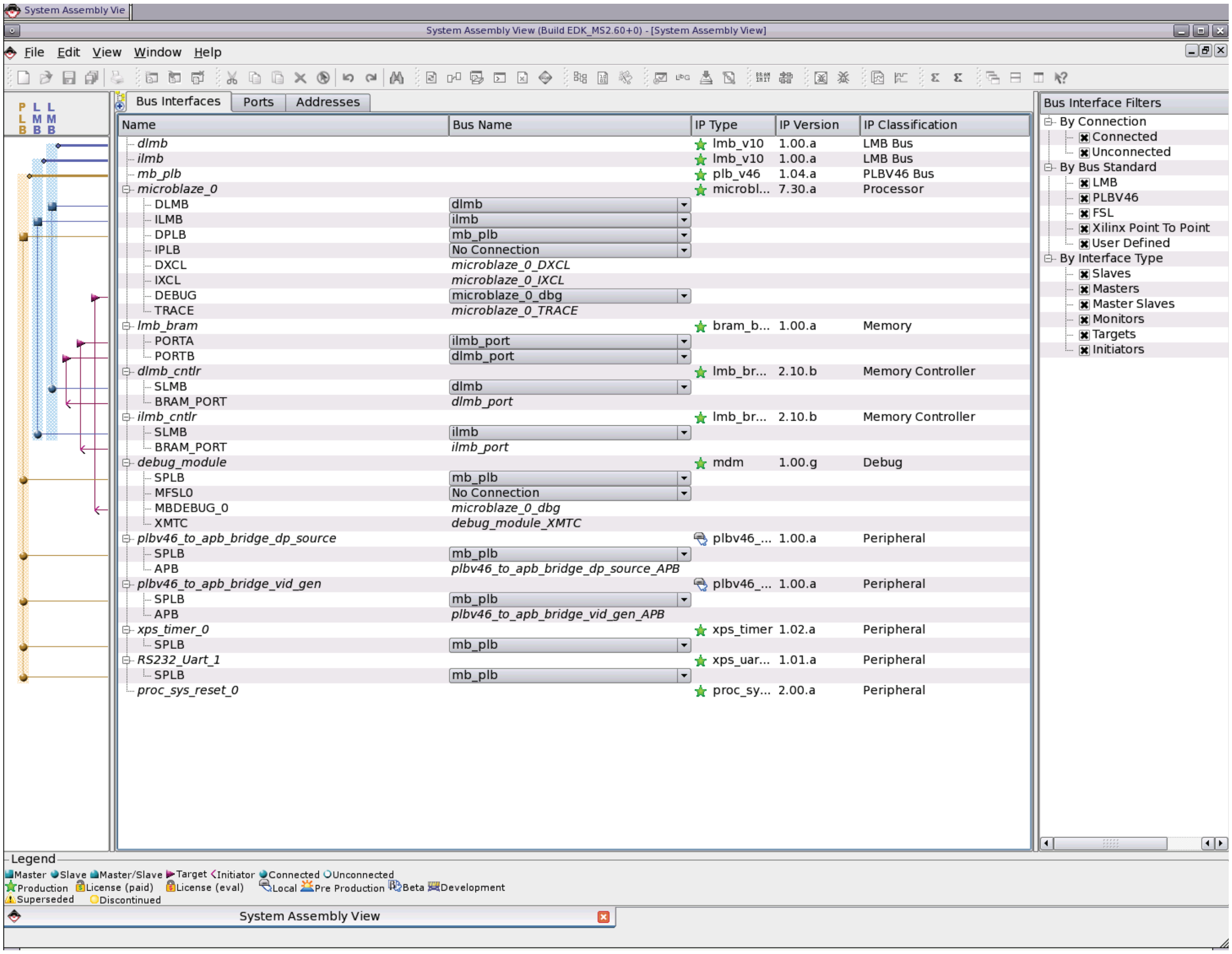
Add the cores listed in Table 16 to the system by locating them in the IP catalog and double-clicking them. After the IP names appear in the System Assembly View, rename them as shown in Table 16.

Table 16: **EDK Required IP**

IP Location	Version	Rename to:
Processor → MicroBlaze	7.30.a	`microblaze_0`
Bus and Bridge → Processor Local Bus (PLB) 4.6	1.04.a	`mb_plb`
Bus and Bridge → Local Memory Bus (LMB) 1.0	1.00.a	`ilmb`
Bus and Bridge → Local Memory Bus (LMB) 1.0	1.00.a	`dlmb`
Memory and Memory Controller → LMB BRAM Controller	2.10.b	`ilmb_cntlr`
Memory and Memory Controller → LMB BRAM Controller	2.10.b	`dlmb_cntlr`
Memory and Memory Controller → Block Ram (BRAM) Block	1.00.a	`lmb_bram`
Communication Low-Speed → XPS UART (Lite)	1.01.a	`RS232_Uart_1`
Debug → MicroBlaze Debug Module (MDM)	1.00.g	`debug_module`
Clock, Reset and Interrupt → Processor System Reset Module	2.00.a	`proc_sys_reset_0`
DMA and Timer → XPS Timer / Counter	1.02.a	`xps_timer_0`
Project Local pcores → USER → PLBV46_TO_APB_BRIDGE	1.00.a	`plbv46_to_apb_bridge_dp_source`
Project Local pcores → USER → PLBV46_TO_APB_BRIDGE	1.00.a	`plbv46_to_apb_bridge_vid_gen`

Connect the Buses

Navigate to the **Bus Interfaces** tab of the **System Assembly View** and connect all of the PLB bus connections, LMB bus connections, and DEBUG bus connections, as shown in Figure 16. This can be done by clicking the circles, squares, and triangles on the left side of the system assembly view or by using the pull-down menus in the Bus Name column. It might be useful to click the **+** located next to the **Bus Interfaces** tab to see the bus connectivity more clearly.



Name	Bus Name	IP Type	IP Version	IP Classification
dlmb		lmb_v10	1.00.a	LMB Bus
ilmb		lmb_v10	1.00.a	LMB Bus
mb_plb		plb_v46	1.04.a	PLBV46 Bus
microblaze_0		microbl...	7.30.a	Processor
DLMB	dlmb			
ILMB	ilmb			
DPLB	mb_plb			
IPLB	No Connection			
DXCL	*microblaze_0_DXCL*			
IXCL	*microblaze_0_IXCL*			
DEBUG	microblaze_0_dbg			
TRACE	*microblaze_0_TRACE*			
lmb_bram		bram_b...	1.00.a	Memory
PORTA	ilmb_port			
PORTB	dlmb_port			
dlmb_cntlr		lmb_br...	2.10.b	Memory Controller
SLMB	dlmb			
BRAM_PORT	*dlmb_port*			
ilmb_cntlr		lmb_br...	2.10.b	Memory Controller
SLMB	ilmb			
BRAM_PORT	*ilmb_port*			
debug_module		mdm	1.00.g	Debug
SPLB	mb_plb			
MFSL0	No Connection			
MBDEBUG_0	*microblaze_0_dbg*			
XMTC	*debug_module_XMTC*			
plbv46_to_apb_bridge_dp_source		plbv46_...	1.00.a	Peripheral
SPLB	mb_plb			
APB	*plbv46_to_apb_bridge_dp_source_APB*			
plbv46_to_apb_bridge_vid_gen		plbv46_...	1.00.a	Peripheral
SPLB	mb_plb			
APB	*plbv46_to_apb_bridge_vid_gen_APB*			
xps_timer_0		xps_timer	1.02.a	Peripheral
SPLB	mb_plb			
RS232_Uart_1		xps_uar...	1.01.a	Peripheral
SPLB	mb_plb			
proc_sys_reset_0		proc_sy...	2.00.a	Peripheral

X493_16_051510

Figure 16: **EDK Bus Connectivity**

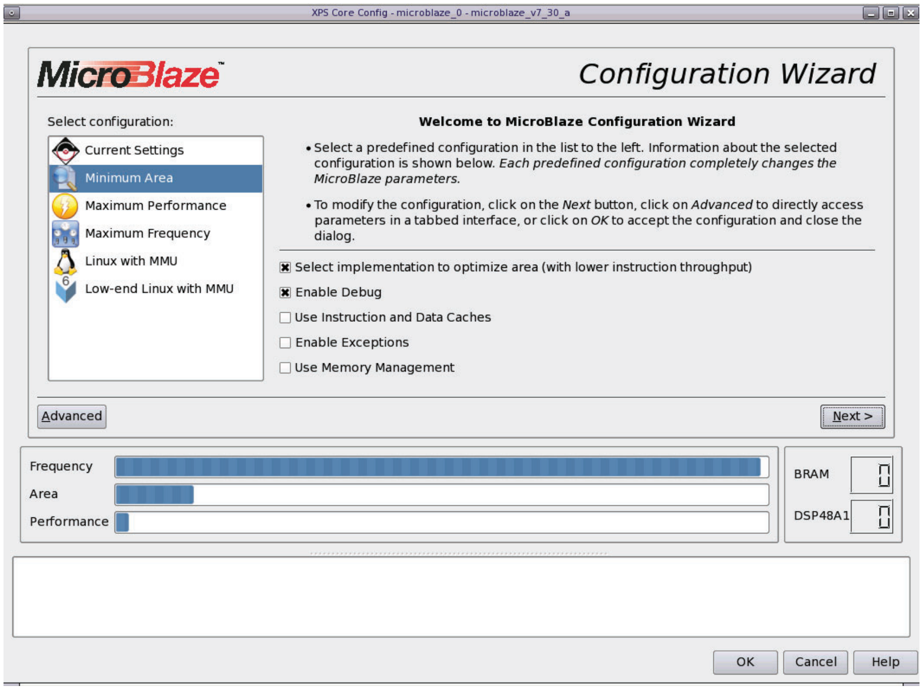
After all of the buses are connected, click the **Addresses** tab and click **Generate Addresses**.

Note: Ensure that `dlmb_cntlr` and `ilmb_cntlr` have a base address of `0x00000000` and a size of 64K.

Now that the bus connections are made, each IP needs to be configured. Navigate to the **Bus Interfaces** tab to begin configuring IP. Many of the IPs in the system are preconfigured or automatically configured; however, the MicroBlaze processor and UART Lite IPs must be configured for this system.

MicroBlaze Configuration

Double-click the **microblaze_0** IP, select **Minimum Area** and click **OK**, as shown in Figure 17.



XPS Core Config - microblaze_0 - microblaze_v7_30_a

MicroBlaze Configuration Wizard

Select configuration:

- Current Settings
- Minimum Area
- Maximum Performance
- Maximum Frequency
- Linux with MMU
- Low-end Linux with MMU

Welcome to MicroBlaze Configuration Wizard

- Select a predefined configuration in the list to the left. Information about the selected configuration is shown below. *Each predefined configuration completely changes the MicroBlaze parameters.*
- To modify the configuration, click on the *Next* button, click on *Advanced* to directly access parameters in a tabbed interface, or click on *OK* to accept the configuration and close the dialog.

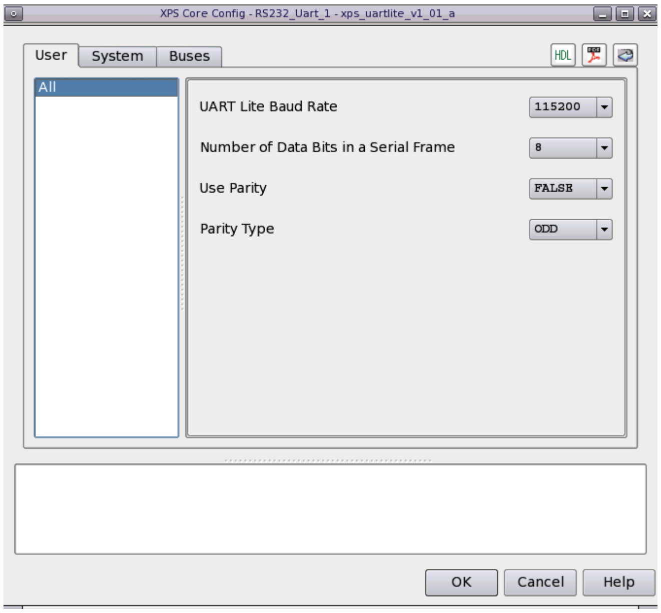
☒ Select implementation to optimize area (with lower instruction throughput)
☒ Enable Debug
□ Use Instruction and Data Caches
□ Enable Exceptions
□ Use Memory Management

X493_17_051510

Figure 17: **MicroBlaze Processor Configuration**

RS-232 UART Configuration

Double-click the **RS232_Uart_1** IP and select the **User** tab. Set the **UART Lite Baud Rate** to **115200**, set **Use Parity** to **FALSE**, and click **OK**. See Figure 18.



XPS Core Config - RS232_Uart_1 - xps_uartlite_v1_01_a

UART Lite Baud Rate	115200
Number of Data Bits in a Serial Frame	8
Use Parity	FALSE
Parity Type	ODD

X493_18_051510

Figure 18: **RS-232 UART Configuration**

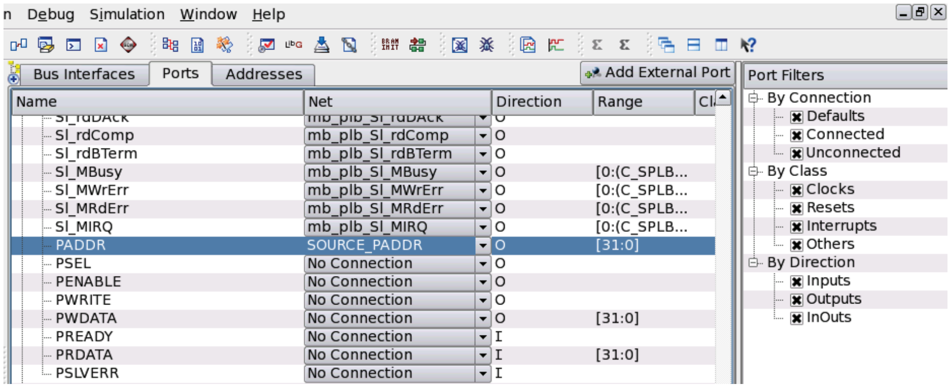
Now that the majority of the system is internally connected and configured, the external connections need to be added. To do this, open the `.mhs` file found on the **Project** tab and add the port declarations shown below on the line following PARAMETER VERSION.

```
PORT RS232_Uart_1_RX_pin = RS232_Uart_1_RX, DIR = I
PORT RS232_Uart_1_TX_pin = RS232_Uart_1_TX, DIR = O
PORT sys_rst_pin = sys_rst_s, DIR = I, RST_POLARITY = 1, SIGIS = RST
PORT SOURCE_PADDR_pin = SOURCE_PADDR, DIR = O, VEC = [31:0]
PORT SOURCE_PSEL_pin = SOURCE_PSEL, DIR = O
PORT SOURCE_PENABLE_pin = SOURCE_PENABLE, DIR = O
PORT SOURCE_PWRITE_pin = SOURCE_PWRITE, DIR = O
PORT SOURCE_PWDATA_pin = SOURCE_PWDATA, DIR = O, VEC = [31:0]
PORT SOURCE_PREADY_pin = SOURCE_PREADY, DIR = I
PORT SOURCE_PRDATA_pin = SOURCE_PRDATA, DIR = I, VEC = [31:0]
PORT SOURCE_PSLVERR_pin = SOURCE_PSLVERR, DIR = I
PORT VID_GEN_PADDR_pin = VID_GEN_PADDR, DIR = O, VEC = [31:0]
PORT VID_GEN_PSEL_pin = VID_GEN_PSEL, DIR = O
PORT VID_GEN_PENABLE_pin = VID_GEN_PENABLE, DIR = O
PORT VID_GEN_PWRITE_pin = VID_GEN_PWRITE, DIR = O
PORT VID_GEN_PWDATA_pin = VID_GEN_PWDATA, DIR = O, VEC = [31:0]
PORT VID_GEN_PREADY_pin = VID_GEN_PREADY, DIR = I
PORT VID_GEN_PRDATA_pin = VID_GEN_PRDATA, DIR = I, VEC = [31:0]
PORT VID_GEN_PSLVERR_pin = VID_GEN_PSLVERR, DIR = I
PORT SYSTEM_RESET = sys_bus_reset, DIR = O
PORT SYSTEM_CLOCK = sys_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 40000000
PORT PLL_LOCKED = pll_locked, DIR = I
```

Save the file, return to the system assembly view, and click the **Ports** tab.

On the **Ports** tab, expand the **External Ports** to see the newly added ports. The `sys_rst_pin` has a reset polarity of 1, and the `SYSTEM_CLOCK` pin has a frequency of 40,000,000 Hz. If the reset polarity or clock frequency changes, these parameters must be changed to reflect that.

Now that the ports are connected to the top level of the EDK system, they must be connected to the cores internal to the EDK project. To do this, navigate to the **Ports** tab and ensure that the **Defaults** port filter is selected so that the APB bus connections are visible, as shown in Figure 19.



Name	Net	Direction	Range	Cl
Sl_rdDAck	mb_plb_Sl_rdDAck	O		
Sl_rdComp	mb_plb_Sl_rdComp	O		
Sl_rdBTerm	mb_plb_Sl_rdBTerm	O		
Sl_MBusy	mb_plb_Sl_MBusy	O	[0:(C_SPLB...	
Sl_MWrErr	mb_plb_Sl_MWrErr	O	[0:(C_SPLB...	
Sl_MRdErr	mb_plb_Sl_MRdErr	O	[0:(C_SPLB...	
Sl_MIRQ	mb_plb_Sl_MIRQ	O	[0:(C_SPLB...	
PADDR	SOURCE_PADDR	O	[31:0]	
PSEL	No Connection	O		
PENABLE	No Connection	O		
PWRITE	No Connection	O		
PWDATA	No Connection	O	[31:0]	
PREADY	No Connection	I		
PRDATA	No Connection	I	[31:0]	
PSLVERR	No Connection	I		

X493_19_051510

Figure 19: **Default Port Filters**

Connect the ports for the cores:

- Connect `plbv46_to_apb_bridge_dp_source` as follows:

```
PORT PADDR = SOURCE_PADDR
PORT PSEL = SOURCE_PSEL
PORT PENABLE = SOURCE_PENABLE
PORT PWRITE = SOURCE_PWRITE
PORT PWDATA = SOURCE_PWDATA
PORT PREADY = SOURCE_PREADY
PORT PRDATA = SOURCE_PRDATA
PORT PSLVERR = SOURCE_PSLVERR
```

- Connect `proc_sys_reset_0` as follows:

```
PORT Slowest_sync_clk = sys_clk_s
PORT Dcm_locked = pll_locked
PORT Ext_Reset_In = sys_rst_s
PORT MB_Reset = mb_reset
```
  (this is a new signal)
```
PORT Bus_Struct_Reset = sys_bus_reset
PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
```

- Connect `Debug_SYS_Rst` to `debug_module`:

```
PORT Debug_SYS_Rst = Debug_SYS_Rst
```
  (this is a new signal)

- Connect `plbv46_to_apb_bridge_vid_gen` as follows:

```
PORT PADDR = VID_GEN_PADDR
PORT PSEL = VID_GEN_PSEL
PORT PENABLE = VID_GEN_PENABLE
PORT PWRITE = VID_GEN_PWRITE
PORT PWDATA = VID_GEN_PWDATA
PORT PREADY = VID_GEN_PREADY
PORT PRDATA = VID_GEN_PRDATA
PORT PSLVERR = VID_GEN_PSLVERR
```

- Connect `RS232_Uart_1` to its ports:

```
PORT RX = RS232_Uart_1_RX
PORT TX = RS232_Uart_1_TX
```

- Connect clock and reset to the `ilmb` bus and `dlmb` bus:

```
PORT LMB_Clk = sys_clk_s
PORT SYS_Rst = sys_bus_reset
```

- Connect clock and reset to the `mb_plb` bus:

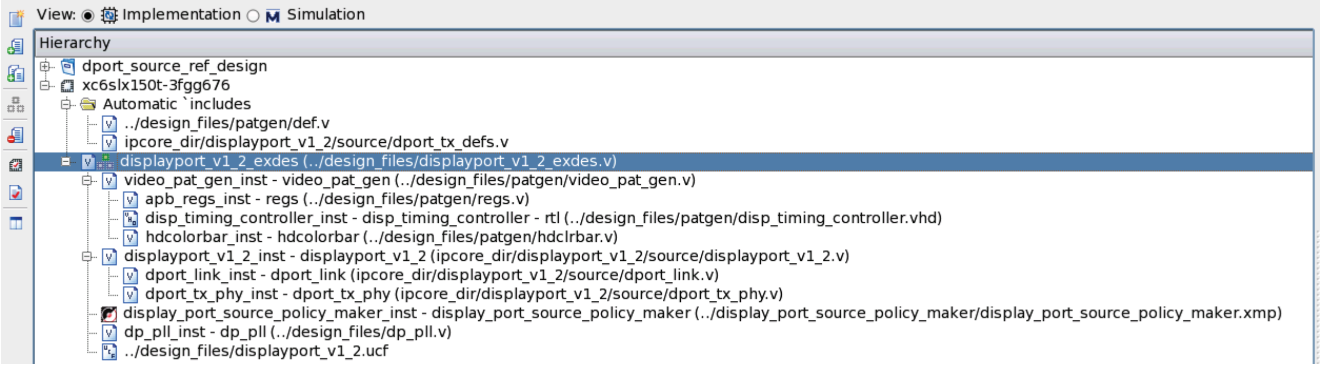
```
PORT PLB_Clk = sys_clk_s
PORT SYS_Rst = sys_bus_reset
```

- Connect reset to `microblaze:`

```
PORT MB_RESET = mb_reset
```

When all of the connections have been made, the `.mhs` file should appear very similar to the `.mhs` included in the reference design (`XAPP493/display_port_source_policy_maker/display_port_source_policy_maker.mhs`).

The XPS portion is now complete and is ready to be integrated into the top-level design. XPS can be closed at this point, and work can resume in the ISE tools. When returning to the ISE tools, `display_port_source_policy_maker_inst` should be populated with the EDK project, as shown in Figure 20.



View: ● Implementation ○ Simulation

Hierarchy

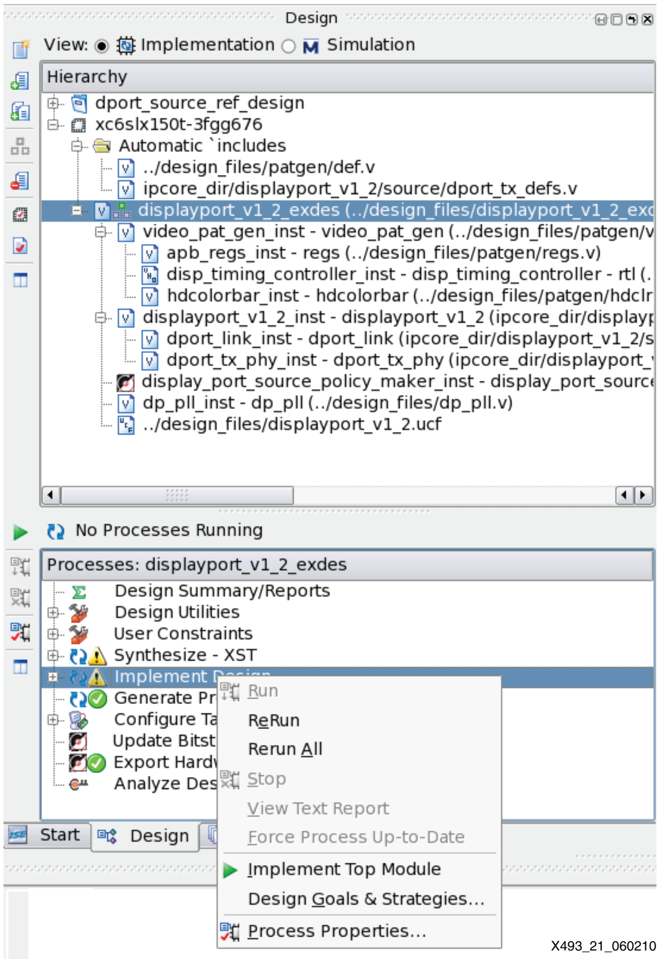
```
dport_source_ref_design
xc6slx150t-3fgg676
  Automatic `includes
    ../design_files/patgen/def.v
    ipcore_dir/displayport_v1_2/source/dport_tx_defs.v
  displayport_v1_2_exdes (../design_files/displayport_v1_2_exdes.v)
    video_pat_gen_inst - video_pat_gen (../design_files/patgen/video_pat_gen.v)
      apb_regs_inst - regs (../design_files/patgen/regs.v)
      disp_timing_controller_inst - disp_timing_controller - rtl (../design_files/patgen/disp_timing_controller.vhd)
      hdcolorbar_inst - hdcolorbar (../design_files/patgen/hdclrbar.v)
    displayport_v1_2_inst - displayport_v1_2 (ipcore_dir/displayport_v1_2/source/displayport_v1_2.v)
      dport_link_inst - dport_link (ipcore_dir/displayport_v1_2/source/dport_link.v)
      dport_tx_phy_inst - dport_tx_phy (ipcore_dir/displayport_v1_2/source/dport_tx_phy.v)
    display_port_source_policy_maker_inst - display_port_source_policy_maker (../display_port_source_policy_maker/display_port_source_policy_maker.xmp)
    dp_pll_inst - dp_pll (../design_files/dp_pll.v)
    ../design_files/displayport_v1_2.ucf
```

X493_20_051510

Figure 20: **Complete ISE Tools System**

Step 5: Generate the Bitstream

With the project now containing all required source files, the bitstream can be generated and the platform can be exported to SDK. To generate the bitstream, the macro search path must be set in translate properties to point to the `dport_link.ngc` file. This `.ngc` file is the one generated by the CORE Generator software as part of the DisplayPort LogiCORE IP. To add the search path, click `displayport_v1_2_exdes` in the Hierarchy window. Make sure that the **Design** tab is visible, then right-click **Implement Design**. Select **Process Properties**, and then select **Translate Properties**, and set **Macro Search Path (-sd)** to point to `<`*`mydirectory`*`>/XAPP/ise_top_level/ipcore_dir/displayport_v1_2`. Click **OK**. Refer to Figure 21.



Design

View: ● Implementation ○ Simulation

Hierarchy

```
dport_source_ref_design
xc6slx150t-3fgg676
  Automatic `includes
    ../design_files/patgen/def.v
    ipcore_dir/displayport_v1_2/source/dport_tx_defs.v
  displayport_v1_2_exdes (../design_files/displayport_v1_2_exd
    video_pat_gen_inst - video_pat_gen (../design_files/patgen/v
      apb_regs_inst - regs (../design_files/patgen/regs.v)
      disp_timing_controller_inst - disp_timing_controller - rtl (.
      hdcolorbar_inst - hdcolorbar (../design_files/patgen/hdclr
    displayport_v1_2_inst - displayport_v1_2 (ipcore_dir/displayp
      dport_link_inst - dport_link (ipcore_dir/displayport_v1_2/s
      dport_tx_phy_inst - dport_tx_phy (ipcore_dir/displayport_
    display_port_source_policy_maker_inst - display_port_source
    dp_pll_inst - dp_pll (../design_files/dp_pll.v)
    ../design_files/displayport_v1_2.ucf
```

No Processes Running

Processes: displayport_v1_2_exdes

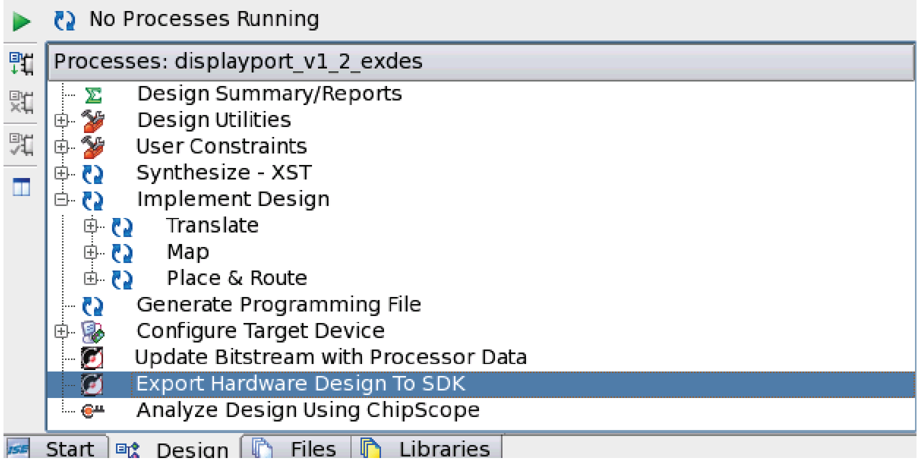
```
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Pr
Configure Ta
Update Bitst
Export Hard
Analyze Des
```

Figure 21: **Implementation Properties**

The design is now ready to be built. Double-click **Generate Programming File**. The design should run through synthesis, implementation, and bitstream generation.

The base hardware system is now built, but it contains no software for the MicroBlaze processor. To add software to the MicroBlaze processor, an SDK project must be created. First, however, the hardware design needs to be exported so that SDK has a reference system.

One of the new features in ISE Design Suite 12.1 is the ability to export an ISE tools design with an EDK subsystem to SDK, as shown in Figure 22.



X493_22_051510

Figure 22: **Export Hardware Design to SDK**

Double-click **Export Hardware Design to SDK**. This creates an XML file in `<`*`mydirectory`*`>/display_port_source_policy_maker/SDK/SDK_Export/hw/` named `display_port_source_policy_maker.xml`. This XML file represents the EDK system and is used by SDK to create a hardware platform.

Step 6: Create an SDK Project

Open SDK and set the Workspace to `<`*`mydirectory`*`>/XAPP/sdk_workspace`.

In SDK, three components are needed to create a software project: a hardware specification, a board support package, and a C or C++ project. The three components are created as described below.

Xilinx Hardware Platform Specification

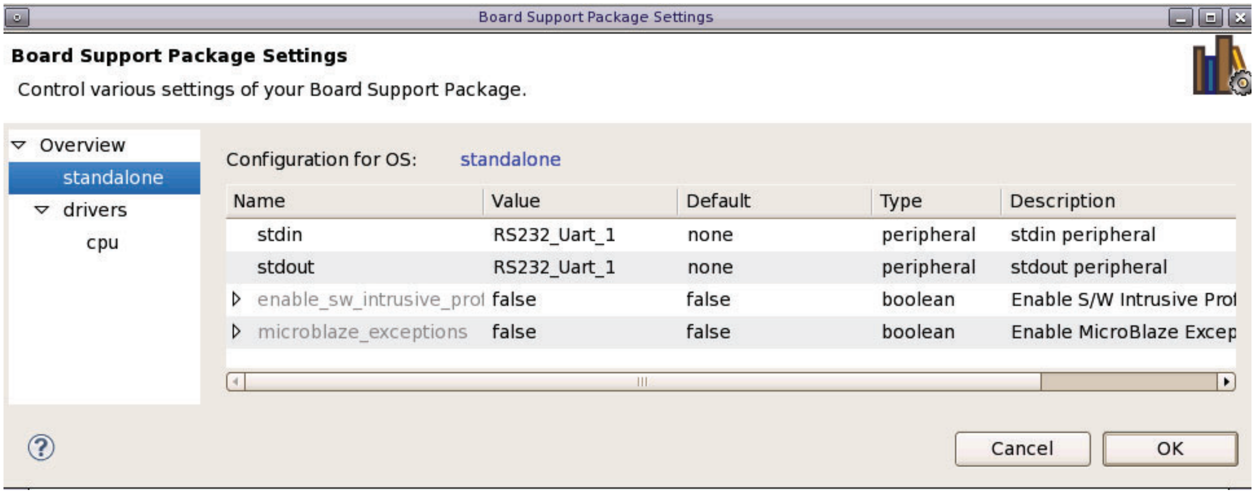
To create the hardware specification:

1. Click **File** → **New** → **Xilinx Hardware Platform Specification**.
2. Set the project name to `hw_platform_0`.
3. Set the target hardware specification to `<`*`mydirectory`*`>/XAPP/display_port_source_policy_maker/SDK/SDK_Export/hw/display_port_source_policy_maker.xml`.
4. Click **Finish**.

Xilinx Board Support Package

To create the board support package:

1. Click **File** → **New** → **Xilinx Board Support Package**.
2. Set the project name to `standalone_bsp_0`.
3. Set the hardware platform to `hw_platform_0`.
4. Set the board support package OS to `standalone`.
5. Click **Finish**.
6. In board support package settings, ensure `stdin` and `stdout` are set to `RS232_Uart_1`, as shown in Figure 23, then click **OK**.



Board Support Package Settings

Control various settings of your Board Support Package.

Configuration for OS: standalone

Name	Value	Default	Type	Description
stdin	RS232_Uart_1	none	peripheral	stdin peripheral
stdout	RS232_Uart_1	none	peripheral	stdout peripheral
enable_sw_intrusive_prol	false	false	boolean	Enable S/W Intrusive Prol
microblaze_exceptions	false	false	boolean	Enable MicroBlaze Excep

X493_23_051510

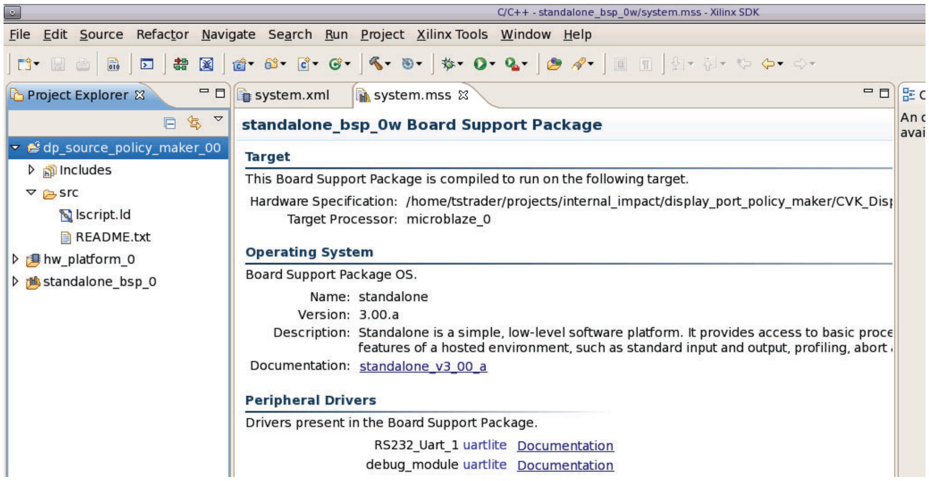
Figure 23: **Board Support Package Settings**

Xilinx C Project

To create the C Project:

1. Click **File** → **New** → **Xilinx C Project**.
2. Set the project name to `dp_source_policy_maker_00`.
3. Set the project template to `Empty Application`.
4. Click **Next**.
5. Click the **Target an existing Board Support Package** radio button and select `standalone_bsp_0`.
6. Click **Finish**.

SDK should now have three projects in the Project Explorer, as shown in Figure 24.



standalone_bsp_0w Board Support Package

Target

This Board Support Package is compiled to run on the following target.

Hardware Specification: /home/tstrader/projects/internal_impact/display_port_policy_maker/CVK_Dis|

Target Processor: microblaze_0

Operating System

Board Support Package OS.

Name: standalone

Version: 3.00.a

Description: Standalone is a simple, low-level software platform. It provides access to basic proce features of a hosted environment, such as standard input and output, profiling, abort

Documentation: standalone_v3_00_a

Peripheral Drivers

Drivers present in the Board Support Package.

RS232_Uart_1 uartlite Documentation

debug_module uartlite Documentation

X493_24_051510

Figure 24: **SDK Project Explorer**

Using a console or folder browser, copy the source files from the reference design folder `XAPP493/sdk_workspace/dp_source_policy_maker_00/src` to `<`*`mydirectory`*`>/XAPP/sdk_workspace/dp_source_policy_maker_00/src`.

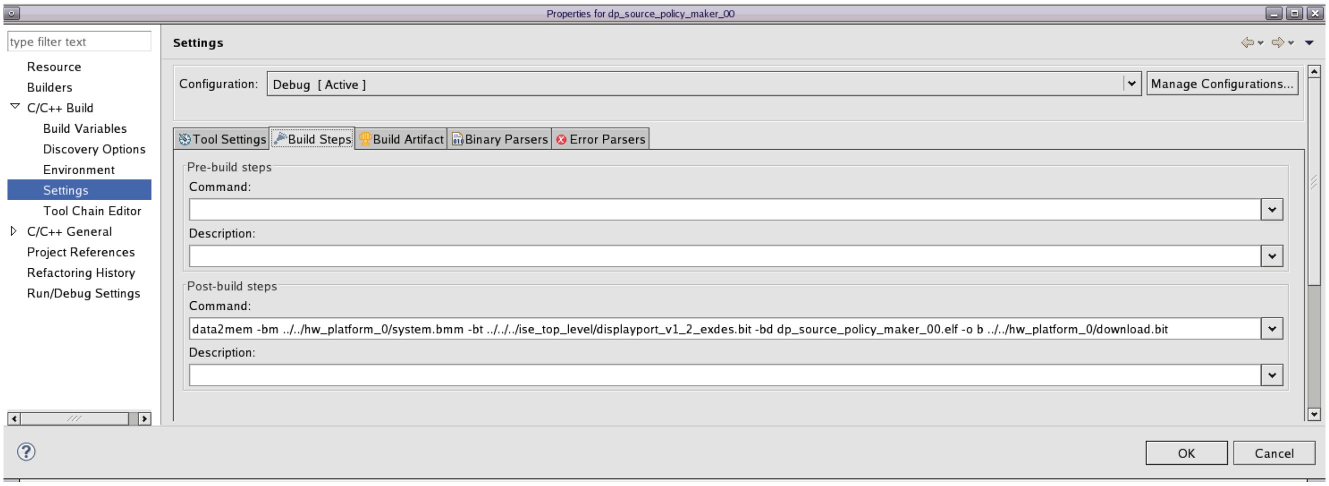
Go back to SDK and refresh the `dp_source_policy_maker_00/src` folder by clicking it in the Project Explorer window and pressing **F5**. The source should automatically compile and place `dp_source_policy_maker_00.elf` in the `<`*`mydirectory`*`>/XAPP/sdk_workspace/dp_source_policy_maker_00/Debug` directory.

Step 7: Update the Bitstream

There are several ways to update the bitstream with processor data. The method used for this application note is a post-processing step in SDK.

From the project explorer in SDK, right-click the `dp_source_policy_maker_00` project and select **C/C++ Build Settings**. Next, click the **Build Steps** tab and, as shown in Figure 25, set the **Command:** in the post-build steps to:

```
data2mem -bm ../../hw_platform_0/system.bmm -bt
../../../ise_top_level/displayport_v1_2_exdes.bit -bd
dp_source_policy_maker_00.elf -o b ../../hw_platform_0/download.bit
```


Properties for dp_source_policy_maker_00

Settings

Configuration: Debug [Active]

Pre-build steps
Command:
Description:

Post-build steps
Command: data2mem -bm ../../hw_platform_0/system.bmm -bt ../../../ise_top_level/displayport_v1_2_exdes.bit -bd dp_source_policy_maker_00.elf -o b ../../hw_platform_0/download.bit
Description:

X493_25_051510

Figure 25: **SDK Post-Build Steps**

This creates a new `download.bit` file in the `<`*`mydirectory`*`>/XAPP/sdk_workspace/hw_platform_0` directory every time the software is rebuilt. The `download.bit` file can now be downloaded to the Spartan-6 FPGA on the CVK board. Refer to Setup and Usage for more information about using the reference design.

Policy Maker Design Footprint

Table 17 shows the resource utilization of the default Policy Maker reference design. The number of block RAMs used is for a more full-featured, user-driven, console-based Policy Maker. Less-featured software implementations are possible with less block RAM utilization.

Table 17: **Policy Maker Design Footprint**

IP	LUTs	Flip-Flops	Block RAMs
MicroBlaze Processor	1,044	836	
UART	140	141	
Timer	343	360	
MDM Debug	120	45	
PLB2APB Bridge	100	156	
PLB2APB Bridge	100	156	
Reset	52	67	
PLB v46	120	128	
LMB			32
Total:	2,019	1,889	32[1]

Notes:

1. Block RAM count can be reduced by decreasing software features.

Reference Design

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

https://secure.xilinx.com/webreg/clickthrough.do?cid=147817

The checklist in Table 18 indicates the tool flow and verification procedures used for the reference design.

Table 18: **Reference Design Matrix**

Parameter	Description
General	
Developer Name	Xilinx
Target Devices (Stepping Level, ES, Production, Speed Grades)	Spartan-6 FPGA (XC6SLX150T-FGG676-3)
Source Code Provided?	Yes
Source Code Format	Verilog, C
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?	No
Testbench Format	Verilog
Simulator Software and Version	ModelSim 6.5c
SPICE/IBIS Simulations?	No

Table 18: **Reference Design Matrix *(Cont'd)***

Parameter	Description
Implementation	
Synthesis Software Tools and Version	XST (M.53d)
Implementation Software Tools and Version	ISE software (M.53d), EDK (EDK_MS1.53d), SDK (EDK_MS1.53d)
Static Timing Analysis Performed?	Yes
Hardware Verification	
Hardware Verified?	Yes
Hardware Platform Used for Verification	Spartan-6 FPGA Consumer Video Kit

References

1. Spartan-6 FPGA Consumer Video Kit
   http://www.xilinx.com/products/devkits/TB-6S-CVK.htm
2. VESA DisplayPort Standard v1.2
   https://fs16.formsite.com/VESA/form608559305/secure_index.html
3. AMBA Protocol Specifications Document Set
   http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html
4. UG696, *LogiCORE IP DisplayPort Source Core v1.2 User Guide*.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
07/21/10	1.0	Initial Xilinx release.

Notice of Disclaimer

Xilinx is disclosing this Application Note to you "AS-IS" with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.