*User's Guide*

# *DLP DLPC964 Apps FPGA*


**ABSTRACT**

The AMD Xilinx™ Virtex™-7 VC-707 Apps FPGA provides functionality to a DLPC964 controller and supported DMD. The DLP® DLPC964 Apps FPGA is just one example of a front end board used to interface with the DLPLCRC964EVM, DLPLCR99EVM, and DLPLCR99UVEVM. The DLPC964 Apps FPGA user's guide details the functions and registers of the DLPC964 Applications FPGA (Apps FPGA) and the organization of the VHDL code used.

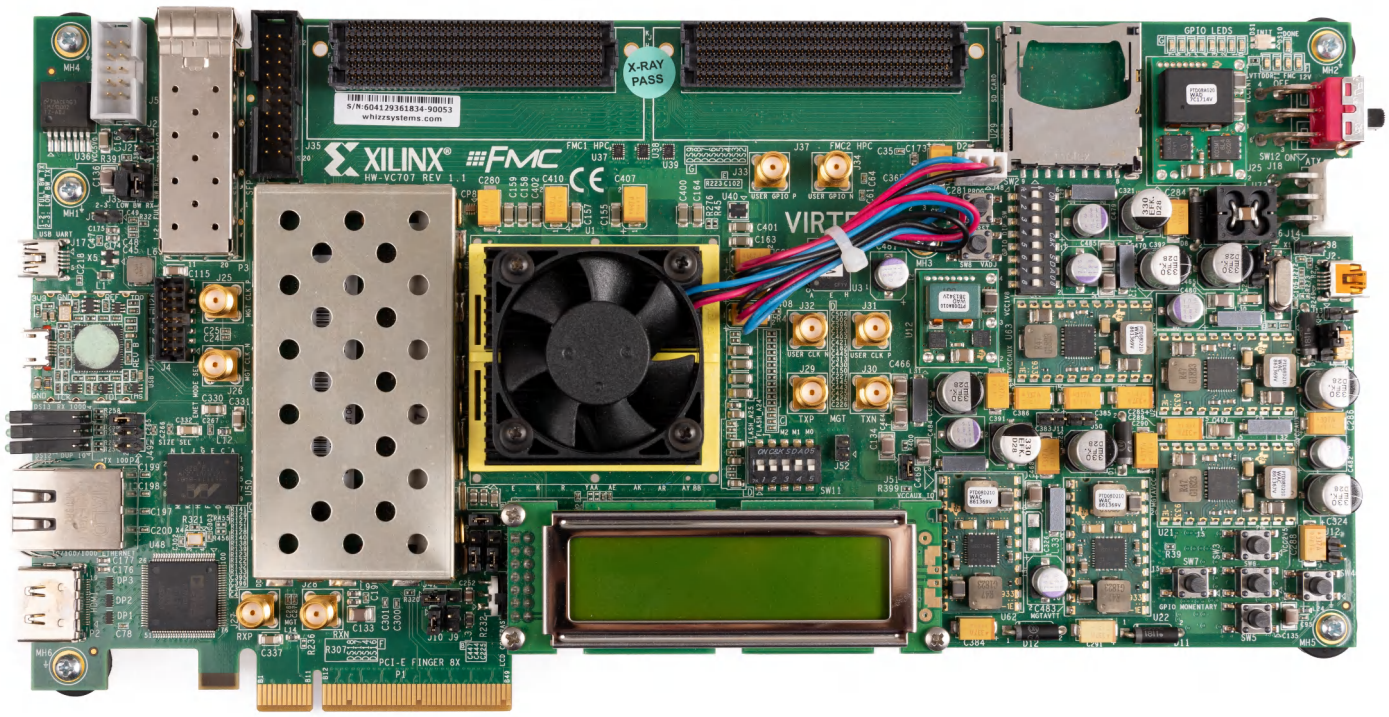




**Figure 1-1. AMD Xilinx Virtex-7 VC707 Evaluation Module**

# Table of Contents



# List of Figures

## List of Tables



## Trademarks

Xilinx™, Virtex™, and Vivado™ are trademarks of Xilinx, Inc.
DLP® is a registered trademark of Texas Instruments.
All trademarks are the property of their respective owners.

# 1 Overview

The DLPC964 Apps FPGA user's guide describes the functions and registers of the DLPC964 Applications FPGA (Apps FPGA) designed to work with a DLP LightCrafter DLPC964 EVM (DLPLCRC964EVM) and a supported DMD EVM (DLPLCR99EVM or DLPLCR99UVEVM). In addition, the guide provides an overview of the VHDL code and implementation.

**Note**

The DLPLCR99EVM, DLPLCR99UVEVM, DLPLCRC964EVM, AMD Xilinx VC-707 Evaluation board, power supplies, optics, and illumination source are sold separately.

## 1.1 Get Started

Please visit the DLPLCRC964EVM Tool Page, DLPLCR99EVM Tool Page, and DLPLCR99UVEVM Tool Page for more information regarding each EVM and the TI E2E DLP products forum for more assistance.

## 1.2 Features

The AMD Xilinx Virtex-7 VC-707 Evaluation kit is a front-end Evaluation Module for the the DLPLCRC964EVM, DLPLCR99EVM, and DLPLCR99UVEVM which consists of 32 HSSI input data lanes with up to 3.6 Gb/s per data lane.

## 1.3 Assumptions

The following document assumes that the user has run the DLPC964 Apps executable from ti.com. The RTL, Vivado™ project, and various scripts is located at: C:\Texas Instruments-DLP\DLPC964-Apps\.

## 1.4 Apps FPGA Hardware Target

The Apps FPGA reference code is targeted to the AMD Xilinx Virtex-7 FPGA housed on an AMD Xilinx VC-707 Evaluation Board. Figure 1-1 shows how the VC-707 Evaluation Board connects to the Texas Instruments (TI) DLPC964 Evaluation Module (DLPLCRC964EVM), which connects to the DLPLCR99EVM or DLPLCR99UVEVM.

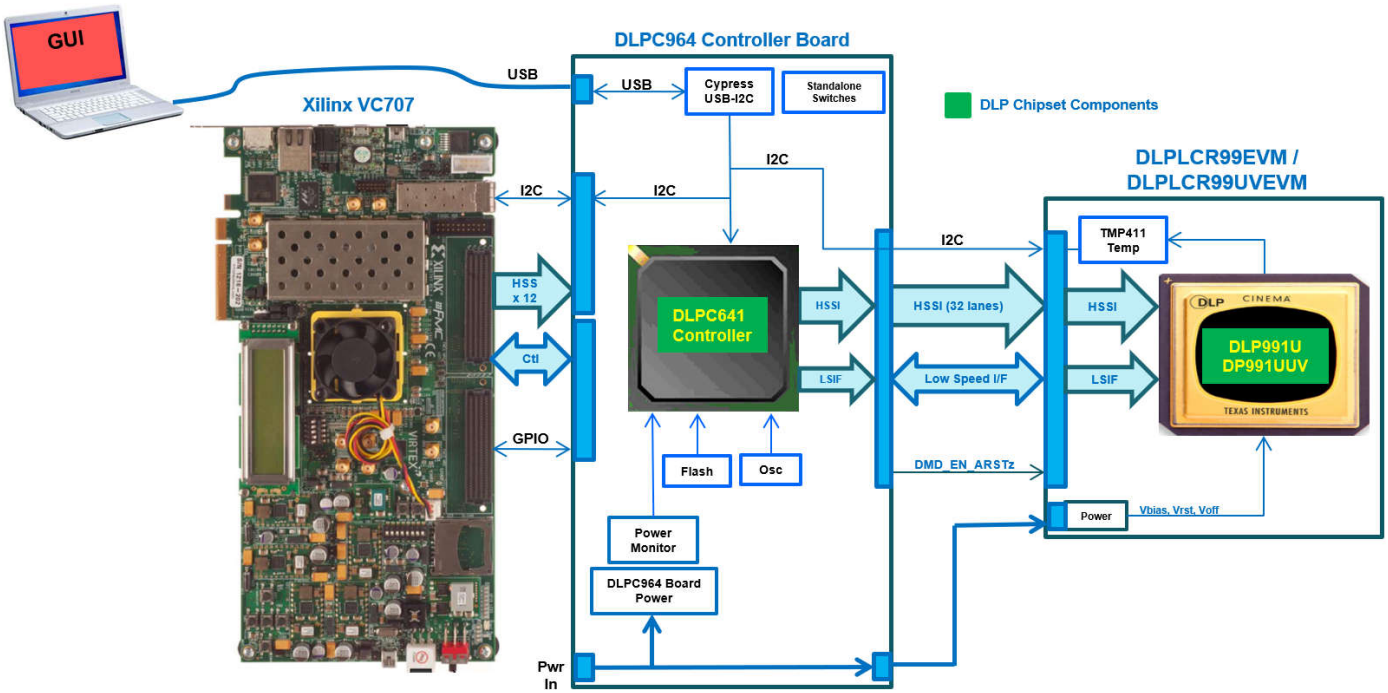



**Figure 1-1. Apps FPGA Hardware Target**

# 2 Apps FPGA Modules

This section details the various modules within the DLPC964 Apps FPGA.

## 2.1 Apps FPGA Block Diagram

Figure 2-1 shows the Apps FPGA Hardware Block diagram with varies modules. Each module plays an important role in trasmitting bitplanes to the DLPC964 controller. The DLPC964 recieves high-speed bit plane data from the external front end source (AMD Xilinx Virtex-7 VC-707) and formats the data prior to loading into a DLPLCR99EVM or DLPLCR99UVEVM for display on a DLP991U or DLP991UUV DMD.

The Bitplane Pattern Generator (BPG) is the main module when interfacing with the DLPC964 Apps FPGA and helps monitor the bitplane data being loaded from PGEN into the DLPC964 controller. The Block Reset Generator (BRG) helps start the PGEN data that was sent to the DLPC964 when the controller is not busy which is determined by the mcp_active signal coming from the DLPC964 controller.

Once the data is ready to be loaded into PGEN, the bitplane data is transmitted through HSSTOP, which is a wrapper for all four GTX channels (gtx0 - gtx3). Each channel helps trasmit the bitplane data to the DLPC964 controller with speeds up to 10Gbps for each channel. These modules are going to be explained in further detail below.

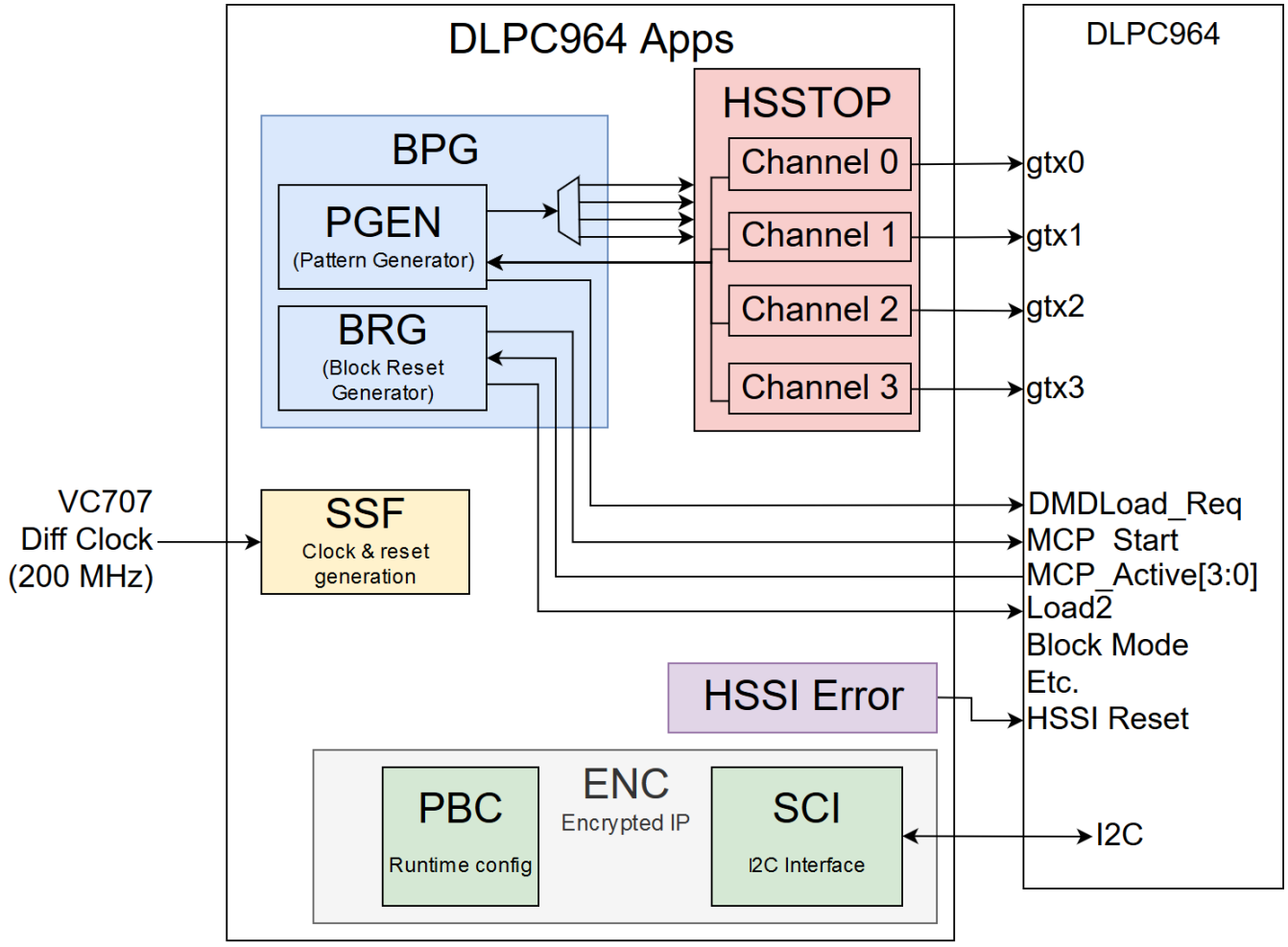



**Figure 2-1. Apps FPGA Hardware Block Diagram**

## 2.2 BPG Module

BPG (Bitplane Pattern Generator) is the main module of the DLPC964 Apps FPGA. This module can be used as an example on how to interface with the DLPC964 and Aurora transmit IP and consists of two main blocks:

1. The BRG (Block Reset Generator)
2. The PGEN (Pattern Generator)

The BPG acts as a wrapper for the two sub-blocks BRG and PGEN. The BRG sub-block is responsible for starting the PGEN and reporting when the DLPC964 is busy loading data.

**Note**

If data is being loaded into the DLPC964 controller, then the BRG waits to load more data from the PGEN until the mcp_active signal is low. Once this signal goes low, the mcp_start signal is sent to the DLPC964 indicating that more data can be loaded into the controller.

The PGEN reports when sending data to the Aurora GTX IP, the next block address that is going to be reset from the DLPC964 with the mcp_start signal, errors that occur (timeout or DMD HSSI), and settings chosen from the user.

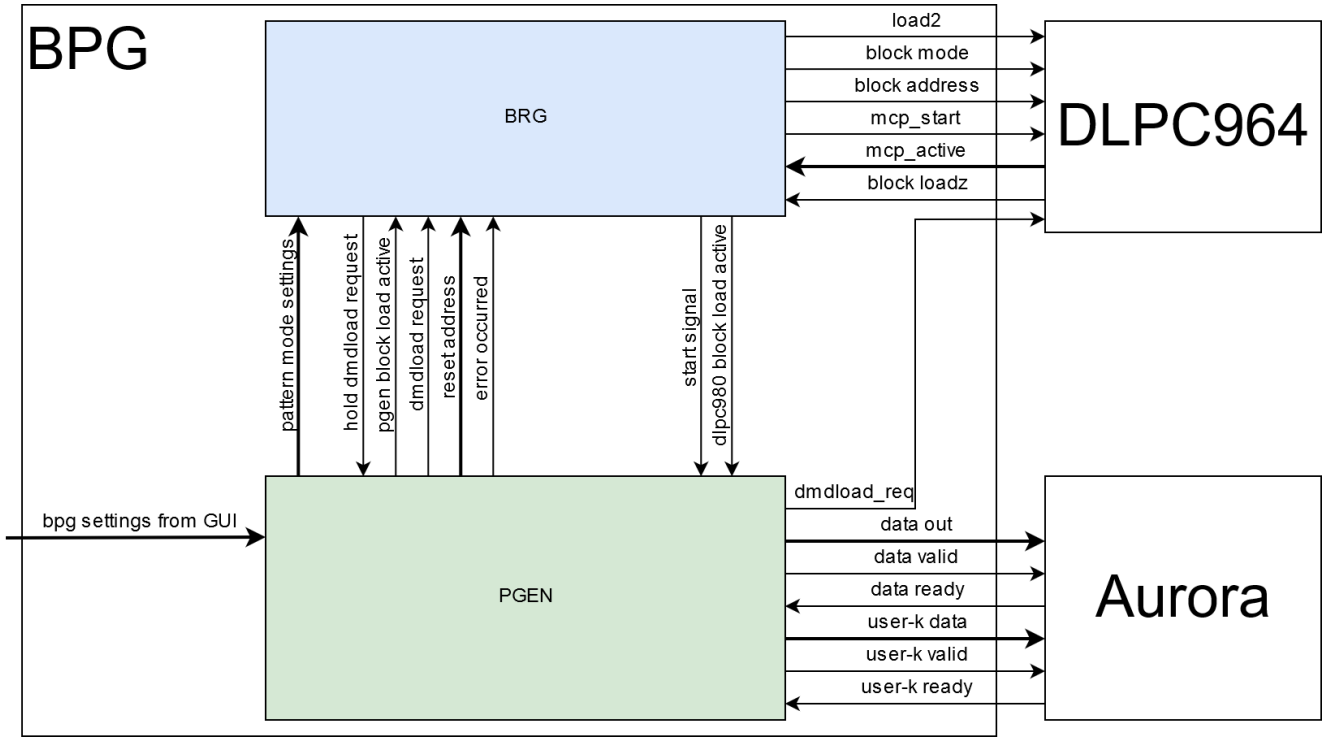



**Figure 2-2. BPG Module Hardware Block Diagram**

## 2.3 BRG Module

The BRG (Block Reset Generator) module is a sub module of the BPG. The BRG is responsible for starting the PGEN (Pattern Generator) and interfacing with the DLPC964. There are several logic processes within the BRG to help determine when the time to start the PGEN and when to send another MCP_Start to the DLPC964 controller.

To keep the block diagram simple, the various processes within the BRG have been stated as Logic modules. Each of these logic modules are shown in the Figure 2-3 and explained in more detail below.

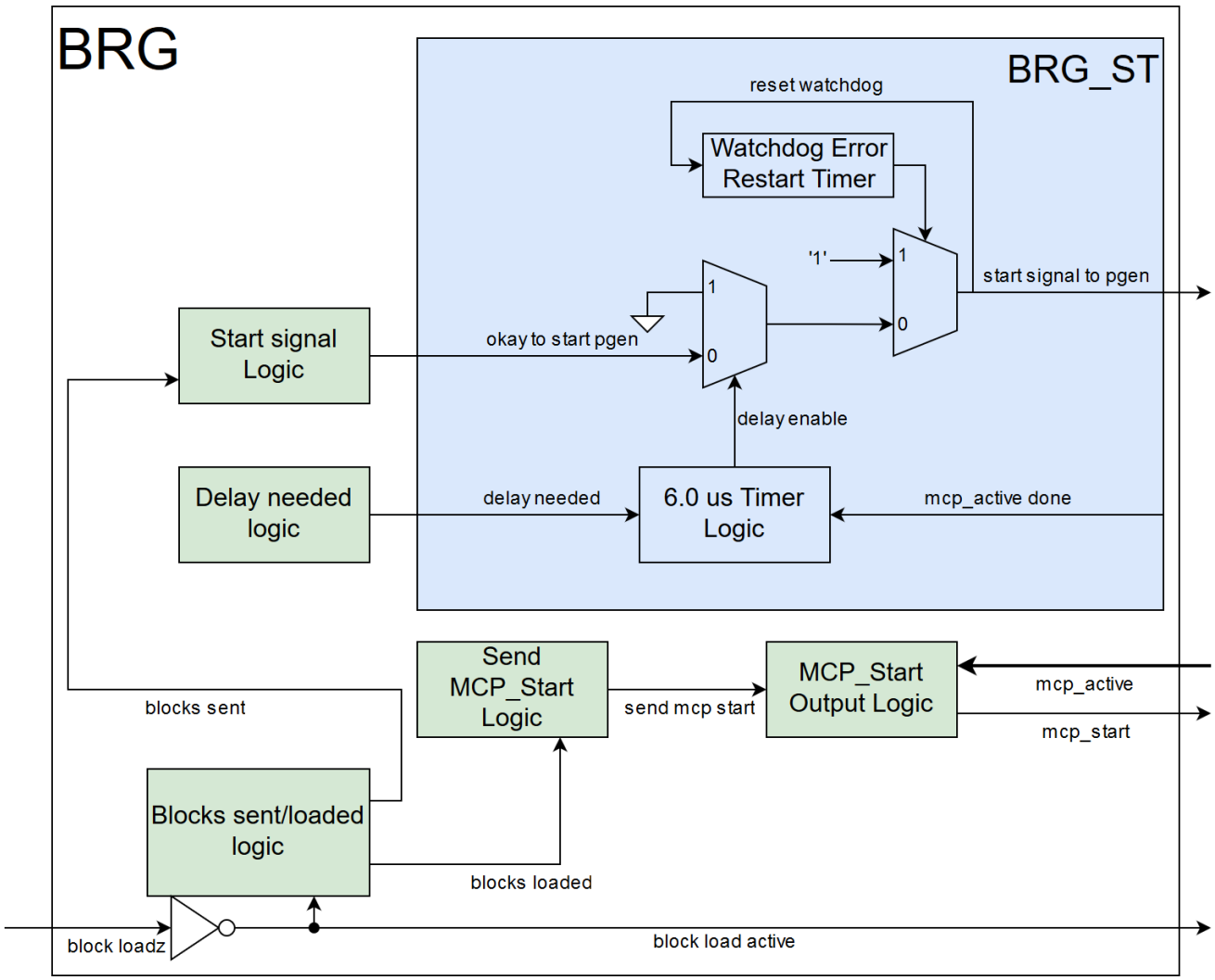



**Figure 2-3. BRG Module Hardware Block Diagram**

### *2.3.1 Start Signal Logic*

Start signal logic handles various scenarios that require the BRG to NOT start the PGEN.

Another block of data is not sent when the any of the following cases are met:

- The PGEN is already sending a block of data to the BRG.
- When all mcp_active signals are high. This signal from the DLPC964 lets the DLPC964 Apps FPGA know that the DLPC964 is busy resetting the DMD blocks and needs to finish before another block of data can be sent to the DMD blocks.
- When all enabled blocks have been loaded by the PGEN.

The PGEN has to wait for an mcp_start from the BRG before the user can send another data transfer to the DLPC964 controller.

### 2.3.2 Delay Needed Logic

The Delay Needed Logic handles when a potential mirror settle time violation occurs and tells the BRG_ST to hold the Start Signal Logic signal until the settle time is done.

The Mirror Settle Time refers to the following situation:

- Whenever a block is reset with an mcp_start signal because the time taken to update the DMD is communicated to the DLPC964 Apps via the mcp_active signal. Once the mcp_active signal goes low, the mirrors in that block are set in the proper state but still is settling into the state.

Loading data into mirrors that are settling can cause the DMD to go into an unknown state. To avoid this, there is a mirror settle time added to the delay logic, which is encountered in Global mode (as well as all other operating modes). This is because all blocks are loaded with data and issued with a mcp_start signal all at once. This means all the mirrors on the DMD need time to settle so the start signal need to be delayed.

**Note**

The DLPC964 Apps cover basic mirror settle time violations and delay the next load appropriately. To avoid complex logic and large amounts of test cases, the BRG adds a delay whenever any of the blocks are disabled.

### 2.3.3 Blocks Sent/Loaded Logic

The number of blocks sent to the DLPC964 and the number of blocks loaded by the DLPC964. This includes keeping track of segments sent when slow mode is enabled.

## 2.4 BRG_ST Module

This module is a sub-module of the BRG (Block Reset Generator) and is responsible for holding the PGEN start signal to avoid settle time issues. An important note about this module is the actual settle time delays used when the delay is requested.

When all blocks are enabled, the settle time delay is set to 6 us. When any one of the 16 blocks are disabled, the settle time delay is set to 6 us. The purpose of the first settle time delay is used in global mode. Since in global mode, all the blocks receive a reset signal, the DLPC964 apps cannot load another block until the mirrors have been given time to settle. The second settle time is used to avoid many of the potential settle time violations.

One example of this situation is if the DLPC964 Apps only had two blocks enabled and was set in single mode. The timing diagram is shown below.

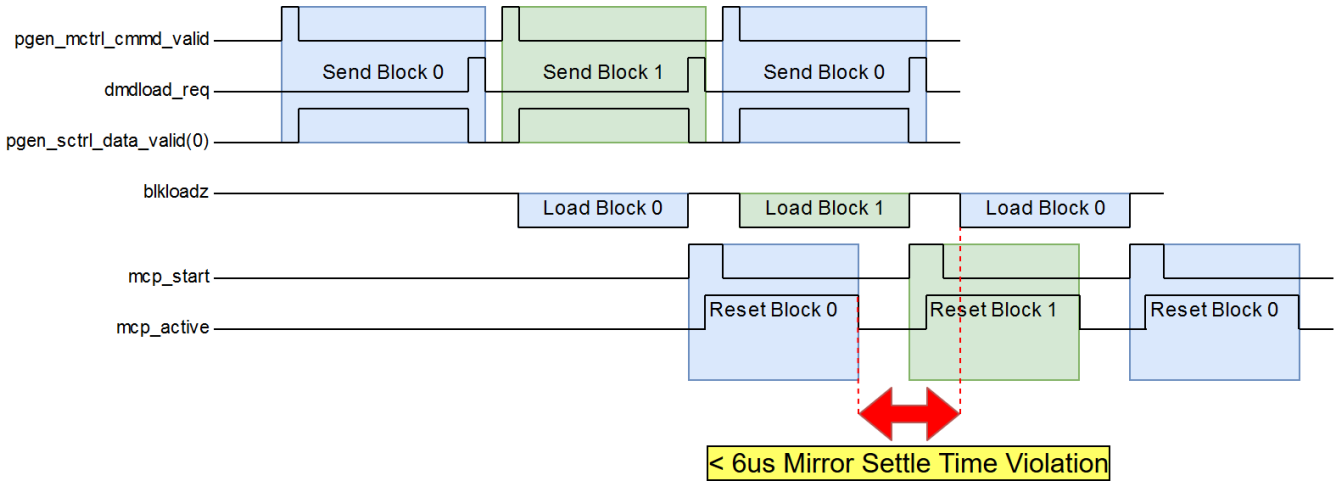



**Figure 2-4. BRG_ST Timing**

## 2.5 PGEN Module

The PGEN (Pattern Generator) module is a sub module of the BPG. The PGEN is responsible for reporting when data is being sent to the Aurora GTX IP, the next block address that is going to be reset from the DLPC964 with the mcp_start signal, errors that occur (timeout or DMD HSSI), and settings chosen from the user. The PGEN modules are shown in Figure 2-5 and are explained in more detail below.

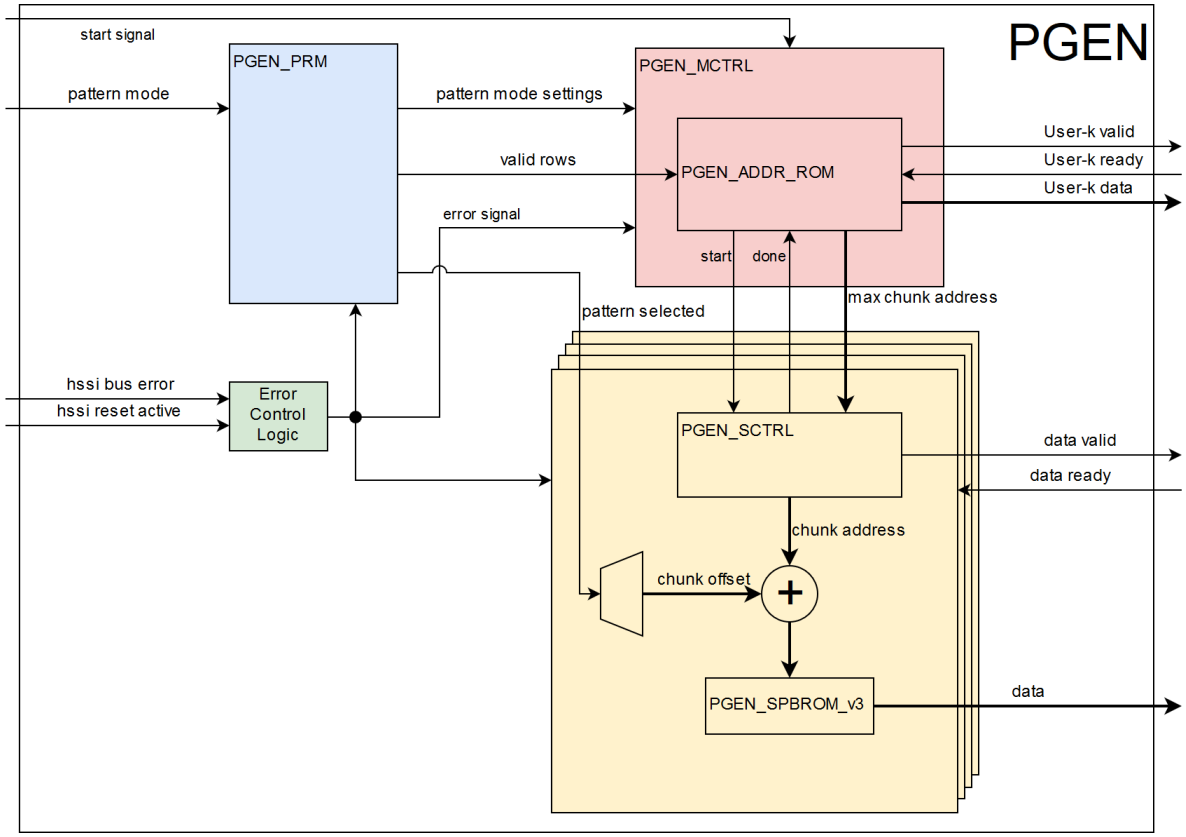



**Figure 2-5. PGEN Module**

## 2.6 PGEN_MCTRL Module

The primary control module is started by the start signal of the BRG, which also controls four copies of the secondary control modules. The secondary control modules are in charge of ROM addressing and outputting biplane images to the DLPC964 controller. The primary control module's core is a Finite State Machine (FSM) that starts off by waiting for the BRG to send the mcp_start signal. Once the signal is sent to the primary module, the FSM initiates. Figure 2-6 depicts the PGEN_MCTRL FSM where each state machine is defined as followed:

- **CV_SEND_SIGNAL** - Initial FSM state. This transitions states when the BRG start signal is received. Depending on the load type selected, the FSM goes to the CV_WAIT_FOR_ACK state or the CV_SEND_DMDLD state. The command valid signal is not needed when the block load type is Clear (001) or Set (001) because no data is sent during these load types.
- **CV_WAIT_FOR_ACK** - Once the FSM is started by the BRG, the command valid signal is sent to the Aurora user-k interface. The user-k valid signal is held high in this state until the Aurora user-k ready signal acknowledges the user-k data. Once the FSM is acknowledged, the FSM de-asserts the user-k valid signal and goes to the next FSM state.
- **CV_SEND_DMDLD** - Now that the command has been sent, the DLPC964 Apps FPGA can start sending bitplane data. This state starts and monitors all four of the secondary control modules. Once all four of the secondary modules report that the modules have completed sending data, the primary control modules can begin to send the DMD load signal over the Aurora user-k interface.
- **CV_HOLD_DMDLD** - The primary control module holds the DMD load signal until for approximately 0.80ns before transitioning to the beginning of the FSM.

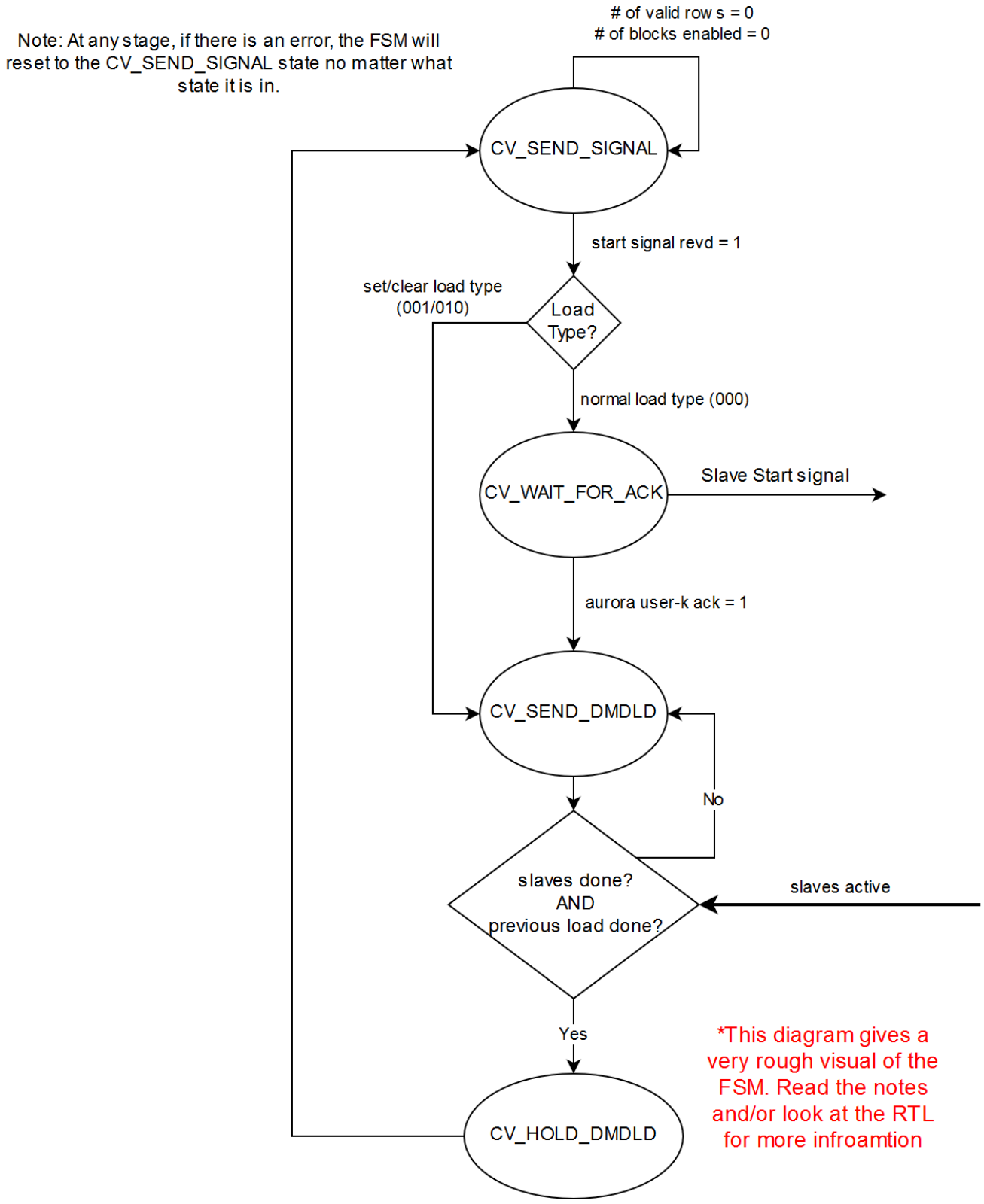



**Figure 2-6. PGEN_MCTRL FSM**

## 2.7 PGEN_SCTRL Module

There are four copies of the secondary control module that are controlled by the primary control module. Each secondary is responsible for sending out the proper length data valid signal and to increment the ROM address. The data valid signal goes to the Aurora interface to mark the data being sent as valid. The Aurora interface can de-assert the ready signal at various times, so the secondary module must take this into account by holding the values and valid signal until the ready signal is re-asserted. The maximum ROM address is sent to the secondary module by the primary control module. The secondary module counts up to this value allowing the ROMs to send out lines up to the user-specified amount. Figure 2-7 depicts the main Secondary FSM where each state machine is defined as followed:

- **IV_IDLE** - When the secondary modules are not needed (for example, when load types clear or set are selected), the secondary module is held in this idle state. If data is needed (load type = Normal), then the FSM goes to the next state.
- **IV_BEGIN** - Wait until the primary module sends a start signal. Once received, the secondary goes to the next state. Otherwise, the FSM holds in this state until the FSM receives the start signal OR the load type is changed.
- **IV_START** - The FSM starts the process to send out the valid signal and the ROM addresses. The valid signal is delayed a few clock cycles to align the ROM output with the valid signal.
- **IV_ACTIVE** - Once the secondary module is started, the process that outputs ROM addresses continues to run until the module has reached the ROM address sen by the primary module. Once the value is reached, the FSM is signaled to go to beginning state.

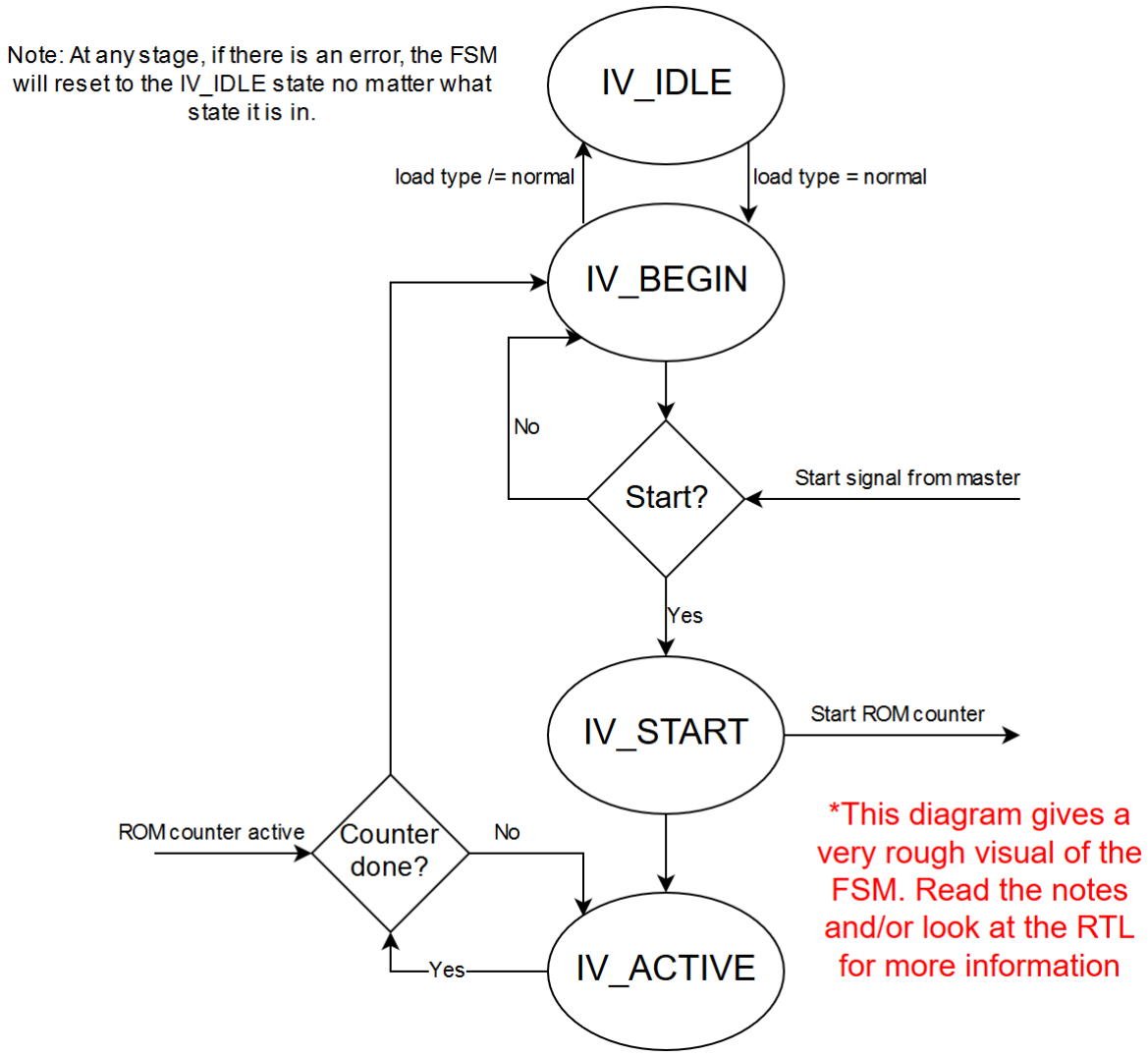



**Figure 2-7. PGEN_SCTRL FSM**

## 2.8 PGEN_PRM Module

The PGEN_PRM (Pattern Generator Parameter) module is responsible for cycling through the patterns, sending the Aurora user-k parameters, and selecting the next block to load/reset. Instead of testing and verifying potentially tens of thousands of test scenarios, the PGEN_PRM allows the user to select various configurations outlined in Section 3.2.3.

The user can enable and disable the pattern cycle timer via the $I^2C$ interface. When the timer is disabled, the user can select a single pattern to be displayed via the pattern select register. The patterns available to the user are outlined in Section 3.2.3. Certain user-k parameters (block addresses and segment numbers) change every time the PGEN is done sending a pattern. The block addresses that are to be loaded and reset are determined by the blocks enabled (pbc_bpg_blken) by the user.

## 2.9 PGEN_ADDR_ROM

Since a single line is the smallest loadable part of the DMD, the user specifies the number of lines desired to be loaded. However the Aurora GTX channel is 192 bits wide so the PGEN_ADDR_ROM translates the number of lines to be loaded into a ROM address using the following formula:

*MAX_ROM_ADDRESS = (# of lines) x 5 + CEIL(# of lines / 3)*

## 2.10 HSSTOP Module

This module contains the Xilinx Aurora IP for transmitting bitplane data to the DLPC964 controller board. This protocal is called the Aurora 64b/66b and for more information, go to Section 4.3.

As shown in Figue 2-8, the HSSTOP module has an AURPRA_APPS_TX_X12LN wrapper for all four of the GTX channels. Each Aurora GTX channel is comprised of three lanes with each lane transmitting 10Gbps. To help keep the GTX lanes synchronized, all four of the channels share the same Aurora clock module.

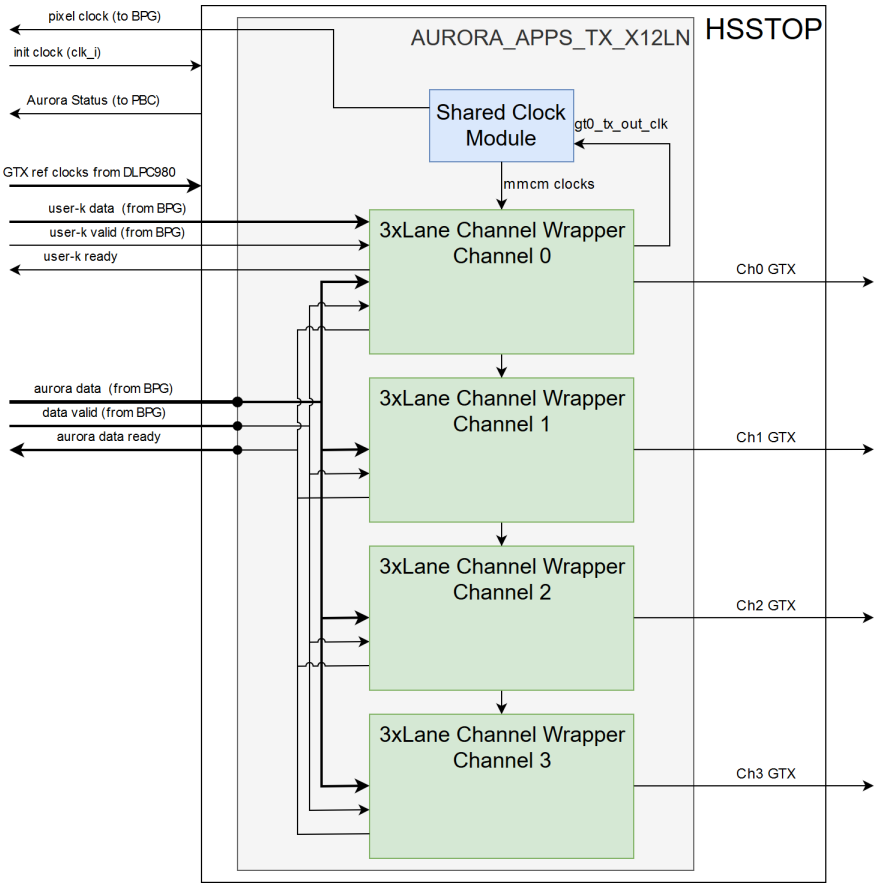



**Figure 2-8. HSSTOP Module Hardware Block Diagram**

The Aurora IP allows the differential signals to have a pre-emphasis and post-emphasis to help with signal integrity. In the DLPC964 Apps FPGA design, the following settings were used.

| Signal Name | Value |
|---|---|
| gt_txpostcursor_in | 0.00dB (00000) |
| gt_txdiffctrl_in | 807mV (1000) |
| gt_txmaincursor_in | 0.00dB (00000) |
| gt_txprecursor_in | 0.00dB (00000) |

## 2.11 SSF Module

The SSF is responsible for creating the DLPC964 Apps FPGA clocks and resets for the whole system. The SSF takes in the asynchronous signals like the push button reset, DLPC964 init done, and the Aurora MMCM lock so each signal can be synchronizes to the proper clock domains.

## 2.12 ENC Module

The ENC (Encrypted) module is just a wrapper for the IP that is not to be released to customers. The PBC and SCI modules are encapsulated by this module to keep all the encrypted IP in the same location.

## 2.13 Xilinx IP

Listed here are the Xilinx IP used in the DLPC964 Apps FPGA Design.

### *2.13.1 PGEN_SPBROM_v3*

This is a 192x192 ROM that holds various pattern data to be read out and sent over the Aurora interface. This IP can be re-programmed with different user patterns. See Section 3.2.5 below.

### *2.13.2 MAINPLL*

The PLL that generates two of the three main clock networks in the design. The clk_i (100MHz) clock used for the Aurora initialization clock. The clk_a (50MHz) is used for the configuration registers and $I^2C$ logic.

### *2.13.3 AURORA_APPS_TX_X3LN_CLOCK_MODULE*

The clock module uses the reference output clock from GTX channel 0 to generate the user and sync clocks. The generated clocks are sent to all four of the Aurora channels to verify that all four of the channels are aligned to the same user clock. Each channel can be slightly out of phase so the BPG architecture takes care of this with the primary/secondary architecture.

### *2.13.4 AURORA_APPS_TX_X3LN_CHANNEL_WRAPPER*

A channel wrapper consists of three GTX lanes and all the Aurora IP modules bundled together. There are four copies of this to make the four channels.

## 2.14 Reference Documents

Please refer to Section 6 for the Aurora 64B/66B v11.2 LogiCORE IP Product Guide.

## 2.15 DLPC964 Apps FPGA IO

| Signal Name | Input/ Output | Description |
|---|---|---|
| refclk_ui_p<br>refclk_ui_n | INPUT | Fixed 200MHz LVDS reference clock generated from DLPC964 Apps FPGA (Reference from VC-707: U51). |
| reset_ui | INPUT | Push button (Reference from VC-707: SW7) to reset the DLPC964 Apps FPGA. |
| irqz | INPUT | PBC Interrupt from DLPC964 Controller. |
| running | OUTPUT | Goes to LED0 (Reference from VC-707 GPIO_LED_0) on DLPC964 Apps FPGA to signal when out of reset. |
| C964_init_done | INPUT | Input from the DLPC964 that tells the DLPC964 Apps FPGA to be pulled out of reset. |
| wdt_enablez | OUTPUT | Watchdog Timer set to '1' when in operation |
| rxlpmen | OUTPUT | Set to 0 for low power mode equalization. Refer to the Xilinx App note for more information. |
| ext_hssi_rst | OUTPUT | Signal that resets the DLPC964 HSSI Interface. |
| hssi_bus_err | INPUT | From DLPC964 that signals there was a sync error when loading last block onto DLPC964. |
| hssi_rst_act | INPUT | From DLPC964 to tell Apps DLPC964 the HSSI is being reset |
| load2 | OUTPUT | Used during DLPC964 init process to setup DMD in load2 mode. |
| blkmode[1:0] | OUTPUT | Used during DLPC964 init process to setup DMD superblock mode. |
| blkaddr[4:0] | OUTPUT | Block (or superblock) address the issued mcp_start is sent. |
| mcp_start | OUTPUT | Signals the DLPC964 to load whatever data was sent onto the DMD. |
| mcp_active[3:0] | INPUT | From the DLPC964 to signal when the DMD is loading data onto the DMD. Only 4 loads can happen at once. |
| blkloadz | INPUT | From the DLPC964 to signal when the block data sent is done being loaded and ready to be sent to the DMD. |
| dmdload_req | OUTPUT | Signals the DLPC964 to load the block recently sent into the controller into the DMD. |
| gtrx_ch0_refclk_p/n<br>gtrx_ch1_refclk_p/n<br>gtrx_ch2_refclk_p/n<br>gtrx_ch3_refclk_p/n | INPUT | Reference clock from the DLPC964 for each of the Aurora Transmit channels (GTX Channel 0 - 3). |
| ch0_gtx_p/n[2:0] | OUTPUT | Aurora 10Gbps Transmit channel 0.<br>User-k data is sent across channel 0 ONLY along with the data.<br>When slow mode is enabled (pbc_bpg_normal_mode_en = 0), channel 0 is the only channel sending data. |
| ch1_gtx_p/n[2:0] | OUTPUT | Aurora 10Gbps Transmit channel 1. |
| ch2_gtx_p/n[2:0] | OUTPUT | Aurora 10Gbps Transmit channel 2. |
| ch3_gtx_p/n[2:0] | OUTPUT | Aurora 10Gbps Transmit channel 3. |
| i2c_sda | INOUT | $I^2C$ Data line shared with the DLPC964. |
| i2c_scl | INOUT | $I^2C$ Clock line shared with DLPC964. |
| fmc_gpio[6:0] | INOUT | GPIO between the DLPC964 Apps FPGA and DLPC964. |
| led | OUTPUT | Goes to LED1 (Reference from VC-707 GPIO_LED_1) on DLPC964 Apps FPGA to signal when BPG is enabled. |
| testmux_uo[15:0] | INOUT | Debug mux for the DLPC964 Apps FPGA. |

## 2.16 Key Definitions

- **Block** - A block is a 136 lines x 4096 pixels portion of the DMD. The DMD is split into 16 of these blocks making the total image size of the DMD 2176 lines x 4096 pixels. Blocks can be individually addressed (0x0 - 0xF) in the DLPC964 Apps FPGA.
- **Segment** - There are four segments (A,B,C, and D) per block. Each segment consists of 136 lines x 1024 pixels. In normal operation, each of the 4 segments are loaded at the same time. In slow mode each segment are loaded individually.
- **Groups** - When in double or quad modes, blocks are updated in groups. These groups are outlined in the table below. Note the importance that the user either enables all blocks in a group or disabled all blocks in a group when in double or quad modes.

| **Block Load Address** | 0x0 | 0x1 | 0x2 | 0x3 | 0x4 | 0x5 | 0x6 | 0x7 | 0x8 | 0x9 | 0xA | 0xB | 0xC | 0xD | 0xE | 0xF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Double Mode Groups** | 0x0 | | 0x2 | | 0x4 | | 0x6 | | 0x8 | | 0xA | | 0xC | | 0xE | |
| **Quad Mode Groups** | 0x0 | | | | 0x4 | | | | 0x8 | | | | 0xC | | | |

Below are simple timing examples of sending blocks in x4 mode and global mode.

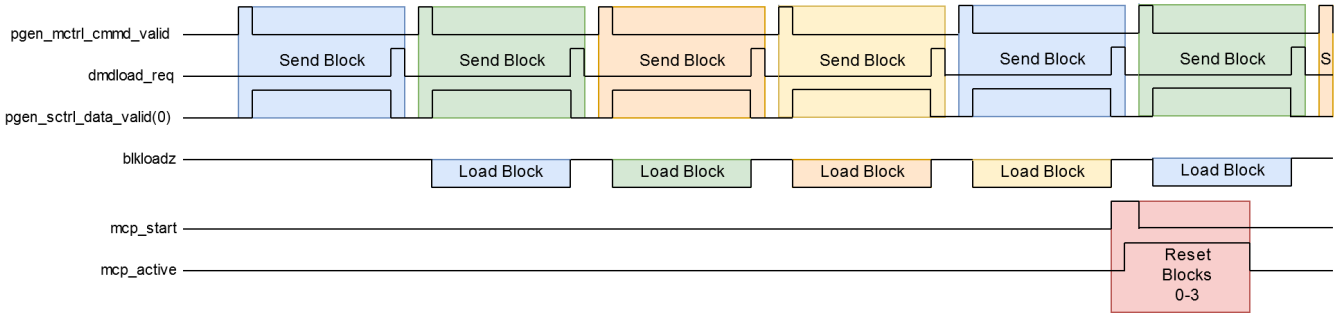



**Figure 2-9. x4 Mode**

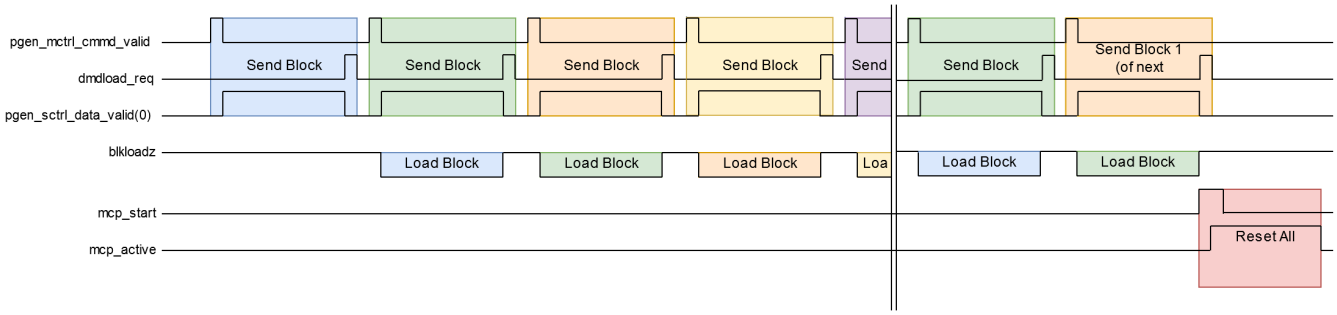



**Figure 2-10. Global Mode**

- **DLPC964 Apps** - The Xilinx VC707 loaded with the Apps FPGA bitstream.
- **DLPC964** - The DLPC964 Controller that interfaces with the DMD.
- **Modes** - There are 4 modes: Single (0x0), Double or x2 (0x1), Quad or x4 (0x2), and Global (0x3). These modes are explained in more detail below and in Section 3.
- **Single (0x0)** - In single mode, each block is loaded with data and once the DLPC964 has finished loading that individual block onto the DMD, the DMD is updated with the MCP_Start signal. Since each block can be updated individually, the valid block mode addresses are 0x0 - 0xF.
- **Double (0x1)** - Double mode means that once the DLPC964 has loaded 2 blocks within a group with data, the DMD updates both of these blocks with a single MCP_Start signal. Since 2 blocks are updated at a time, the valid block mode address are 0x0, 0x2, 0x4, 0x6, 0x8, 0xA, 0xC, and 0xE.

**Note**

Note the importance to know that when enabling and disabling blocks in double mode, all blocks within a group must be enabled or disabled.

- **Quad (0x2)** - Quad mode means that once the DLPC964 has loaded 4 blocks within a group with data, the DMD updates all 4 blocks with a single MCP_Start signal. Since 4 blocks are updated at a time, the valid block mode address are 0x0, 0x4, 0x8, and 0xC.
- **Global (0x3)** - This is the default startup mode. In global mode, all enabled blocks are loaded and once the DLPC964 has finished, the MCP_Start signal updates all the blocks at once. Since all the blocks have been updated at once, the next load operation needs to wait until the mirrors have settled.

**Note**

This is known as the mirror settle time and must be around 8 us.

- **Block Load Type** - The block load type is sent before the data using the user-k in the Aurora GTX interface. The DMD supports 3 different types of loads; Normal (0x0), Clear (0x1), and Set (0x2).
- **Normal (0x0)** - This is the default block load type. The Normal load type tells the DLPC964 to load the DMD with whatever data comes after the user-k data.
- **Clear (0x1)** - Clear load type does not send any data. This is because when the DLPC964 receives a clear load type, the DLPC964 sets the mirrors in the block specified to the off state (0).
- **Set (0x2)** - Set load type does not send any data. This is because when the DLPC964 receives the set load type, the DLPC964 sets the mirrors in the block specified to the on start (1).
- **MCP_Start** - The MCP_Start (Mirror Clocking Pulse Start) signals the DMD to update the mirrors with whatever data was sent. The DLPC964 determines which blocks to update based on the Mode selected and the block mode address.
- **Lines/Rows** - A line of pixels refers to the horizontal row of pixels 4096 across. Think of this as the y-position on the DMD.
- **Pixels/Columns** - A column of pixels refers to the vertical column of pixels 2176 across. Think of this as the x-position on the DMD.
- **Fast/Slow Mode** - Fast mode by default is enabled. Fast mode refers to sending all 4 segments of a block in parallel across the 4 Aurora GTX channels. Slow mode uses only the first GTX channel and sends the segments sequentially. Refer to the Pattern Modes section for more details on the segment ordering.
- **Load2 Mode** - Load2 is disabled by default. When in Load2 mode, the DLPC964 Apps only sends half of the number of rows specified (136 lines requested, only 68 are sent). This is because in this mode, the DMD loadS every 2 lines with the same data.

# 3 Functional Configuration

Upon startup, the DLPC964 Apps FPGA is in global reset mode, changing the pattern being sent to the DLP DMD EVM sent every 2 seconds. If the user wants to change pattern modes, then follow the steps listed below in Section 3.2.4.

## 3.1 Blocks Enabled

The default startup of the DLPC964 Apps FPGA enables all 16 DMD blocks. The register used to disable/enable the 16 blocks is a 16 bit configurable register. To avoid code complexity DMD timing violation issues, if a single block is disabled, then each subsequent DMD load is delayed. TI recommends to disable the BPG before modifying the number of blocks enabled.

**Note**

When in Double or Quad reset modes, either all the blocks in a reset group are enabled or all are disabled. For example, in Quad reset mode, only enabling blocks 0-3 and 8-11 is valid but enabling blocks 1-4 and 9-12 is not.

## 3.2 Pattern Cycle Enable

The default startup of the DLPC964 Apps FPGA cycles through the first eight predefined patterns approximately every 2 seconds. This can be toggled by the user. When disabled, a single selected pattern is sent to the DLPC964. Section 3.2.2 reviews what patterns are available to the user for display on the DLP DMD EVM.

### *3.2.1 North/South Flip*

Enabling the North/South Flip lets the DLPC964 to flip the image sent vertically.

### 3.2.2 TPG Patterns

When Pattern Cycle is enabled, patterns 1-8 is cycled through the DLPC964 controller. Patterns 9-14 are not cycled through, but can be selected by the customer.

**Table 3-1. TPG Patterns**

| Pattern Number | Value | Name | Description |
|---|---|---|---|
| 1 | 0x0 | Full-on | Full white background where all the mirrors on the DMD is in the on position. |
| 2 | 0x1 | Full-off | Full black background where all the mirrors on the DMD is in the off position. |
| 3 | 0x2 | Checkerboard | A black and white checkerboard pattern with squares 64 pixels long by 68 lines high. The height and width were chosen to be a repeatable 136 x 1024 image. |
| 4 | 0x3 | Single pixel grid with border | A single pixel border surrounds every 136 x 1024 area with a grid pattern inscribed inside each one. The vertical lines are spaced 32 pixels apart and the horizontal lines are spaced 34 lines apart. |
| 5 | 0x4 | West to East Diagonal Lines | Diagonal lines spanning west to east of each segment. |
| 6 | 0x5 | East to West Diagonal Lines | Diagonal lines spanning east to west of each segment. |
| 7 | 0x6 | Horizontal Lines | 16 line wide horizontal lines. |
| 8 | 0x7 | Vertical Lines | 16 pixel wide vertical lines. |
| 9 | 0x8 | Load2 checkerboard | **DEBUG PATTERN** A black and white checkerboard pattern with squares 32 pixels by 34 line high. The pattern continues between lines 0-67. Lines 68-135 are all black. This is to easily show how the load2 operation works. |
| 10 | 0x9 | Dots 10 by 10 | **DEBUG PATTERN** Customer requested pattern where single white pixels are spaced 8 pixels evenly in the X and Y direction. |
| 11 | 0xA | Inverting Checkerboard | **DEBUG PATTERN** This is an inverted version of the checkerboard pattern (0x2). When the user selects this pattern (0xA), the pattern timer register causes the BPG to flip between this pattern and the original checkerboard pattern (0x2). This is to help with hinge memory issues and must be used whenever the light source is off. |
| 12 | 0xB | Random Noise Pattern | **DEBUG PATTERN** Randomized noise pattern for customer tilt angle testing. |
| 13 | 0xC | 1x1 Horizontal Lines | **DEBUG PATTERN** Every row alternates between black/white and can be used to check for issues with row loads. |
| 14 | 0xD | 1x1 Vertical Lines | **DEBUG PATTERN** Every column alternates between black/white and can be used to check for issues with data bus lines. |
| 15 | 0xE | Full on/off | **DEBUG PATTERN** Selecting this pattern causes the BPG to toggle between full-on (0x0) and full-off (0x1) patterns based on the pattern timer value. |

### 3.2.3 Pattern Mode

**Note**

When changing pattern mode, follow the steps in Section 3.2.4 below

The pattern mode register allows the user to experiment with the various DLPC964 modes of operation. The table below goes over all the available pattern modes:

| Mode Number | Value | Name | Settings | Notes |
|---|---|---|---|---|
| 1 | 0x0 | Global Mode | • Global Reset Mode (0x3)<br>• Normal Load Type (0x0)<br>• Load2 Disabled (0x0)<br>• Fast Mode Enabled (0x1)<br>• Total Rows Loaded (136 = 0x88) | In global reset mode, all enabled blocks are loaded with data sequentially. Once all blocks have been loaded, the MCP_Start signal resets all the blocks at the same time. |
| 2 | 0x1 | Quad Mode | • Quad Reset Mode (0x2)<br>• Normal Load Type (0x0)<br>• Load2 Disabled (0x0)<br>• Fast Mode Enabled (0x1)<br>• Total Rows Loaded (136 = 0x88) | In quad reset mode, 4 blocks are loaded sequentially. Once the 4 blocks in a group have been loaded, the MCP_Start signal issues a reset to the 4 blocks in that group at the same time.<br>**Note**<br>There are 4 "groups" of blocks in Quad reset mode. Blocks 0-3, 4-7, 8-11, and 12-15. All blocks in a group must be enabled or disabled. |
| 3 | 0x2 | Double Mode | • Double Reset Mode (0x1)<br>• Normal Load Type (0x0)<br>• Load2 Disabled (0x0)<br>• Fast Mode Enabled (0x1)<br>• Total Rows Loaded (136 = 0x88) | In double reset mode, 2 blocks are loaded sequentially. Once the 2 blocks in a group have been loaded, the MCP_Start signal issues a reset to the 2 blocks in that group at the same time.<br>**Note**<br>There are 8 "groups" of blocks in Double reset mode. Blocks 0-1, 2-3, 4-5, 6-7, 8-9, 10-11, 12-13, 14-15. All blocks in a group must be enabled or disabled. |
| 4 | 0x3 | Single Mode | • Single Reset Mode (0x0)<br>• Normal Load Type (0x0)<br>• Load2 Disabled (0x0)<br>• Fast Mode Enabled (0x1)<br>• Total Rows Loaded (136 = 0x88) | In single reset mode, a single block is loaded at a time and once the DLPC964 has loaded the DMD with the data sent, the MCP_Start signal resets that single block. |
| 5 | 0x4 | Global Clear Mode | • Global Reset Mode (0x3)<br>• Clear Load Type (0x1)<br>• Load2 Disabled (0x0)<br>• Fast Mode Enabled (0x1)<br>• Total Rows Loaded (136 = 0x88) | This mode shows how the Clear block load type is used in the DLPC964 system.<br>A clear load type does not require any data because the block puts all the mirrors in the off state (0). Because the clear load type does not have any data to be sent following, the command valid signal is not needed so only the dmd load signal is sent.<br>The MCP_Start signal follows the same pattern as Global Mode. |

| Mode Number | Value | Name | Settings | Notes |
|---|---|---|---|---|
| 6 | 0x5 | Global Set Mode | • Global Reset Mode (0x3)<br>• Set Load Type (0x2)<br>• Load2 Disabled (0x0)<br>• Fast Mode Enabled (0x1)<br>• Total Rows Loaded (136 = 0x88) | This mode shows how the Set block load type is used in the DLPC964 system.<br>A set load type does the opposite of the clear load type and also does not require any data.<br>The set load type sets all all the mirrors in the on state (1). Just like the clear load type, there is no need for the command valid signal, only the dmd load signal.<br>The MCP_Start signal follows the same pattern as Global Mode. |
| 7 | 0x6 | Global Load2 Mode | • Global Reset Mode (0x3)<br>• Normal Load Type (0x0)<br>• Load2 Enabled (0x1)<br>• Fast Mode Enabled (0x1)<br>• Total Rows Loaded (136 = 0x88) | Enabling the load2 operation tells the DMD to load 1 line of data received into 2 rows of the DMD.<br>The role of the DLPC964 Apps FPGA during a Load2 operation is to make sure that at most 68 lines are sent over the Aurora HSS channels and that the number of rows enabled in the user-k control parameter is halved as well. |
| 8 | 0x7 | Single Slow Mode | • Single Reset Mode (0x0)<br>• Normal Load Type (0x0)<br>• Load2 Disabled (0x0)<br>• Slow Mode Enabled (0x0)<br>• Total Rows Loaded (136 = 0x88) | Slow mode (or disabling the fast mode) causes the DLPC964 Apps FPGA to send data across a single channel only (4x 10Gbps lanes compared to 12x).<br>To do this, each segment of a block must be sent sequentially across 1 channel instead of parallel. The segments must be sent in the following order: D (0x3) → C (0x2) → B (0x1) → A (0x0). Once all 4 segments are sent, the MCP_Start signal can be issued.<br>The MCP_Start signal behaves the same as in Single Mode. |

### 3.2.4 Switching Modes

Follow the instructions below for the proper way to switch between pattern modes (Ex. Going from Global reset mode to double reset mode).

1. Turn off the BPG.
2. Change the DLPC964 Apps FPGA to the desired pattern mode.
3. Reset the DLPC964 and wait for the DLPC964 to come out of reset.
4. Turn on the BPG.

**Note**

Following these steps verifies that the hardware does not go into an unknown state.

### 3.2.5 Changing the BPG Patterns

Below are instructions to help users change the default patterns in the ROM for TPG selections.

1. Go into the directory C:\Texas Instruments\DLPC964-Apps\docs\patterns and verify that the user has Python 2.6 (or greater) installed.
2. Open the binary_to_coe.py file and read through the top comments. The patterns used in the DLPC964 Apps were generated using this script. Look at the `bit_fnames` list located near the top of the script.

```
#       RTL-defined pattern (full-on)           # Pattern 1   (0x0)
#       RTL-defined pattern (full-off)          # Pattern 2   (0x1)
bit_fnames = ["chkrbrd_136x1024.txt"          , # Pattern 3   (0x2)
              "grid_136x1024.txt"             , # Pattern 4   (0x3)
              "diag_e2w_136x1024.txt"         , # Pattern 5   (0x4)
              "diag_w2e_136x1024.txt"         , # Pattern 6   (0x5)
              "horiz_136x1024.txt"            , # Pattern 7   (0x6)
#       RTL-defined pattern (veritcal lines)    # Pattern 8   (0x7)
              "load2chkrbrd_136x1024.txt"     , # Pattern 9   (0x8)
              "invchkrbrd_136x1024.txt"       , # Pattern 10  (0x9)
              "dots8by8_136x1024.txt"         , # Pattern 11  (0xA)
              "rand_136x1024.txt"             , # Pattern 12  (0xB)
              "horiz1x1_136x1024.txt"         , # Pattern 13  (0xC)
              "vert1x1_136x1024.txt"          ] # Pattern 14  (0xD)
#       TBD                                     # Pattern 15  (0xE)
#       Full-on/off toggle                      # Pattern 16  (0xF)
```

3. The user can create a new .txt file and replace one of the names within the `bit_fnames` list.

**Note**

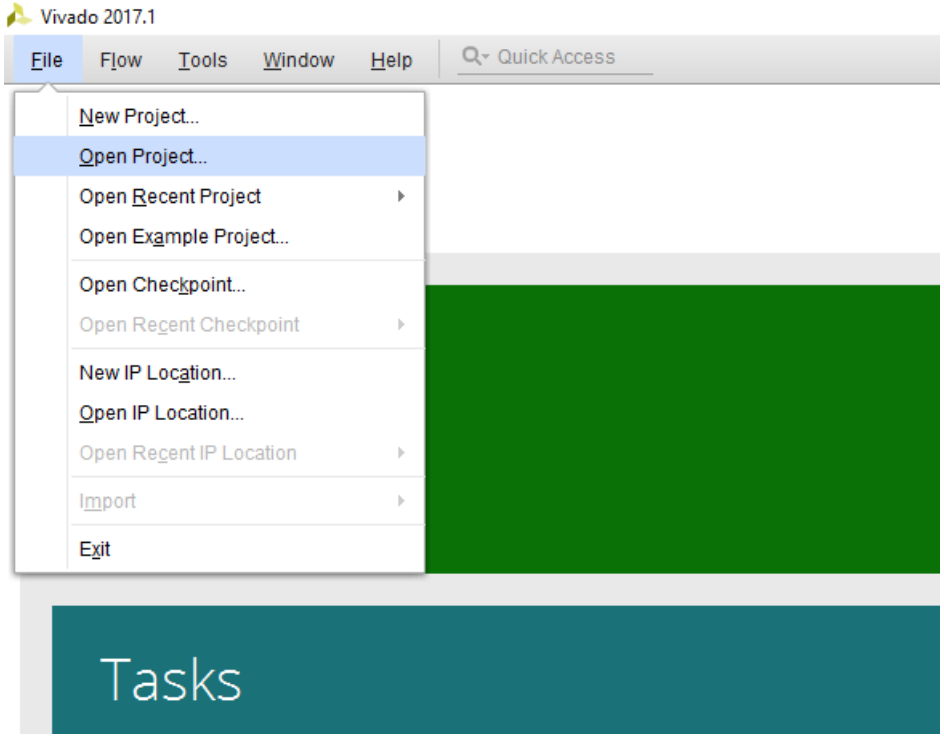
Any patterns designated RTL defined pattern cannot be changed to a different pattern since the patterns are not read from the ROM.

   a. Instructions for creating a .txt file for the python script:
      i. The text file MUST have 1024 columns and 136 lines.
      ii. Each character in the text file must either be a '1' or a '0'.
      iii. Make sure the text file is in the same directory as the python script.
4. Once the `bit_fnames` has been updated with the new text file name, run the python script. This creates a file called `bpg_patterns.coe`

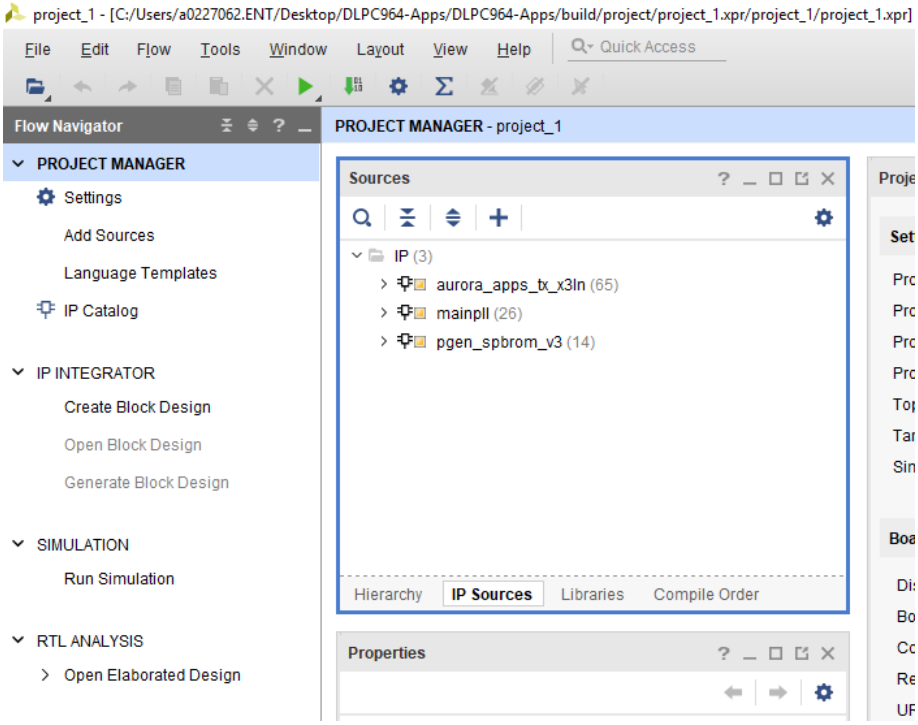
5. Open the Vivado project (either by unzipping the archived project in the build\project directory, or by running the run.tcl script).

**Note**

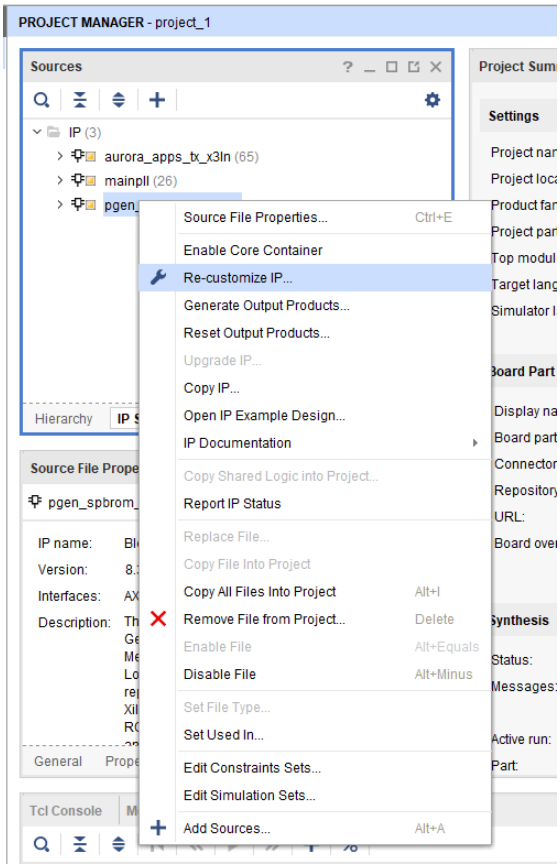
Unzipping the project is faster but if desired, then the run.tcl script has instructions on how to run.



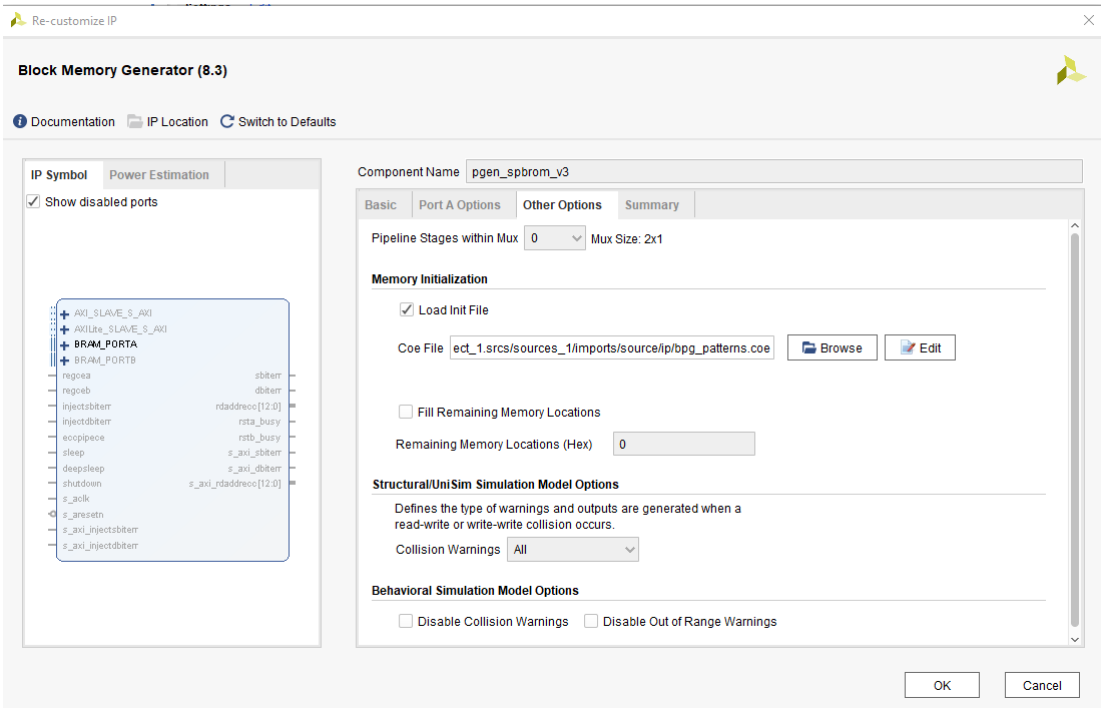

6. Once the project is open, in the Project Manager window, find the tab labeled IP sources and click.



7. Right-click on pgen_spbrom_v3 and select re-customize IP.

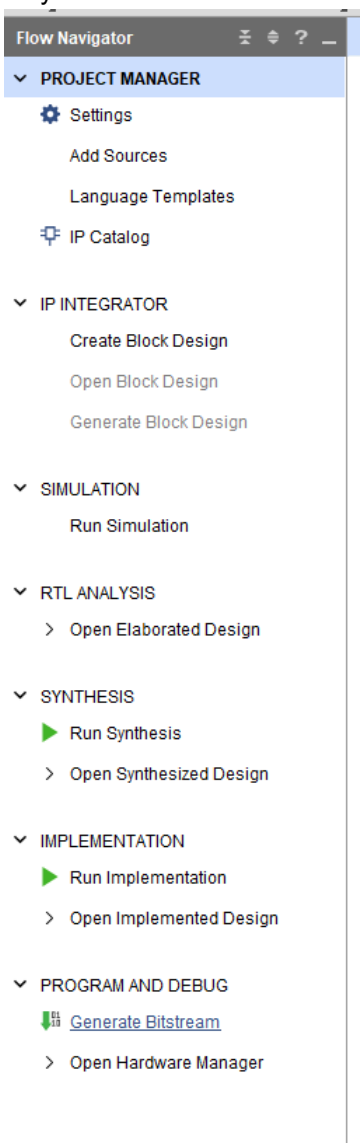



8. Once the IP configuration tool is open, go to the options tab and the user sees a *Memory Initialization* section.




9. Click *Browse* and navigate to the location of the bpg_patterns.coe file created by the python script in step 4. Assuming there are no errors, click *OK*. In the next window, click *Generate*.
10. The user now has reprogrammed the ROM in the DLPC964 Apps FPGA. Now, re-build the project.

11. Once Xilinx finishes generating output products, click *Generate Bitstream* located on the left in the Flow Navigator. Click *OK* to any prompts and once Vivado finishes, the bitstream can be found inside the project_1\project_1.runs\impl_1\ directory.



# 4 Appendix

## 4.1 Vivado Chipscope Captures

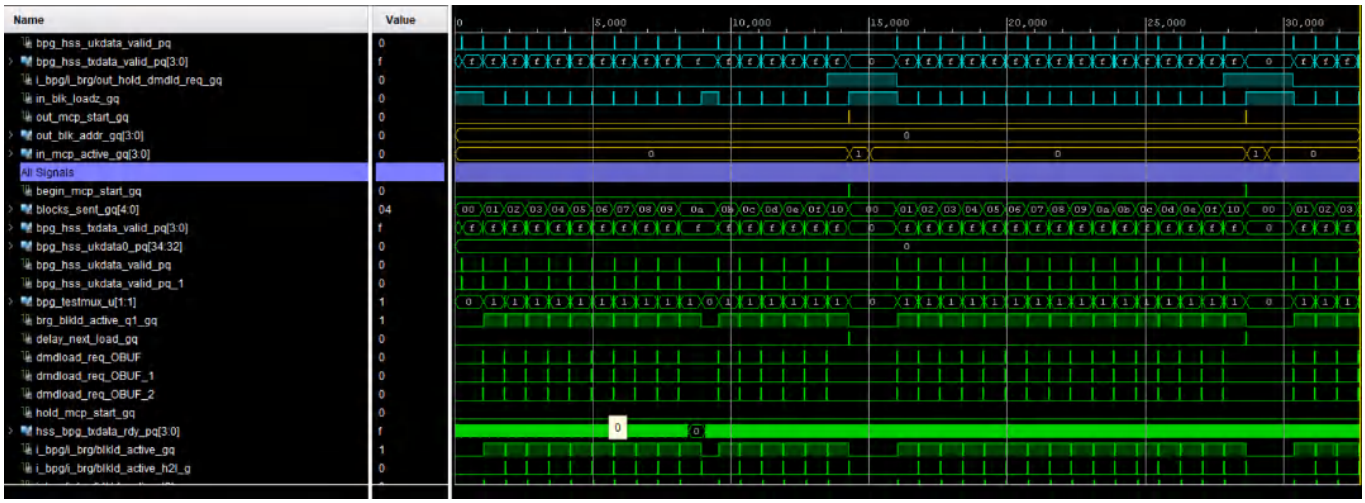



**Figure 4-1. Pattern Mode 0 Capture**

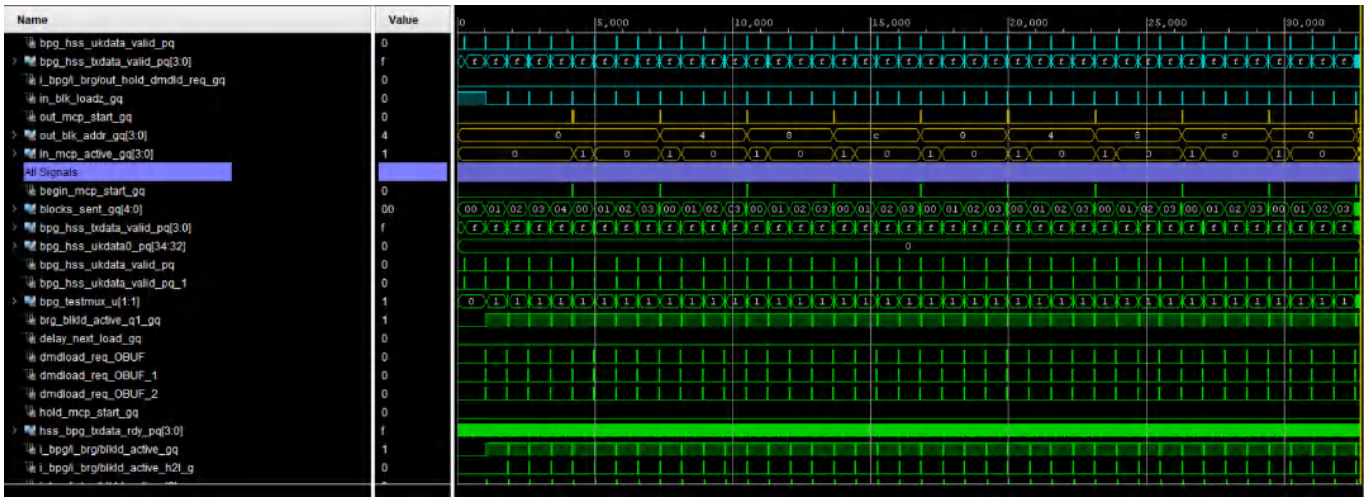



**Figure 4-2. Pattern Mode 1 Capture**

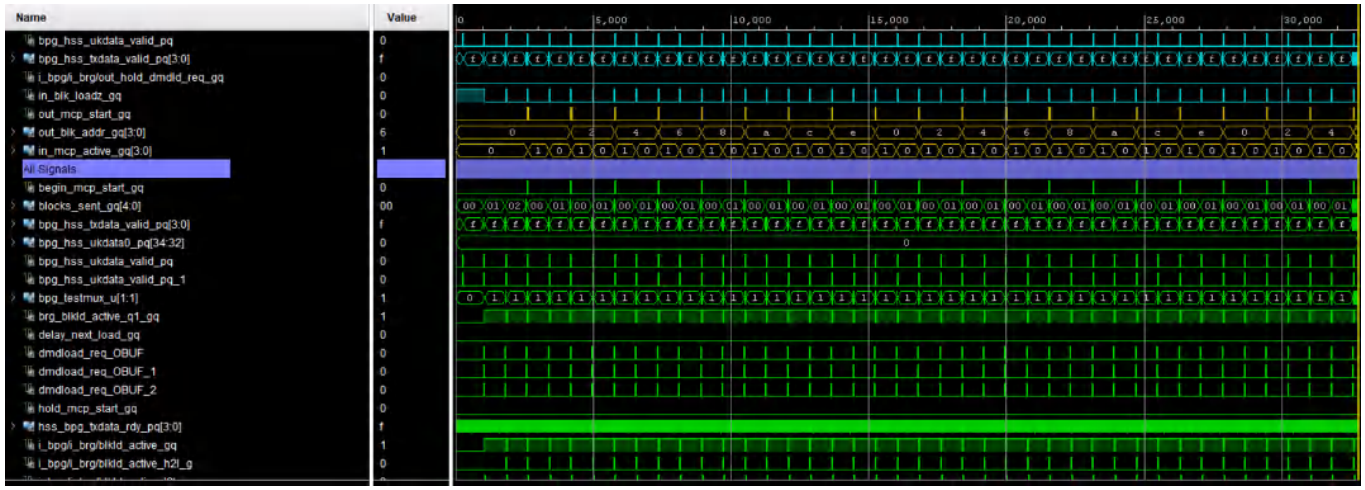



**Figure 4-3. Pattern Mode 2 Capture**

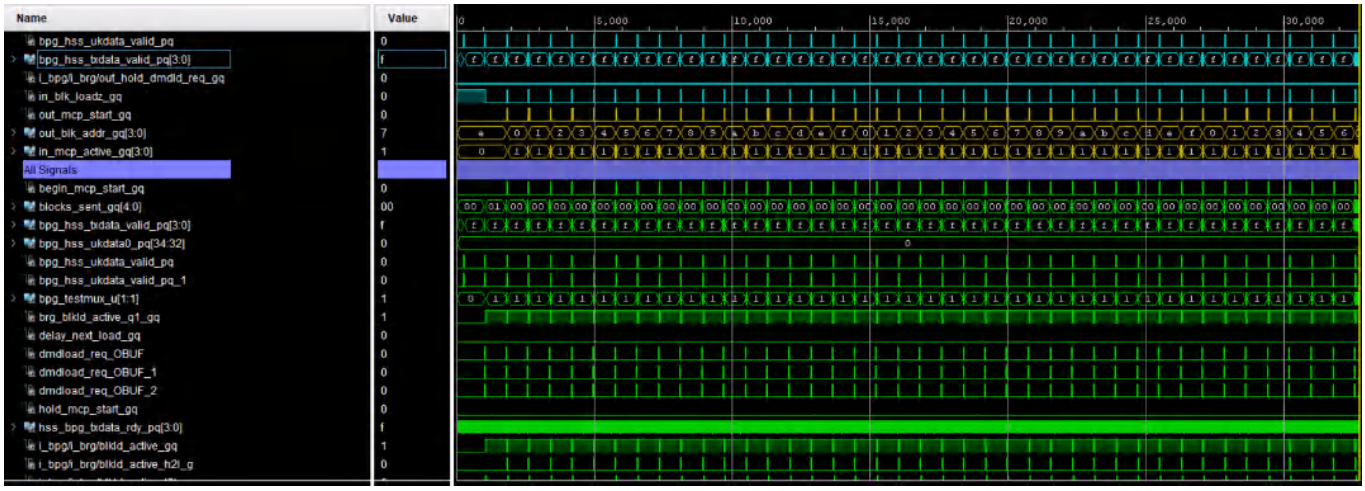



**Figure 4-4. Pattern Mode 3 Capture**


**Figure 4-5. Pattern Mode 4 Capture**


**Figure 4-6. Pattern Mode 5 Capture**


**Figure 4-7. Pattern Mode 6 Capture**

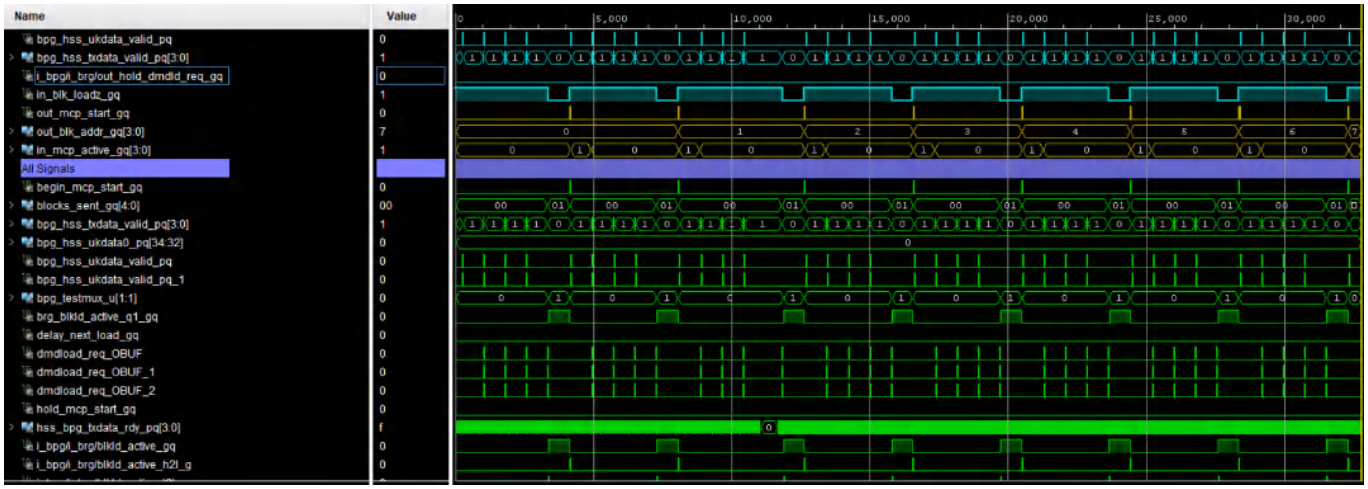



**Figure 4-8. Pattern Mode 7 Capture**

## 4.2 DLPC964 Apps Bitstream Loading

### *4.2.1 Loading Bitstream onto FPGA*

Follow the instructions below for loading the DLPC964 Apps binary onto the FPGA via a bitstream using Vivado Lab Solutions 2018.2.

**Note**

Click the link above to download Vivado Lab Solutions 2018.2. Once the webpage is loaded, find the archived 2018.2 folder and then navigate to the Vivado Lab Solutions 2018.2 downloadable link and download the installation.

**Note**

The FPGA needs to be reloaded each time power is lost/disconnected from the AMD EVM.

1. Plug in the Micro-B USB cable into the side of the VC707 and the other end into the computer running Vivado.
2. Start Vivado Lab Solutions 2018.2 on the computer.
3. Select *Open Hardware Manager* from the main window.
4. Click *open target* located in the top left of the hardware manager then *Auto Connect*.
   a. If the AMD EVM is the only FPGA plugged into the computer, then Vivado automatically connects to the AMD EVM.
5. Right-click on the *FPGA* and select *Program Device*.
6. Navigate to the appstop.mcs file and select *Program*.

### *4.2.2 Loading Bitstream onto Flash*

Follow the instructions below for loading the DLPC964 Apps binary onto the flash via a bitstream using Vivado Lab Solutions 2018.2.

**Note**

Click the link above to download Vivado Lab Solutions 2018.2. Once the webpage is loaded, find the archived 2018.2 folder and then navigate to the Vivado Lab Solutions 2018.2 downloadable link and download the installation.

**Note**

The bitstream is always loaded onto the FPGA upon power-up of the AMD EVM.

1. Plug in the micro USB into the side of the AMD EVM and the other end into the computer running Vivado.
2. Make sure to set SW11 to 00010 (1 = on, Position 1 → Position 5, left to right).

**Figure 4-9. FPGA Configuration Mode**

3. Set SW2 to 00000000 (1 = on, Position 1 → Position 8, left to right).


**Figure 4-10. GPIO Dip Switches (VC707)**

4. Start Vivado Lab Studios 2018.2 on the computer.
5. Select *Open Hardware Manager* from the main window.
6. Click *open target* located in the top left of the hardware manager then *Auto Connect*.
   a. If the AMD EVM is the only FPGA plugged into the computer, then Vivado automatically connects to the AMD EVM. Otherwise, the process is slightly more involved.
7. Right-click on the FPGA and select *Add Configuration Memory Device*.
8. Find the Flash name mt28gu01gaax1e-bpi-x16 and click *OK*.
9. Select *OK* again and select the configuration file (appstop.mcs).
   a. Make sure all other settings match.
10. Once setup, click *OK*. The programming can take a few minutes.
11. Once completed, power cycle the AMD EVM and the DLPC964 Apps Bitstream automatically loads onto the AMD EVM.

## 4.3 Interfacing To DLPC964 Controller with Aurora 64B/66B

### *4.3.1 Theory of Operation*

Data transactions to the DLPC964 controller is DMD block based. The DLP991U and DLP991UUV has a total of 16 DMD blocks where each block is 4096 columns by 136 rows each. As shown in Figure 4-11, a single row of DMD column is further divided into four segments of 1024 columns and mapped independently to the four Aurora serial channels. Therefore, each Aurora core is a full DMD block array of 1024 columns by 136 rows.

### *4.3.2 Overview*

Data transfer between the VC-707 Apps FPGA and DLPC964 Controller are carried out with twelve 10Gbps serial links as shown in Figure 4-11 below. The link-layer protocol is the Xilinx Aurora 64b/66b serial interface.

The Aurora 64b/66b serial interface covers:

- The generation of the Aurora 64b/66b TX core with Xilinx Vivado IP Catalog.
- Interface signaling between the RTL wrapper aurora_apps_tx_x12ln.v and Apps user logics.
- Theory of operation for transporting DMD data block with Aurora 64b/66b.
- Using Xilinx IBERT toolset to verify eye opening of the 10Gbps channel links.

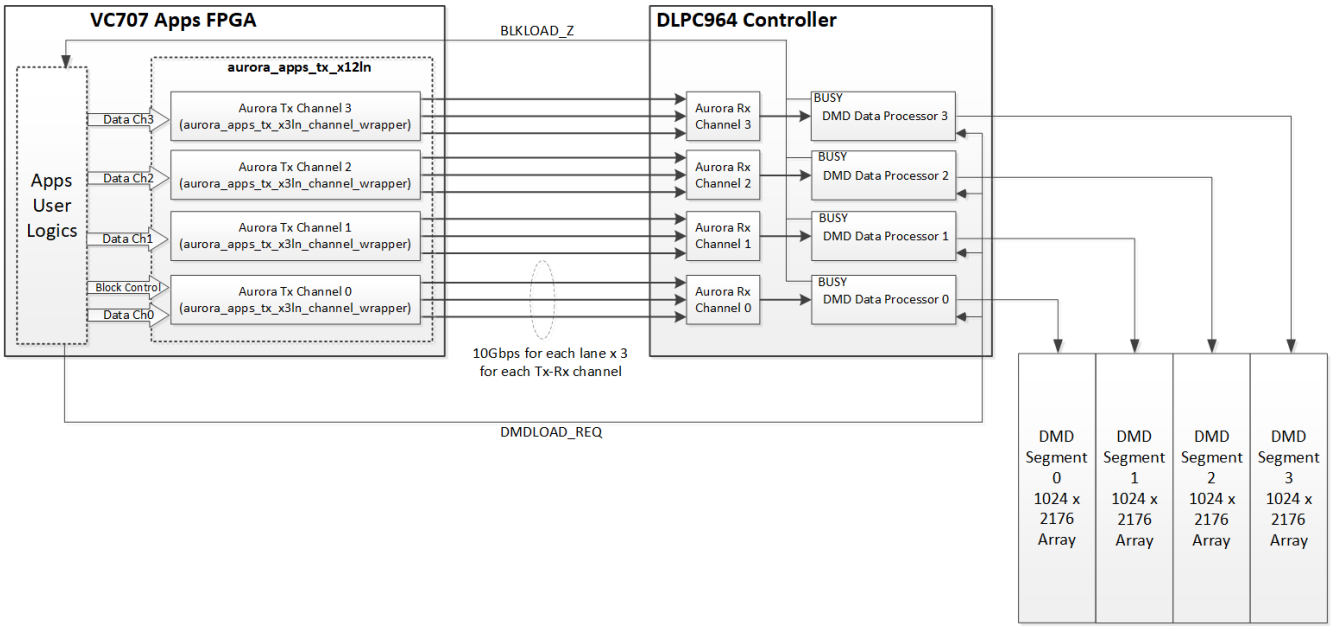



**Figure 4-11. DLPC964 System Block Diagram**

### *4.3.3 Aurora 64B/66B TX Core and RTL Generation*

TI released a RTL module called the aurora_apps_tx_x12ln.v for the VC-707 Apps FPGA reference project. This module contains four individual Aurora 64B/66B x3 lanes cores (aurora_apps_tx_x3ln_channel_wrapper.v) to manage traffic for each of the transmit channel. This section covers the generation step of the Aurora core using Xilinx Vivado 11.2 IP Catalog.

#### 4.3.3.1 Select Aurora 64B66B From IP Catalog

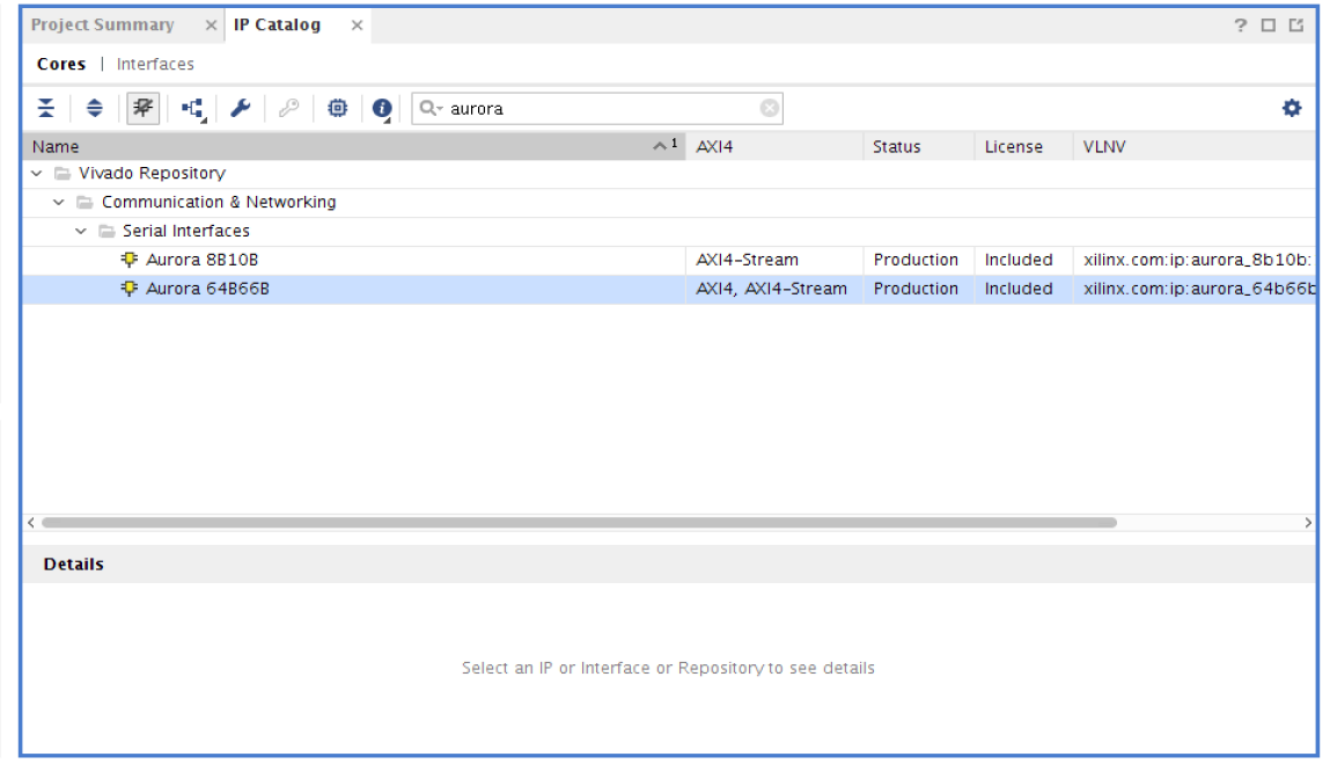



**Figure 4-12. Selecting from IP Catalog**

#### 4.3.3.2 Configure Core Options

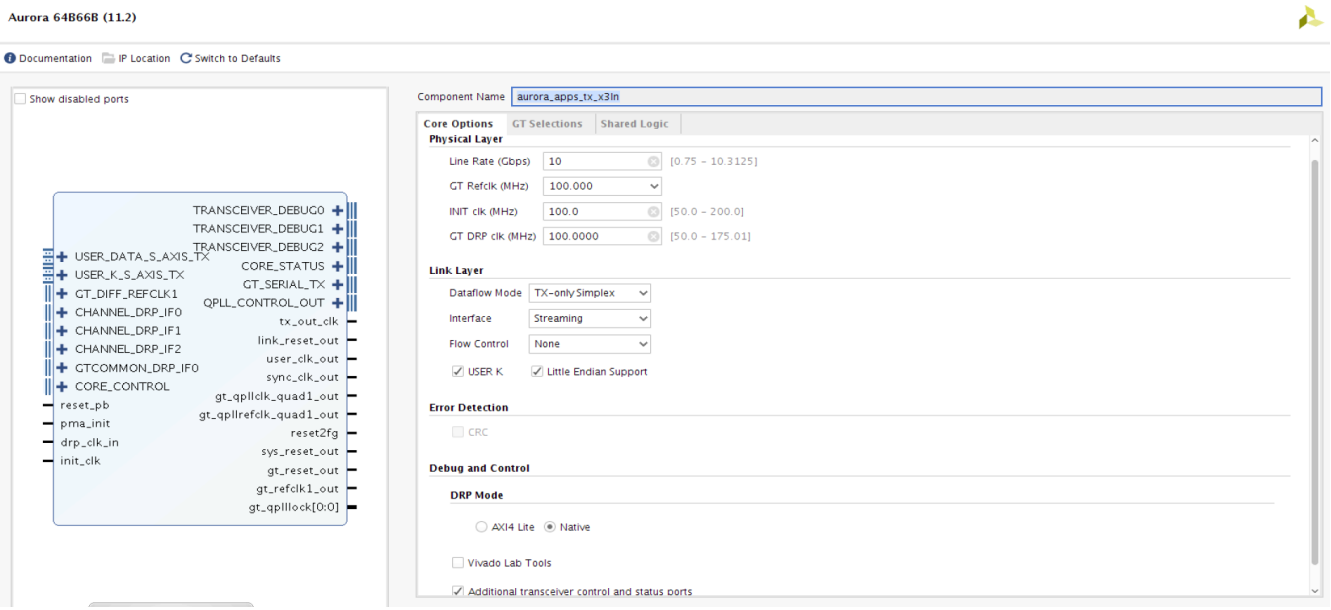



**Figure 4-13. Configuring Core Options**

#### 4.3.3.3 Lane Configurations

Select 3 lanes configuration under *GT Selections*.

**Note**

Although quad GTXQ0 is selected during this step, the RTL of the core does not have any location lock. This allows for instantiation of four times in our top level x12 lanes module (aurora_apps_tx_x12ln.v).

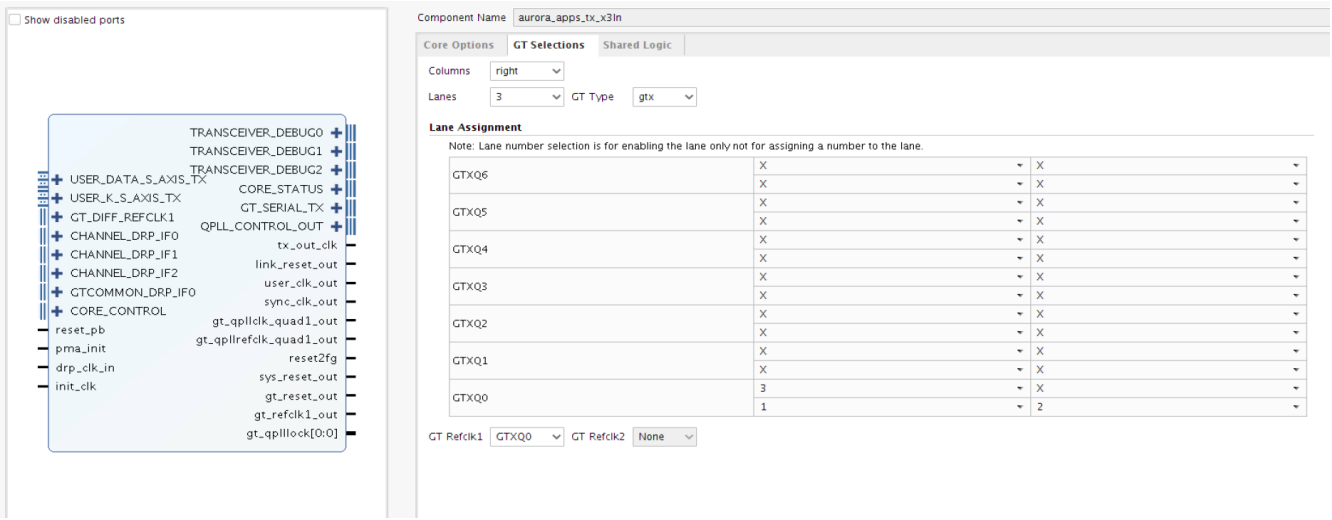



**Figure 4-14. Lane Configurations**

#### 4.3.3.4 Shared Logic Options

Select *Include Shared Logic in core* option under *Shared Logic*.

Proceed to generate the Aurora core.

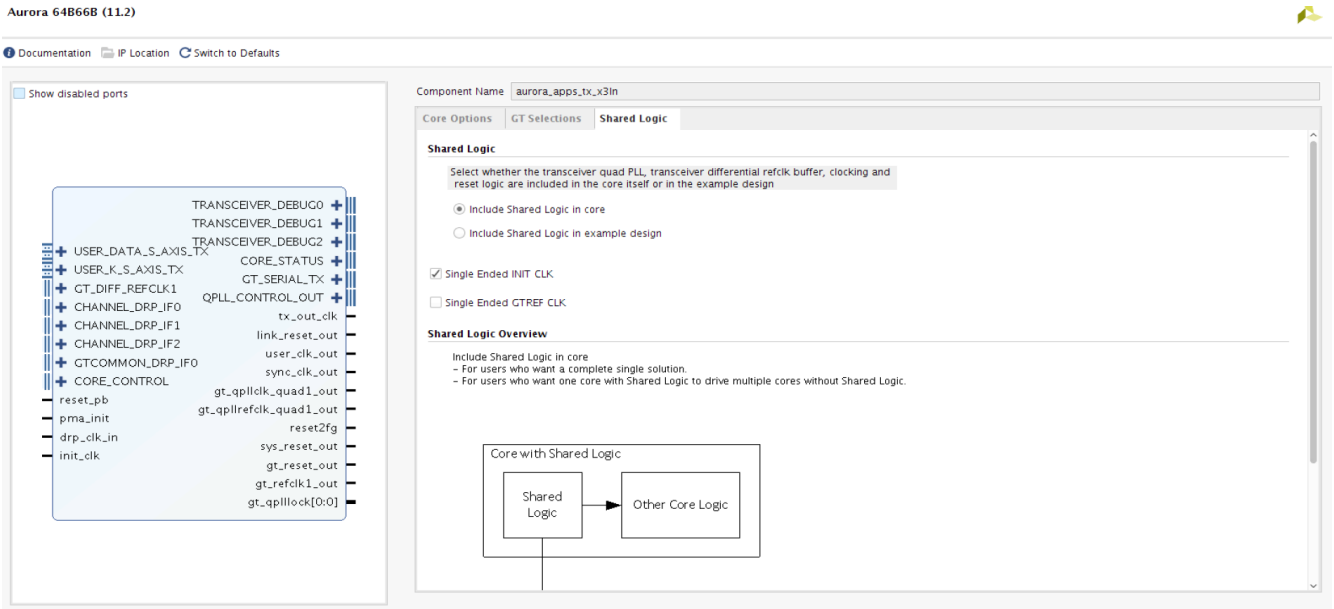



**Figure 4-15. Shared Logic Options**

#### 4.3.3.5 Generate Example Design Files

Users needs to find the generated core in the project's *Sources* section. Right-click and select the *Open IP Example Design* option to allow Vivado generate the example design.

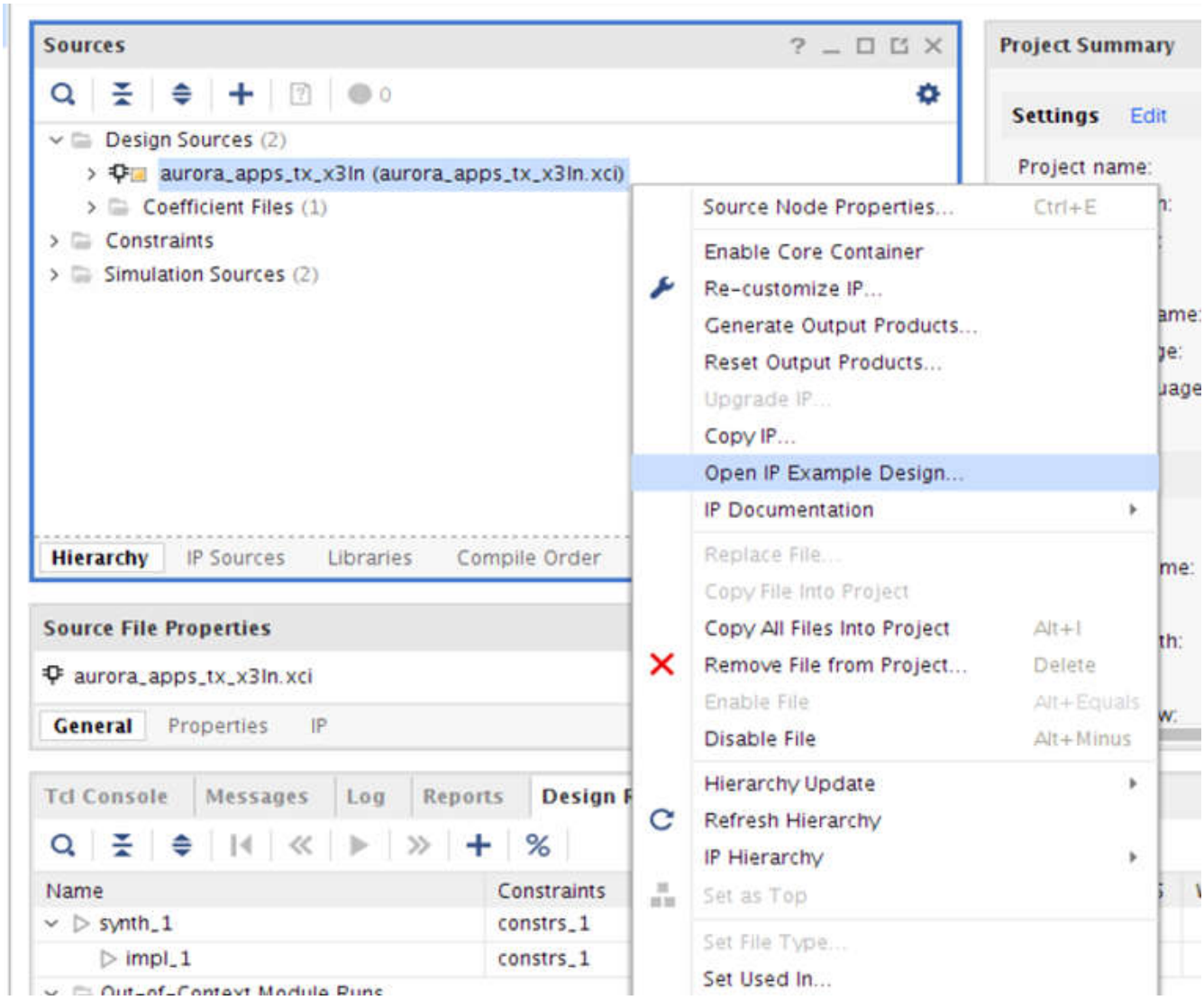



**Figure 4-16. Design File Generation**

#### 4.3.3.6 RTL File List

List of RTL files generated by Vivado IP Catalog shown below.

**Note**

PROJECT_DIRECTORY is the home directory of the user's Vivado project

The RTL example design files are located at the following directory:

**PROJECT_DIRECTORY/ip/aurora_apps_tx_x3ln/aurora_apps_tx_x3ln/example_design/gt**:

- aurora_apps_tx_x3ln_gtx.v
- aurora_apps_tx_x3ln_multi_wrapper.v
- aurora_apps_tx_x3ln_wrapper.v

All main source files for the RTL design files are located at the following directory:

**PROJECT_DIRECTORY/ip/aurora_apps_tx_x3ln/aurora_apps_tx_x3ln/src/**:

- aurora_apps_tx_x3ln_64b66b_scrambler.v
- aurora_apps_tx_x3ln_axi_to_ll.v
- aurora_apps_tx_x3ln_cdc_sync.v
- aurora_apps_tx_x3ln_clock_module.v
- aurora_apps_tx_x3ln_gt_common_wrapper.v
- aurora_apps_tx_x3ln_ll_to_axi.v
- aurora_apps_tx_x3ln_reset_logic.v
- aurora_apps_tx_x3ln_standard_cc_module.v
- aurora_apps_tx_x3ln_support_reset_logic.v
- aurora_apps_tx_x3ln_sym_gen.v
- aurora_apps_tx_x3ln_tx_aurora_lane_simplex.v
- aurora_apps_tx_x3ln_tx_ch_bond_code_gen_simplex.v
- aurora_apps_tx_x3ln_tx_channel_err_detect_simplex.v
- aurora_apps_tx_x3ln_tx_channel_init_sm_simplex.v
- aurora_apps_tx_x3ln_tx_err_detect_simplex.v
- aurora_apps_tx_x3ln_tx_global_logic_simplex.v
- aurora_apps_tx_x3ln_tx_lane_init_sm_simplex.v
- aurora_apps_tx_x3ln_tx_startup_fsm.v
- aurora_apps_tx_x3ln_tx_stream_control_sm_simplex.v
- aurora_apps_tx_x3ln_tx_stream_datapath_simplex.v
- aurora_apps_tx_x3ln_tx_stream_simplex.v
- aurora_apps_tx_x3ln_support.v

#### 4.3.3.7 Single Channel 3 Lanes Aurora Core RTL Wrapper

The 3-channels Aurora wrapper aurora_apps_tx_x3ln_channel_wrapper.v is a simplified and cleaned-up RTL of aurora_apps_tx_x3ln_support.v with the below changes:

- Tied off all DRP ports.
- Tied off unused GTX control ports.
- Consolidate the DIFFCTRL, post-, pre- and main- cursor controls of all three channels to one.
- Removal of the aurora_apps_tx_x3ln_clock_module (the clock module is moved to the aurora_apps_tx_x12ln.v so all four channels can share the same user clock). See Section 4.3.3.8 for details).

The purpose of cleaned-up RTL is to simplified I/O port list for ease of component instantiation in the next section.

#### 4.3.3.8 Four Channels 12 Lanes Top Level RTL Wrapper

Figure 4-17 showing aurora_apps_tx_x12ln.v has four instantiation of module aurora_apps_tx_x3ln_channel_wrapper.v to form the four Aurora TX channels entity.

Tx_out_clk from channel 0 feeds into aurora_apps_tx_x3ln_clock_module.v to generate the clk_user which drive the Apps FPGA and Aurora user logics interface. Refer to the Xilinx app note *Chapter 2 Table 2-7: Aurora 64B/66B Core Clock Ports* and *Chapter 3 Figure 3-1 Aurora 64B/66B Clocking Architecture* for information regarding tx_out_clk and clk_user.

**Note**

For link speed of 10Gbps with 64B/66B interface, clk_user frequency = 10GHz / 64 = 156.25MHz.

Reset logics generate reset signals *reset_pb* and *pma_init* to the four Aurora TX channels. Refer to the Xilinx app note *Chapter 3, Figure 3-5 Aurora 64B/66B Simplex Normal Operation Reset Sequence* for specification of generating reset_pb and pma_init.

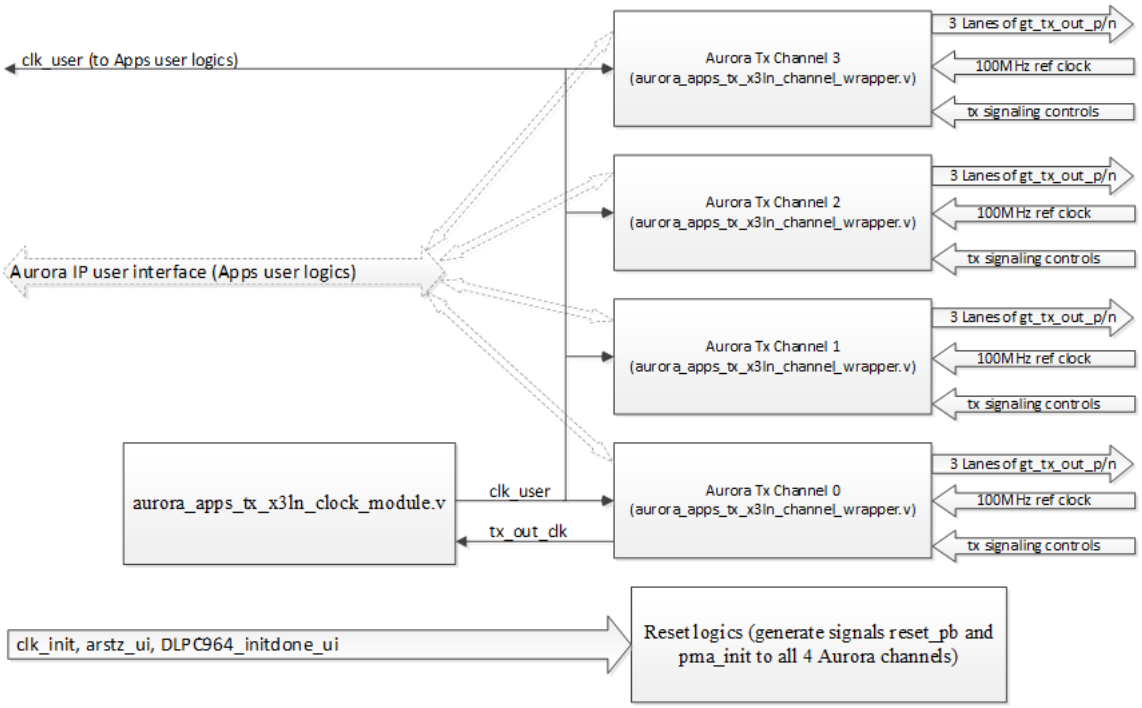



**Figure 4-17. Aurora_apps_tx_x12ln.v RTL Block Diagram**

**Table 4-1. Signal Port List for RTL aurora_apps_tx_x12ln.v**

| Name | Direction | Clock Domain | Description |
|---|---|---|---|
| clk_init | Input | clk_init | 100MHz free running clock for generate reset signals to Aurora cores. |
| arstz_ui | Input | async | Active low reset input. Input low trigger reset operation to Aurora cores. |
| dlpc964_initdone_ui | Input | async | DLPC964 Controller INIT_DONE status signal. Low keeps Aurora cores in reset state, |
| gt0_txout_n[0:2] | Output | async | Channel 0 10Gbps differential output Lane 0,1, and 2 to DLPC964 Controller |
| gt1_txout_p[0:2] | Output | async | Channel 1 10Gbps differential output Lane 0,1, and 2 to DLPC964 Controller |
| gt2_txout_p[0:2] | Output | async | Channel 2 10Gbps differential output Lane 0,1, and 2 to DLPC964 Controller |
| gt3_txout_p[0:2] | Output | async | Channel 3 10Gbps differential output Lane 0,1, and 2 to DLPC964 Controller |
| gt0_refclkin_p | Input | async | Channel 0 100MHz differential transceiver external reference clock from a low-jitter oscillator. |
| gt1_refclkin_p | Input | async | Channel 1 100MHz differential transceiver external reference clock from a low-jitter oscillator. |
| gt2_refclkin_p | Input | async | Channel 2 100MHz differential transceiver external reference clock from a low-jitter oscillator. |
| gt3_refclkin_p | Input | async | Channel 3 100MHz differential transceiver external reference clock from a low-jitter oscillator. |
| gt_txpostcursor_in[4:0] | Input | async | Transceiver post-cursor TX pre-emphasis control and is set to "00000" for TI EVM hardware.<br>Customer must perform IBERT eyescan to determine the best setting for the hardware. |
| gt_txdiffctrl_in[3:0] | Input | async | Transceiver TX driver swing control and is set to "1000" (807mV differential peak to peak swing) for TI EVM hardware.<br>Customer must perform IBERT eyescan to determine the best setting for the hardware. |
| gt_txmaincursor_in[6:0] | Input | async | Transceiver main-cursor TX control and is set to "0000000" for TI EVM hardware. Customer must perform IBERT eyescan to determine the best setting for the hardware. |
| gt_txprecursor_in[4:0] | Input | async | Transceiver pre-cursor TX pre-emphasis control and is set to "00000" for TI EVM hardware. Customer must perform IBERT eyescan to determine the best setting for the hardware. |
| clk_user | Output | clk_user | 156.25MHz clock for user interface to Aurora cores. |
| clk_user_not_locked_uo | Output | async | When high, indicates clk_user is not locked, and can be used to keep user logics in reset state if clk_user loose lock in power-up or system reset condition. |
| gt(0,1,2,3)_s_axi_tx_tdata[191:0] | Input | clk_user | DMD pixel data to be transmitted across the Aurora links. |
| gt(0,1,2,3)_s_axi_tx_tvalid | Input | clk_user | User logics asserted this signal high to indicate to Aurora core the DMD pixel data is valid to transmit. Aurora cores ignore data if tvalid is low. Refer to the Xilinx app note for the AXI4-stream *tready* signal behavior. |
| gt(0,1,2,3)_s_axi_tx_tready | Output | clk_user | Aurora cores assert this signal high when DMD pixel data is accepted. Deasserted when pixel data are ignored, ie. cores are not ready to accept data. Refer to the Xilinx app note for the AXI4-stream *tready* signal behavior. |
| gt(0,1,2,3)_s_axi_user_k_tx_tdata[191:0] | Input | clk_user | User-k control word data to be transmitted across the Aurora links. |
| gt(0,1,2,3)_s_axi_user_k_tx_tvalid | Input | clk_user | User logics asserted this signal high to indicate to Aurora core th user-k control word data is valid to transmit. Aurora core ignore data if tvalid is low. |
| gt(0,1,2,3)_s_axi_user_k_tx_tready | Output | clk_user | Aurora core asserts this signal high when user-k control word data are accepted. Deasserted when data are ignored, ie. cores not ready to accept data. |
| gt(0,1,2,3)_hard_err | Output | clk_user | Asserted high when Aurora core detects a hard error. Refer to the Xilinx app note *Table 2-13* for the hard error definition. |
| gt(0,1,2,3)_soft_err | Output | clk_user | Asserted high when Aurora core detects a soft error. Refer to the Xilinx app note *Table 2-13* for the soft error definition. |
| gt(0,1,2,3)_channel_up | Output | clk_user | Asserted high after Aurora cores complete the channel initialization sequence. |
| gt(0,1,2,3)_lane_up[2:0] | Output | clk_user | Asserted high for each lane upon successful lane initialization with each bit representing one lane. |
| tp_gt0_pll_lock | Output | async | Asserted high when Aurora channel 0 tx_out_clk is stable. As stated in earlier section, channel 0 tx_out_clk is used to generate clk_user.<br>tx_out_clk is 312.5MHz out of the Aurora transceiver and divided by two to form clk_user of 156.25MHz. |

#### 4.3.3.9 Block Start with Block Control Word

Aurora is a generic data transport link without concept of a DMD block. To define the start of a DMD block, the Apps user logics must send a Block Control word packet through channel 0 Aurora user-k port before data transmission.

Refer to Xilinx app note Chapter 2 Table 2-10 for detailed information regarding the Aurora user-k interface port. In summary, the user-k interface ports are used to implement application-specific control functions, and are independent and higher priority than the data interface. As shown in Table 4-2 , the RTL wrapper aurora_apps_tx_x12ln.v has four channels of user-k port interface exposed to Apps FPGA user logics.

**Note**

Only channel 0 is used to transmit the Block Control word. Control word packets sending over the user-k port of channel 1, 2 and 3 are not used and ignored by DLPC964 Controller.

**Table 4-2. RTL Wrapper "aurora_apps_tx_x12ln.v" User-k Ports Usage**

| Signal Name | Signal Direction | DLPC964 Application Usage |
|---|---|---|
| gt0_s_axi_user_k_tx_tdata[191:0] | Input to Aurora Channel 0 | 192 bits Block Control word packet to be trasmitted |
| gt0_s_axi_user_k_tx_tvalid | Input to Aurora Channel 0 | User logics asserted this signal high to indicate to Aurora core the Block Control word is valid to transmit.<br>Aurora cores ignore word if tvalid is low. |
| gt0_s_axi_user_k_tx_tready | Output from Aurora Channel 0 | Aurora cores assert this signal high when the Block Control word is accepted.<br>Deasserted when words are ignored, ie. cores are not ready to accept input word. |
| gt1_s_axi_user_k_tx_tdata[191:0] | Input to Aurora Channel 1 | Unused |
| gt1_s_axi_user_k_tx_tvalid | Input to Aurora Channel 1 | Unused |
| gt1_s_axi_user_k_tx_tready | Output from Aurora Channel 1 | Unused |
| gt2_s_axi_user_k_tx_tdata[191:0] | Input to Aurora Channel 2 | Unused |
| gt2_s_axi_user_k_tx_tvalid | Input to Aurora Channel 2 | Unused |
| gt2_s_axi_user_k_tx_tready | Output from Aurora Channel 2 | Unused |
| gt3_s_axi_user_k_tx_tdata[191:0] | Input to Aurora Channel 3 | Unused |
| gt3_s_axi_user_k_tx_tvalid | Input to Aurora Channel 3 | Unused |
| gt3_s_axi_user_k_tx_tready | Output from Aurora Channel 3 | Unused |

Table 4-3 describes the various fields within the 192 bits Block Control word. Block Control word not only defines the start of a DMD block, but also contains instruction and information to guide the DLPC964 controller on how to process the receiving DMD block data

**Table 4-3. Block Control Word Fields Definition**

| Field Position | Field Type | Field Description |
|---|---|---|
| gt0_s_axi_user_k_tx_tdata[7:0] | USER_BLOCK_NUMBER | Must set to zeros (0x00). Values other than 0x00 are invalid, DLPC964 controller ignores the entire 192 bits control word if this field is not zeros. |
| gt0_s_axi_user_k_tx_tdata[11:8] | BLOCK_ADDRESS | Indicates the address of the DMD block to which DLPC964 applies the operation.<br>0000: DMD Block 0<br>0001: DMD Block 1,<br>0010: DMD Block 2<br>……..<br>1111: DMD Block 15 |
| gt0_s_axi_user_k_tx_tdata[15:7] | | Reserved, unused. |
| gt0_s_axi_user_k_tx_tdata[24:16] | ROW_LENGTH | Number of row DLPC964 loads the user data. DLP991U and DLP991UUV DMD has 136 rows per block, thus valid range is 1-136. All other values, including 0 are invalid. Set to 136 for full block operation. Value 1 to 135 for partial block.<br>**Note**<br>This field used if LOAD_TYPE is 000. |
| gt0_s_axi_user_k_tx_tdata[34:32] | LOAD_TYPE | 000: Block loading. DLPC964 load the user data to the DMD array area defined by the BLOCK_ADDRESS and ROW_LENGTH.<br>001: Block clear. DLPC964 clear the DMD array to zeros of the entire block define by BLOCK_ADDRESS.<br>010: Block set. DLPC964 set the DMD array to ones of the entire block defined by BLOCK_ADDRESS. Other values: reserved, do not use.<br>**Note**<br>When in 001 (block clear) or 010 (block set) operation, the ROW_LENGTHand NORTH_SOUTH_FLIP fields are ignored because clear and set operation affect the entire DMD block array. In other words, block set/clear operation do not support partial block operation. |
| gt0_s_axi_user_k_tx_tdata[36] | NORTH_SOUTH_FLIP | Control the direction of data loading within a DMD block.<br>0: DLPC964 load data starting and counting up from row 1.<br>1: DLPC964 load data starting and counting down from row 136. |
| gt0_s_axi_user_k_tx_tdata[29:28] | DMD_SEGMENT | When SINGLE_CHANNEL_MODE = '1', select the DMD segment to which the DLC964 applies the operation. DLPC964 Controller ignores this field if SINGLE_CHANNEL_MODE = '0'. |
| gt0_s_axi_user_k_tx_tdata[30] | SINGLE_CHANNEL_MODE | 1: Signle channel operation. Operate the DMD array with only Aurora channel 0.<br>0: Normal operation. Operate the DMD array with all four Aurora channels. |
| gt0_s_axi_user_k_tx_tdata[191:31] | | reserved, unused. |

Figure 4-18 showing the transmission of the 192 bits Block Control word over the channel 0 user-k port at the start of an Aurora block transfer (in this example, loading all 136 lines of DMD block 1).

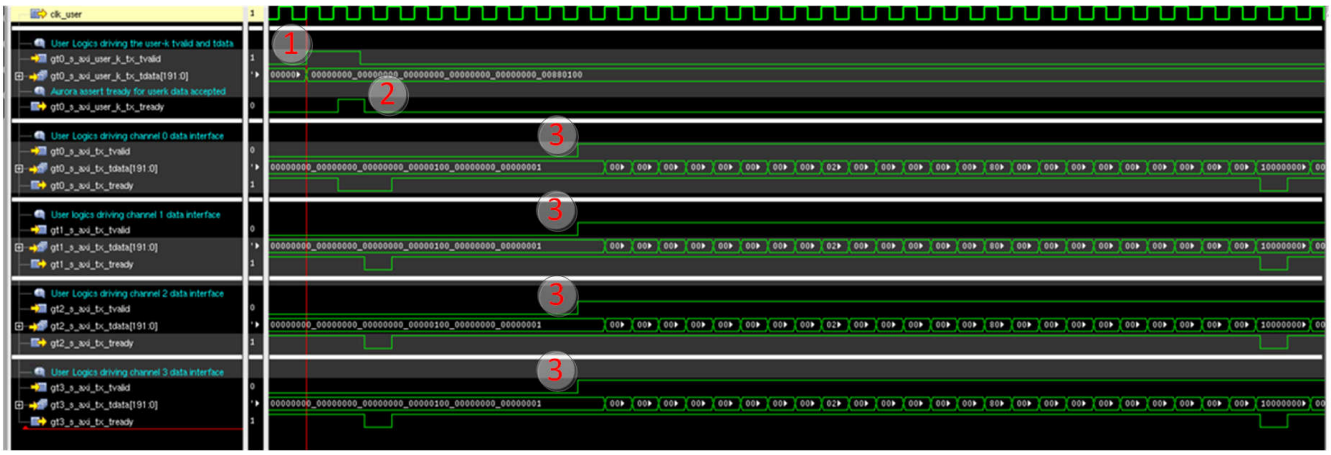



**Figure 4-18. Block Start with Block Control Word Waveform**

1. With the proper Block Control word on bus gt0_s_axi_user_k_tx_tdata[191:0], the Apps FPGA user logics asserts the TVALID flag (gt0_s_axi_user_k_tx_tvalid), and waits for the response of the Aurora core.
2. Aurora asserts the TREADY flag, gt0_s_axi_user_k_tx_tready, indicating the core has accepted the 192 bits user-k data.
3. After the Block Control word is sent, Apps FPGA user logic starts the Aurora block transfer on all four data interfaces.

#### 4.3.3.10 Block Complete with DMDLOAD_REQ

Refer to Figure 4-11: DLPC964 System Block Diagram

DMDLOAD_REQ is an output signal from the Apps FPGA to DLPC964 Controller.

Once an Aurora block data transfer is completed, the Apps FPGA user logics must assert the DMDLOAD_REQ to signal the DLPC964 that this is the end of a DMD block and trigger to carry out the operation encoded in the Block Control word.

Guidelines for asserting the DMDLOAD_REQ signal and sending Block Control word:

- Apps FPGA user logics must wait for the block transfer to be completed on all four Aurora data channels before asserting DMDLOAD_REQ.
  - Apps FPGA must take into account that the four Aurora data channel interfaces are not fully synchronous to each other, thus data completion do not happen at the exact same clock cycle. Therefore, the Apps FPGA must monitor and verify the Aurora block data transfer is completed on all four channels before asserting DMDLOAD_REQ.

**Note**

Asserting DMDLOAD_REQ before completion of Aurora block transfer can result in data not loaded properly to DMD.

- DMDLOAD_REQ can be asserted immediately after completing an Aurora block transfer, as long as the 300ns DMDLOAD_REQ setup time is met (refer to Section 4.3.3.11 for more information).
- Apps FPGA user logics must assert DMDLOAD_REQ for the current block before initiate the transmission of the next new DMD block. Every block must start with a Block Control word packet and end with DMDLOAD_REQ assertion.
- DMDLOAD_REQ is still required for operations that do not involve block data transfer (such as block clear/set operation), and must still meet the 300ns setup time (refer to Section 4.3.3.11 for more information).
- Refer to Figure 4-19 for the scenario where after the Apps user logics has completed the transfer of current block. The user can find that the DLPC964 is still loading the previous block to the DMD (for example, BLKLOADZ is low). The Apps FPGA can still assert the DMDLOAD_REQ while BLKLOADZ is low because the DLPC964 can detect and store this DMDLOAD_REQ request. After completing the Aurora data transfer of the current DMD block and asserting the DMDLOAD_REQ signal, the Apps FPGA must wait for the de-assertion of BLKLOADZ by the DLPC964 (for example, BLKLOADZ transition from low to high) before starting the next block. De-assertion of BLKLOADZ means the DLPC964 has completed the DMD data loading operation for previous block and a data buffer is freed up for accepting a new data block from the Aurora interface.

**Note**

The DLPC964 has two data block buffers; one for receiving the incoming Aurora data block, the other for holding the previous block for streaming out to DMD. The buffers can be overrun and data is not loaded correctly to the DMD if Apps FPGA does not synchronize the Aurora block transfer with the BLKLOADZ's de-assertion signal.

- Refer to Figure 4-20 for a scenario where the Apps FPGA chooses to send the DLPC964 a DMD data block, but delay the assertion of the DMDLOAD_REQ.


**Figure 4-19. End of Block DMDLOAD_REQ Assertion Follow By New Block Control Word Waveform**

1. Apps FPGA user logics assert DMDLOAD_REQ immediately after the completion of current block data transmission on all four Aurora data interfaces.
2. DLPC964 de-assert BLKLOADZ indicating completion of data loading operation for previous DMD block.
3. Apps FPGA user logics detects the de-assertion of BLKLOADZ and send a new Block Control word on Aurora channel 0 user-k port for next block.
4. Apps FPGA user logics sending data for the next block.
5. BLKLOADZ asserted low by DLPC964 indicating that the data loading operation for the current block is triggered by DMDLOAD_REQ.

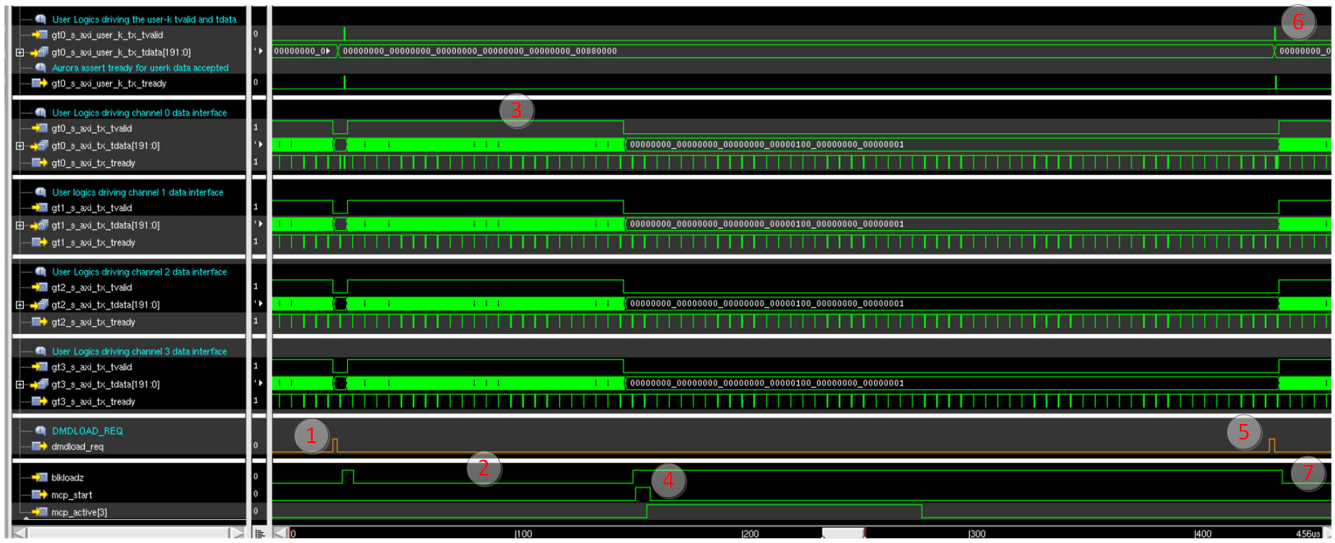



**Figure 4-20. DMDLOAD_REQ Delayed Assertion Waveform**

1. Apps FPGA finishes sending the last block (block 15) data of current pattern and asserts DMDLOAD_REQ to instruct the DLPC964 to carry out the data load operation.
2. DLPC964 loading data to block 15 triggered by DMDLOAD_REQ from 1.
3. Apps FPGA sends the first block (block 0) of data of the next pattern over the Aurora data interfaces while the DLPC964 is loading block 15 of the current pattern.
4. DLPC964 de-asserts BLKLOADZ after the data loading of block 15 of current pattern is complete. The Apps FPGA detected the BLKLOADZ de-assertion as the last block of the current pattern and has been loaded onto the DMD and issued a MCP_START for global block reset operation.
5. The Apps FPGA delayed the assertion of DMDLOAD_REQ for block 0 for the next pattern due to the requirement of meeting the mirror settling time.
6. Send Block Control word for block 1 for the next pattern after the assertion of DMDLOAD_REQ for block 0.
7. DLPC964 asserts BLKLOADZ to indicate that the DMD data loading operation was triggered by DMDLOAD_REQ from part 5.

#### 4.3.3.11 DMDLOAD_REQ Setup Time Requirement

Apps FPGA user logics can assert the DMDLOAD_REQ signal as soon as completing an Aurora block transfer as long as the signal is at least 300ns after sending the first data packet of that block. This setup time requirement is due to the 300ns transmit latency of the Aurora TX/RX channel paths, thus verifies the DLPC964 receives the DMDLOAD_REQ flag after the arrival of Aurora block data.

In most cases, this 300ns setup requirement is met naturally as a data block is large enough to verify over 300ns from the first valid data packet being sent to the last ones of a block when the Apps can assert the DMDLOAD_REQ signal. Cases of this 300ns setup window become critical is when Apps FPGA tries to send a small partial DMD block such as in Figure 4-21 showing an example of the Apps FPGA sending a total of 3 rows (Table 4-2 , ROW_LENGTH = 3) of a DMD partial block to DLPC964:

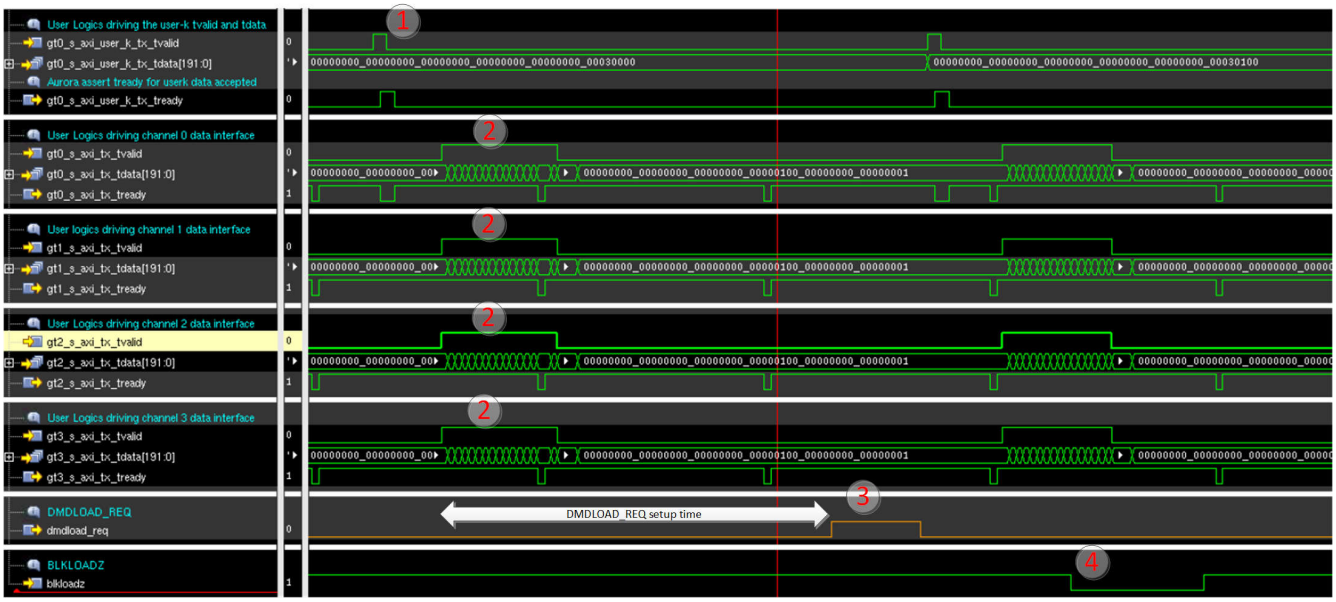



**Figure 4-21. DMDLOAD_REQ Setup Time for Three DMD Rows Load Operation**

1. Apps FPGA transmits a Block Control word to indicate the start of an Aurora block transfer.
2. After sending three rows of data through the four Aurora data interface channels, the Apps FPGA waits for the 300ns setup time to expire before issues a DMDLOAD_REQ.

**Note**

The 300ns is measured from the start of the first TVALID on the data interface.

3. Apps FPGA asserts DMDLOAD_REQ once the setup time is meet.
4. BLKLOADZ asserted by DLPC964 indicating DMD data load operation in progress.

For operations that do not require the data packet, such as block clear (Table 3, LOAD_TYPE = 001) and block set (Table 3, LOAD_TYPE = 010). This DMDLOAD_REQ of 300ns setup time is still required and measured from the Block Control word packet. Figure 4-22 is an example of a block set operation.

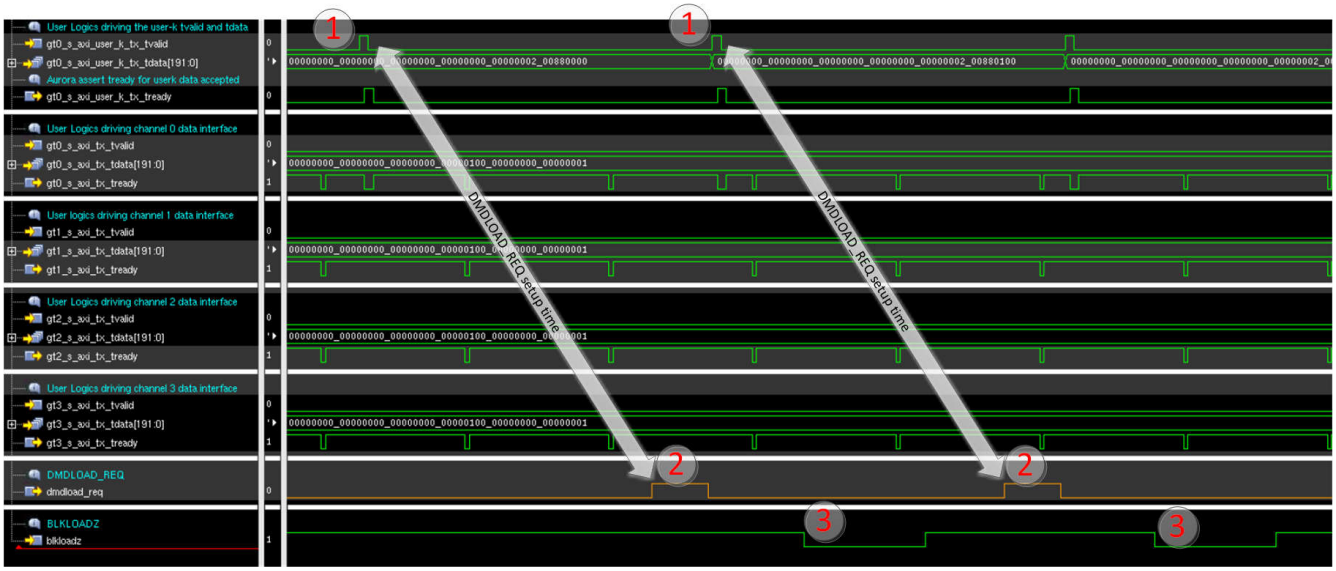



**Figure 4-22. DMDLOAD_REQ Setup Time For Block Set Operation**

1. Apps FPGA transmits a Block Control word packet to start a block set operation. Notice this operation does not require any block data as the four data interfaces stay idle (gtX_s_axi_tx_tvalid = ‘0’).
2. Apps FPGA asserts DMDLOAD_REQ after the 300ns setup time. 300ns is measured from Block Control word as block set operation does not require Aurora data transfer.
3. DLPC964 asserts BLKLOADZ indicating block set operation in progress.

#### 4.3.3.12 Single Channel Transfer Mode

For non-critical pattern rate application, the DLPC964 supports operating with only Aurora channel 0. Only the three 10Gbps serial links for channel 0 are used and must be channel 0 in this mode of operation.

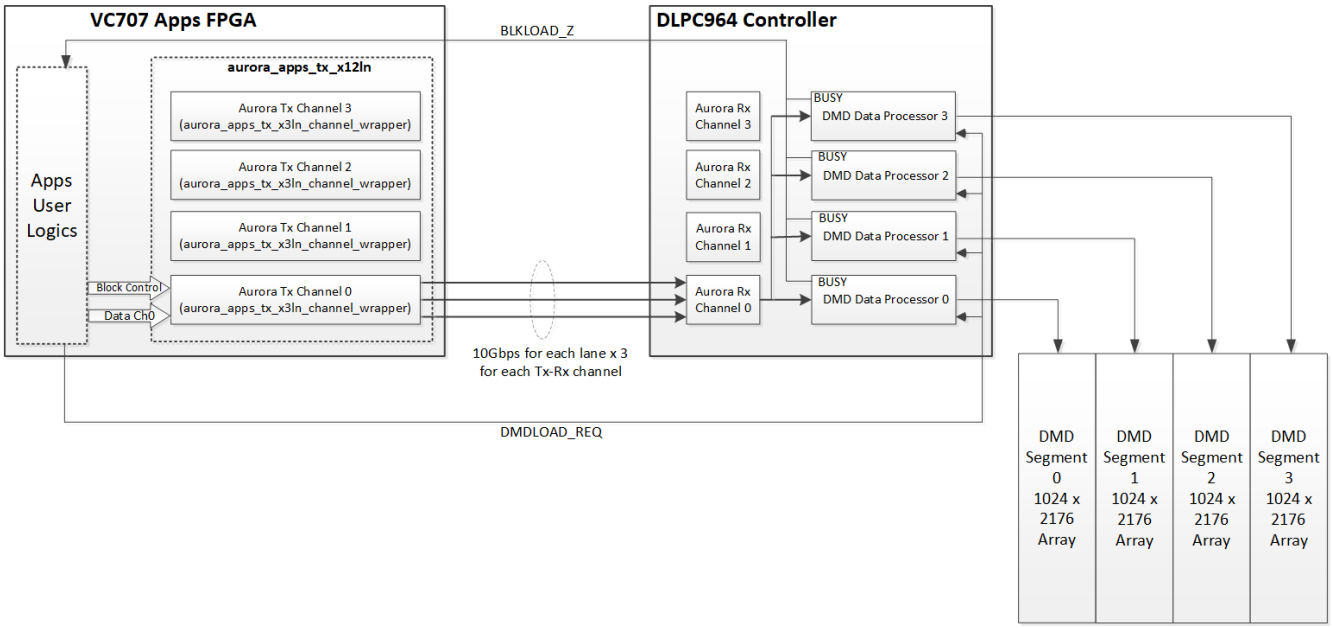



**Figure 4-23. System Block Diagram For Single Channel Operation**

Operation is enabled by setting Block Control word field, SINGLE_CHANNEL = '1' (Table 4-2 ), and transfer DMD segment in order of 3(first), 2, 1, 0(last) (DMD_SEGMENT field in Table 4-2 ). In other words, to control a particular DMD block, theApps FPGA must operate segment 3 of that block first, follow by segment 2, segment 1, and 0 being the last transfer segment.

The guidelines stated in Section 4.3.3.9, Section 4.3.3.10, and Section 4.3.3.11, still apply to single channel operation mode where each block and segment Aurora transfer must still begin with a Block Control word and end with the DMDLOAD_REQ along with the 300ns setup time. However, one major difference regarding the Apps FPGA/DLPC964 handshaking in this mode is that the actual DMD operation only triggered by the DMDLOAD_REQ corresponds to segment 0; for example, BLKLOADZ is not asserted for segment 3, 2 and 1. (See Figure 4-24 for details).

All four segments of the selected block must be operated and in order of segment 3 (first), 2, 1 and 0 (last), otherwise the DLPC964 do not carry out the proper DMD operation to that block.

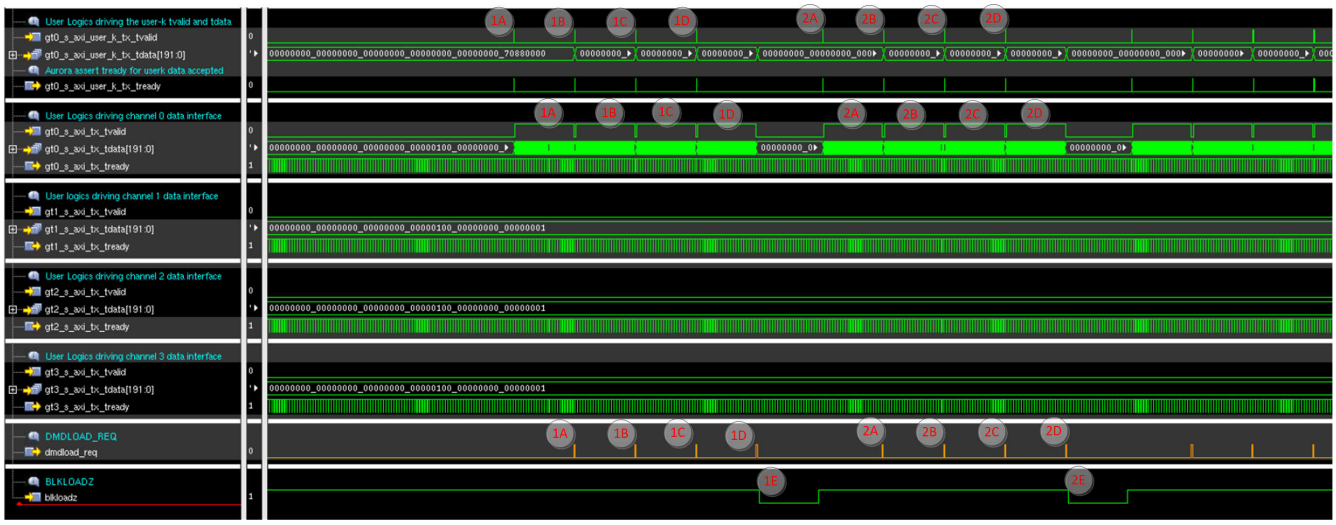



**Figure 4-24. Single Channel Operation Waveform Example**

1. Apps FPGA Aurora Data Transfer for DMD Block 0 in Single Channel Mode.
   a. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 0 Segment 3
   b. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 0 Segment 2.
   c. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 0 Segment 1.
   d. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 0 Segment 0.
   e. Segment 0's DMDLOAD_REQ triggers DLPC964 to begin block 0 data loading, assert BLKLOADZ to indicate operation in process.
2. Apps FPGA Aurora Data Transfer for DMD Block 1 in Single Channel Mode.
   a. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 1 Segment 3.
   b. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 1 Segment 2.
   c. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 1 Segment 1.
   d. Block Control word, DMD data, and DMDLOAD_REQ for DMD block 1 Segment 0.
   e. Segment 0's DMDLOAD_REQ triggers DLPC964 to begin block 1 data loading, assert BLKLOADZ to indicate operation in process.

Note that there is no data transfer happening on GT channel 1, 2, and 3. Only channel 0 is operated in this Single Channel mode.

#### 4.3.3.13 DMD Block Array Data Mapping

To each Aurora core, a full DMD block array is 1024 columns by 136 rows. Table 4-25 shows the mapping of the 192 bits Aurora data bus to a full DMD block in increment direction (first Aurora data packet start at row 0). A full DMD block requires 726 Aurora data packets to transmit a full block. For the last packet, only bits 0-63 are required and bits 64-191 are ignored by DLPC964.

Table 4-26 next page shows the data mapping in the decrement direction with the first Aurora data packet start at row 135.

| | | | | | | |
|---|---|---|---|---|---|---|
| | Data 0 [0:191] | Data 1 [0:191] | Data 2 [0:191] | Data 3 [0:191] | Data 4 [0:191] | Data 5 [0:63] |
| Row 0 | Column 0-191 | Column 192-383 | Column 384-575 | Column 576-767 | Column 768-959 | Column 960-1023 |
| | Data 5 [64:191] | Data 6 [0:191] | Data 7 [0:191] | Data 8 [0:191] | Data 9 [0:191] | Data 10 [0:127] |
| Row 1 | Column 0-127 | Column 128-319 | Column 320-511 | Column 512-703 | Column 704-895 | Column 896-1023 |
| | Data 10 [128:191] | Data 11 [0:191] | Data 12 [0:191] | Data 13 [0:191] | Data 14 [0:191] | Data 15 [0:191] |
| Row 2 | Column 0-63 | Column 64-255 | Column 256-447 | Column 448-639 | Column 640-831 | Column 832-1023 |
| | Data 16 [0:191] | Data 17 [0:191] | Data 18 [0:191] | Data 19 [0:191] | Data 20 [0:191] | Data 21 [0:63] |
| Row 3 | Column 0-191 | Column 192-383 | Column 384-575 | Column 576-767 | Column 768-959 | Column 960-1023 |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| | Data 714 [128:191] | Data 715 [0:191] | Data 716 [0:191] | Data 717 [0:191] | Data 718 [0:191] | Data 719 [0:191] |
| Row 134 | Column 0-63 | Column 64-255 | Column 256-447 | Column 448-639 | Column 640-831 | Column 832-1023 |
| | Data 720 [0:191] | Data 721 [0:191] | Data 722 [0:191] | Data 723 [0:191] | Data 724 [0:191] | Data 725 [0:63] |
| Row 135 | Column 0-191 | Column 192-383 | Column 384-575 | Column 576-767 | Column 768-959 | Column 960-1023 |

**Figure 4-25. Aurora Data Bus to DMD Block Array Mapping Increment Direction**

| | | | | | | |
|---|---|---|---|---|---|---|
| | Data 720 [0:191] | Data 721 [0:191] | Data 722 [0:191] | Data 723 [0:191] | Data 724 [0:191] | Data 725 [0:63] |
| Row 0 | Column 0-191 | Column 192-383 | Column 384-575 | Column 576-767 | Column 768-959 | Column 960-1023 |
| | Data 714 [128:191] | Data 715 [0:191] | Data 716 [0:191] | Data 717 [0:191] | Data 718 [0:191] | Data 719 [0:191] |
| Row 1 | Column 0-63 | Column 64-255 | Column 256-447 | Column 448-639 | Column 640-831 | Column 832-1023 |
| | Data 709 [64:191] | Data 710 [0:191] | Data 711 [0:191] | Data 712 [0:191] | Data 713 [0:191] | Data 714 [0:127] |
| Row 2 | Column 0-127 | Column 128-319 | Column 320-511 | Column 512-703 | Column 704-895 | Column 896-1023 |
| | Data 704 [0:191] | Data 705 [0:191] | Data 706 [0:191] | Data 707 [0:191] | Data 708 [0:191] | Data 709 [0:63] |
| Row 3 | Column 0-191 | Column 192-383 | Column 384-575 | Column 576-767 | Column 768-959 | Column 960-1023 |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| | Data 5 [64:191] | Data 6 [0:191] | Data 7 [0:191] | Data 8 [0:191] | Data 9 [0:191] | Data 10 [0:127] |
| Row 134 | Column 0-127 | Column 128-319 | Column 320-511 | Column 512-703 | Column 704-895 | Column 896-1023 |
| | Data 0 [0:191] | Data 1 [0:191] | Data 2 [0:191] | Data 3 [0:191] | Data 4 [0:191] | Data 5 [0:63] |
| Row 135 | Column 0-191 | Column 192-383 | Column 384-575 | Column 576-767 | Column 768-959 | Column 960-1023 |

**Figure 4-26. Aurora Data Bus to DMD Block Array Mapping Decrement Direction**

#### 4.3.3.14 Xilinx IBERT

The signal integrity of the 10Gbps link can be verified with the Xilinx IBERT (Integrated Bit Error Ratio Tester) toolset. Refer to Xilinx user's guides (Section 6) for more details regarding the IBERT tool.

As shown in Table 4-1, there are four input ports to the RTL to control the TX transceiver setting. The TI EVM hardware are configured as below.

**Table 4-4. Input Ports to the RTL to Control the TX Transceiver Setting**

| Signal Name | I/O Direction | Clock Domain | Description |
|---|---|---|---|
| gt_txpostcursor_in[4:0] | Input | Async | Transceiver post-cursor TX pre-emphasis control. Set to "00000" for TI EVM hardware. Customers must perform IBERT eyescan to determine the best setting for the hardware. |
| gt_txdiffctrl_in[3:0] | Input | Async | Transceiver TX driver swing control. Set to "1000" (807mV differential peak to peak swing) for TI EVM hardware. Customer must perform IBERT eyescan to determine the best setting for the hardware. |
| gt_txmaincursor_in[6:0] | Input | Async | Transceiver main-cursor TX control. Set to "0000000" for TI EVM hardware. Customer musy perform IBERT eyescan to determine the best setting for the hardware. |
| gt_txprecursor_in[4:0] | Input | Async | Transceiver pre-cursor TX pre-emphasis control. Set to "00000" for TI EVM hardware. Customer must perform IBERT eyescan to determine the best setting for the hardware. |

In addition, the DLPC964 has an input pin RXLPEN to select between the low power mode (‘0’) or DFE (‘1’) equalization for the DLPC964 Xilinx GT cell transceiver. For TI EVM, RXLPEN is set to 0 for low power mode equalization. Refer to the Xilinx app note for information regarding RXLPEN.

With the above RX/TX transceiver settings (TX post-, main-, pre- cursor, TX diffctrl, and RXLPEN) selected and enabled in the IBERT GUI, Figure 4-27 shows an IBERT scan result of one of the 12 high speed links with eye opening of 200+ vertical codes, and 0.6UI horizontal at BER of 1e-12 (the purple area).

**Note**

Users need to select PRBS31, the highest duty cycle binary sequence option within the IBERT tool, to mimic the traffic pattern of the 64b/66B encoding characteristics.

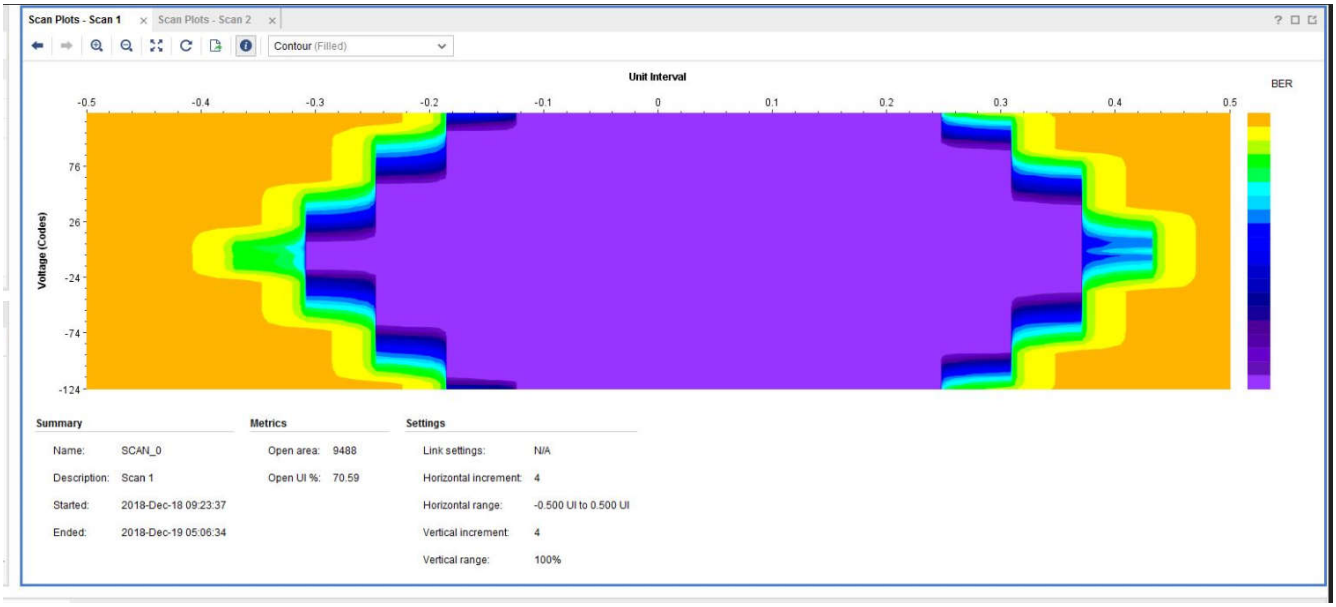



**Figure 4-27. IBERT Eye-scan For Aurora Channel0 Link0 Using TI EVM Hardware**

# 5 Abbreviations and Acronyms

The following lists abbreviations and acronyms used in this manual:

| | |
|---|---|
| **Apps FPGA** | AMD Xilinx Virtex 7 FPGA on the VC-707 EVM or similar board for customer applications |
| **BRG** | Block Reset Generator |
| **DMD** | Digital Micromirror Device |
| **EVM** | Evaluation Module (Board) |
| **FCC** | Federal Communications Commission |
| **FPGA** | Field Programmable Gate Array |
| **FW** | Firmware |
| **GPIF** | General Purpose Interface |
| **GPIO** | General Purpose Input Output |
| **GUI** | Graphical User Interface |
| **HSSI** | High Speed Serial Interface |
| **HW** | Hardware |
| **IBERT** | Integrated Bit Error Ratio Tester |
| **$I^2C$** | Inter-Integrated Circuit |
| **IP** | Internet Protocol |
| **JTAG** | Joint Test Action Group |
| **MCP** | Mirror Clocking Pulse |
| **PBC** | Processor Bus Control |
| **PCB** | Printed Circuit Board |
| **PGEN** | Pattern Generator |
| **PLL** | Phase-Locked Loop |
| **SDK** | Software Development Kit |
| **SPI** | Serial-Peripheral Interface |
| **SW** | Switch |
| **USB** | Universal Serial Bus |
| **VHDL** | Verification and Hardware Description Language |

# 6 Related Documentation from Texas Instruments

Component data sheets, technical documents, design documents, and ordering information can be found at the following links:

DLPC964 Digital Controller Product Folder

DLP LightCrafter DLPC964 EVM Tool Folder

DLPC964 EVM Users Guide

DLP991UFLV DMD Product Folder

DLP LightCrafter DLP991UFLV DMD EVM Product Folder

DLP LightCrafter DLP991UUVFLV DMD EVM Product Folder

Aurora 64B/66B v11.2 LogiCORE IP Product Guide

Integrated Bit Error Ratio Tester 7 Series GTH Transceivers v3.0

# 7 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

**Changes from March 11, 2024 to October 11, 2024 (from Revision * (March 2024) to Revision A (February 2025))** **Page**

- The DLPLCR99UVEVM (DLP991UUV DMD EVM) is now included throughout the document as an alternative DMD EVM to the DLPLCR99EVM (DLP991U DMD EVM).....1
- Included the DLP991UUV DMD in Apps FPGA Hardware Target (Figure 1-1).....4

# IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265