DLP® Discovery™ 4100 - Applications FPGA Pattern Generator Design

User's Guide

Literature Number: DLPU045A
September 2016–Revised December 2018

Contents

Revision History

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

Changes from Original (September 2016) to A Revision **Page**

- Added load4z output to IO list 5
- Added USB signals to IO list.................... 5
- Added gpio signals to IO list 5
- Added Note - only 400 MHz DCLK supported in new revision (w/DLPR410A). 8
- Added "appcore" reference and diagram to newly added GUI block. 9
- Added new files supporting 32 and 64 idle clocks between lines. 11
- Added herringbone pattern 12
- Added GUI block section and subsections and diagram 13

[1][2]

[1] DLP® Discovery is a trademark of Texas Instruments.

[2] DLP is a registered trademark of Texas Instruments.

User's Guide
DLPU045A–September 2016–Revised December 2018

DLP® Discovery™ 4100 - Applications FPGA Pattern Generator Design

This document discusses the design of the Applications FPGA pattern generator. Block diagrams and operation scenarios are provided to offer an overall understanding of the block functionality and describe the various modes of operation.

1 General Overview

The pattern generation Applications FPGA is part of the DLP® Discovery ™ 4100 development platform. It is intended to exercise the functions of the DLPC410 DMD Controller to display images and patterns on the corresponding digital micromirror device (DMD). There are five supported DMDs and they are: DLP650LNIR, DLP7000, DLP7000UV, DLP9500 and DLP9500UV.

1.1 IO List

The APPSFPGA has the following input and output ports:

Table 1. IO List

Signal Name	Input/Output	Description
clk_i	Input	Input 50 MHz reference clock
reset_i	Input	Input reset
finished_iv_o	Output	Configuration complete output
arstz_o	Output	Output reset (PLL reset)
clk_r_o	Output	Output 50 MHz reference clock
dout_ap_o	Output(15:0)	Channel A output data
dout_an_o	Output(15:0)	Channel A output data
dout_bp_o	Output(15:0)	Channel B output data
dout_bn_o	Output(15:0)	Channel B output data
dout_cp_o	Output(15:0)	Channel C output data
dout_cn_o	Output(15:0)	Channel C output data
dout_dp_o	Output(15:0)	Channel D output data
dout_dn_o	Output(15:0)	Channel D output data
dclk_ap_o	Output	Channel A clock
dclk_an_o	Output	Channel A clock
dclk_bp_o	Output	Channel B clock
dclk_bn_o	Output	Channel B clock
dclk_cp_o	Output	Channel C clock
dclk_cn_o	Output	Channel C clock
dclk_dp_o	Output	Channel D clock
dclk_dn_o	Output	Channel D clock
dvalid_ap_o	Output	Channel A data valid
dvalid_an_o	Output	Channel A data valid
dvalid_bp_o	Output	Channel B data valid
dvalid_bn_o	Output	Channel B data valid
dvalid_cp_o	Output	Channel C data valid

Table 1. IO List (continued)

Signal Name	Input/Output	Description
dvalid_cn_o	Output	Channel C data valid
dvalid_dp_o	Output	Channel D data valid
dvalid_dn_o	Output	Channel D data valid
rowmd_o	Output(1:0)	Row Mode
rowad_o	Output(10:0)	Row Address
stepvcc_o	Output	Unused
comp_data_o	Output	Complement data
ns_flip_o	Output	North/South flip
blkad_o	Output(3:0)	Block Address
blkmd_o	Output(1:0)	Block mode
wdt_enablez_o	Output	Watchdog time enable
ddc_version_i	Input(2:0)	DLPC410 Version number
dmd_type_i	Input(3:0)	DMD type designator
pwr_floatz_o	Output	Power Float
apps_cntl_an	Input	Unused
apps_cntl_ap	Input	Unused
rst2blkz_o		Dual block reset mode select
in_rst_active_i	Input	Reset active indicator
in_init_active_i	Input	Initialization active indicator
in_dip_sw_i	Input(7:0)	Switch inputs
in_pb_sw_i	Input	Input pushbutton
apps_logic_rstn	Input	Input reset
apps_testpt	Output(30:0)	Test point outputs
load4z	Output	Load4z enable signal($\overline{\text{LOAD4}}$)
clk_usb	Input	Input USB clock
ctl0	Input	USB Control Bit 0
ctl1	Input	USB Control Bit 1
ctl2	Input	USB Control Bit 2
bidir	InputOutput(15:0)	USB Data Bus
gpioa_o	Output(2:0)	GPIO outputs
gpioa_i	Input(2:0)	GPIO inputs
gpio_ext_reset_in	Input	gpio external reset input
gpio_reset_complete_o	Output	gpio reset complete

2 APPSFPGA Top Level

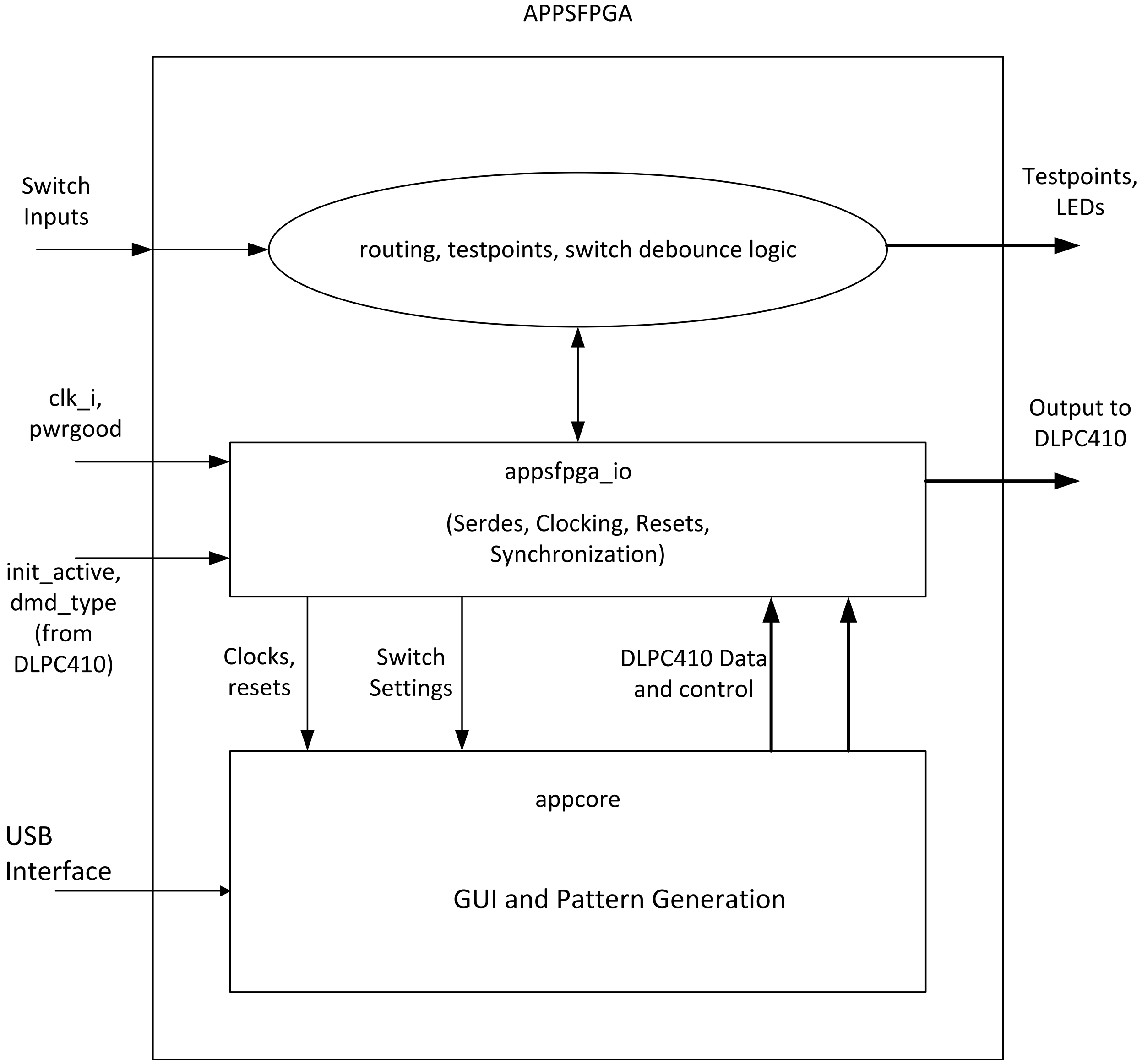


Figure 1. APPSFPGA Top Level

The top level of the APPSFPGA design provides some routing, contains the incoming DIP switch and pushbutton debounce logic, test point output assignments, and instantiation of the two major blocks of the design. These are the "appsfpga_io" block and the "appcore" block.

2.1 Input Switch Functions

There are several input switch functions that control the various operational modes of the APPSFPGA. They are:

- in_pb_sw_i : Power Float [active low (0)]

  Sends a sequence of commands that places the DMD mirrors in a flat or “parked” state (must power cycle to recover from this command).
- in_dip_sw_i(7) : Watchdog Timer enable [active high (1)]

  Enables a timer that will force a mirror reset to the DMD approximately every 10 seconds (if one has not been issued).
- in_dip_sw_i(6) : Row Address mode [low (0) = automatic increment/decrement mode, high (1) = row addressing mode]

  Selects between addressing specific rows and a row increment/decrement mode.
- in_dip_sw_i(5:4) : Reset Type [encoded value]

  Determines if global, single, dual, or quad resets are being used.
  - Single block -- 0,0
  - Dual block -- 0,1
  - Global -- 1,0
  - Quad block -- 1,1
- in_dip_sw_i(3) : North/South flip [active high (1) -- flips the image top to bottom]

  Determines if the row and reset addressing starts at the “top” or “bottom” of the DMD.
- in_dip_sw_i(2) : Complement Data [active high (1) -- inverts the image data]

  Causes the DMD to invert the incoming image data such that 0's become 1's and 1's become 0's
- in_dip_sw_i(1) : Count Halt [active high (1) -- halts the main counter]

  “Freezes” the load operation at a particular point in the pattern count cycle. See Section 4.1.1.
- in_dip_sw_i(0) : IO Float [active high (1) -- sets V_Bias to 0]

  Recoverable version of the “Power float” command (this command will recover if de-asserted).

For additional information see the DLPC410 datasheet.

3 "appsfpga_io" Block

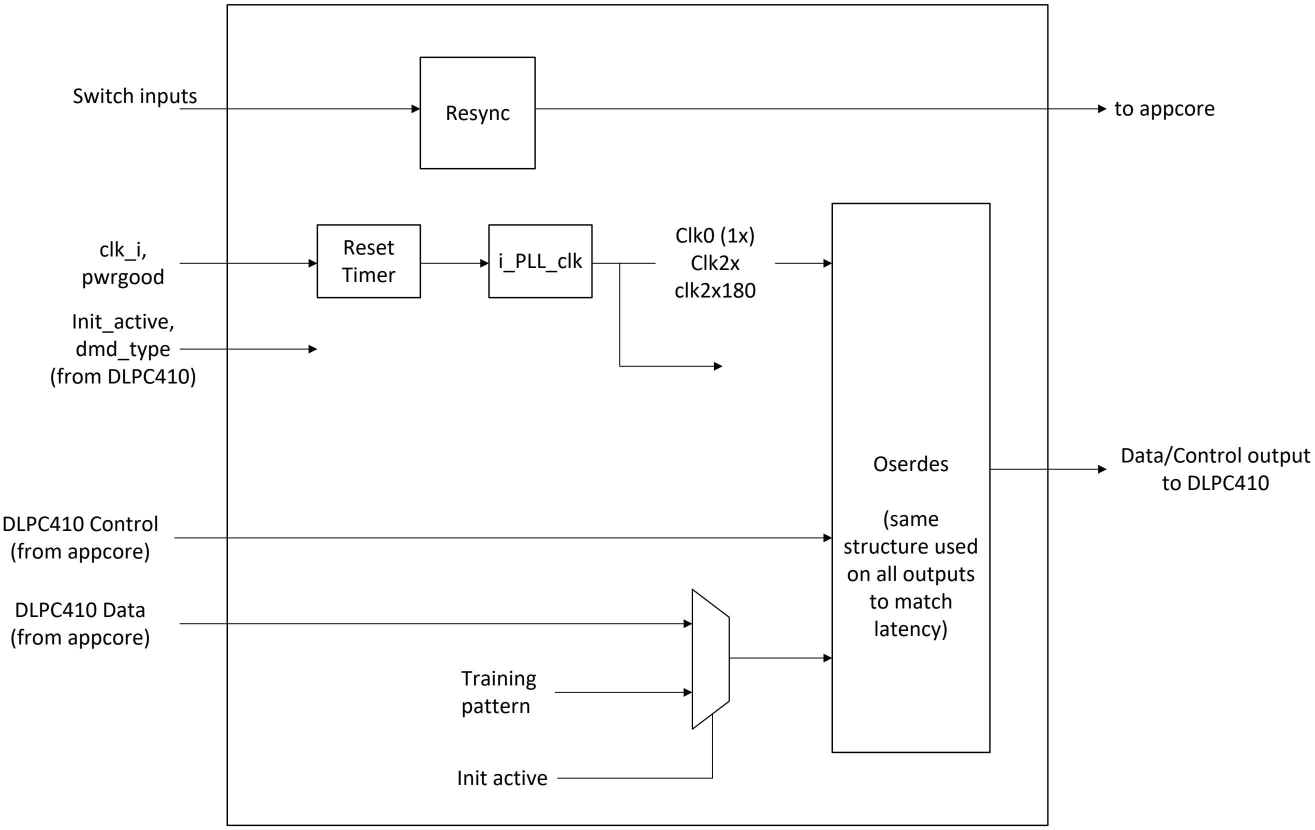


Figure 2. "appsfpga_io" Block Diagram

The "appsfpga_io" block performs the following functions:

It receives the input reference clock and uses the PLL to create the other clocks used in the design. The incoming clock is multiplied up to a clock that is one half the base clock frequency (200 MHz for a 400 MHz DDR data rate design) which is used in the rest of the design, and a phase aligned clock at the base clock frequency (200 MHz for the 400 MHz DDR data rate design) which is only used in the output section. Three versions of the "appsfpga_io" block are provided, the appsfpga_io_200_a.vhd, the appsfpga_io_320_a.vhd, and the appsfpga_io_400_a.vhd, which operate at 200 MHz, 320 MHz, and 400 MHz respectively.

NOTE: Previous versions of the APPSFPGA and [DLPC410 + DLPR410 + DMD] supported 200 MHz, 320 MHz, and 400 MHz DCLK frequencies. Although appsfpga_io_200_a.vhd, appsfpga_io_320_a.vhd, and appsfpga_io_400_a.vhd are all included in the APPSFPGA source code, the most recent DLPR410 revision A [DLPC410 + DLPR410A + DMD] only supports operation at the 400 MHz DCLK frequency.

The test pattern data and control signals are sent from the "appcore" block and clocked out through the "appsfpga_io" block. A simple multiplexer will substitute the required training pattern data whenever the incoming top level “init_active” signal is detected.

The output data is reformatted and sent to the DLPC410 using the Xilinx 4:1 oserdes primitives. All of the output signals (even the lower speed outputs) use the output serdes structure. Since the oserdes are implemented in the hardened IO ring of the FPGA, this ensures that the clock-output timing will be very close across all of the outputs.

4 "appcore" Block

The "appcore" block consists of the PGen block and the GUI block.

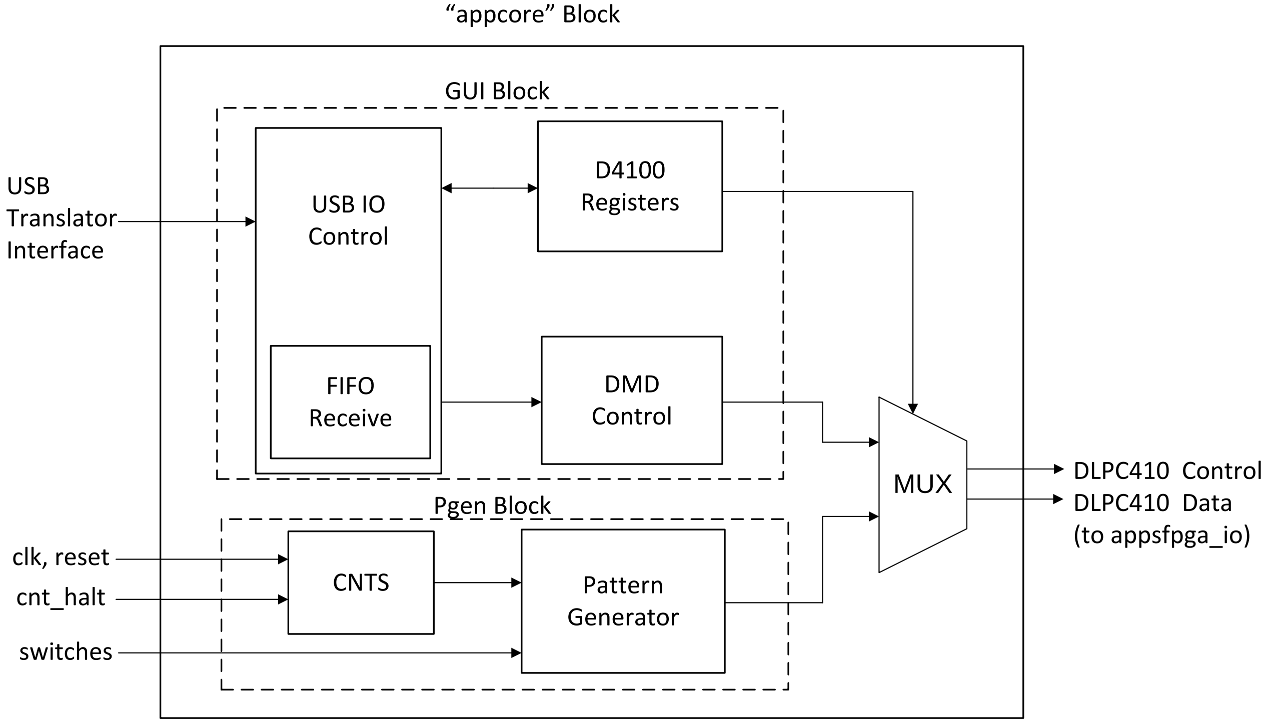
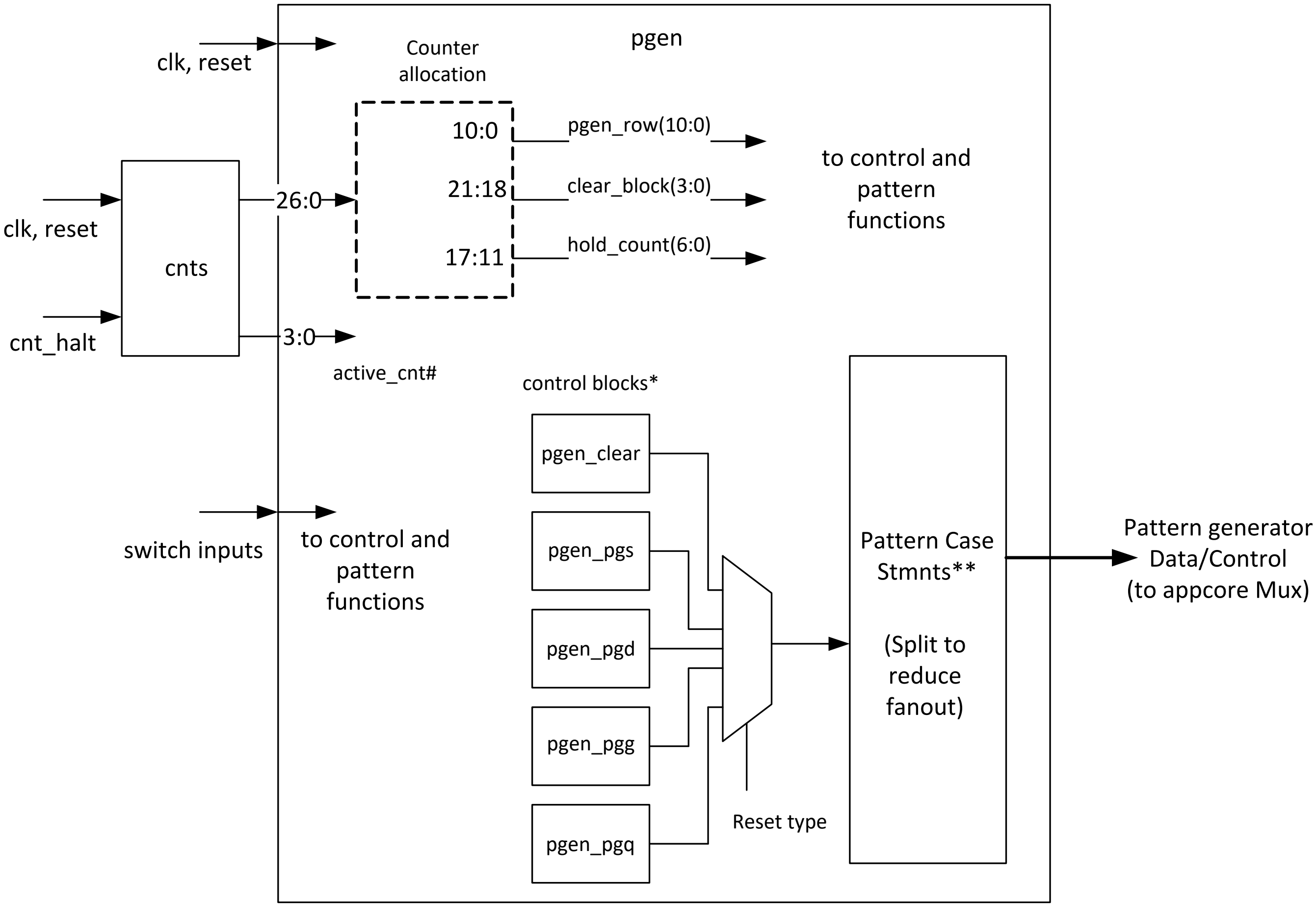


Figure 3. "appcore" Block

4.1 PGen Block

The "PGen" block handles the pattern and control generation functions of the design and is made up of the "cnts" and "pgen" sub-blocks.



* These blocks generate control signals for the various reset and clear modes

** This process generates the data for the various test patterns

# This count defines the various segments of active rows

Figure 4. PGen Block

4.1.1 "cnts" Sub-block

The “cnts” sub-block consists of two counters that drive all of the other functions in the design. A small 4-bit counter (cnts_active_cnt) is used to tell the PGEN logic which segment of the currently active line it should be generating. A larger 27-bit counter (cnts_pattern_cnt) is used to keep track of the pattern and row counts, generate clears and resets, create correct spacing, etc. Activating the “count halt” switch will freeze this counter, effectively freezing the image. Five variations of the “cnts” block are provided that generate different spacing between the patterns/active rows/resets.

cnts_a_1and0clks.vhd places zero idle clocks between lines.

cnts_a_1and2clks.vhd places two idle clocks between lines.

cnts_a_1and16clks.vhd places sixteen idle clocks between lines.

cnts_a_1and32clks.vhd places thirty-two idle clocks between lines.

cnts_a_1and64clks.vhd places sixty-four idle clocks between lines.

NOTE: The APPSFPGA on the DLPLCRC410EVM Controller board is instantiated with a compiled APPSFPFGA code version which places thirty-two idle clocks between lines (using cnts_a_1and32clks.vhd).

NOTE: Since the internal clock rate of the target subsystem (DLPC410 + DMD) is one half the clock rate of the DLPC410 input clock (DCLK), any spacing the APPSFPGA places between consecutive rows of data to the DLPC410 must be an even number of DCLK clocks.

4.1.2 "pgen" Sub-block

The "pgen" sub-block uses these counter values to generate the pattern data and control signals that are sent to the DLPC410. The 27 bit count value is received from the cnts block and is split into three separate fields:

- cnts_pattern_cnt(10:0) = pgen_row(10:0)
- cnts_pattern_cnt(17:11) = hold_count(6:0)
- cnts_pattern_cnt(21:18) = clear_block(3:0)
- cnts_pattern_cnt(26:24) = pattern_index(2:0)

NOTE: (23:22) are not used.

When looking at any of the counter values used in these blocks, it is important to remember that the clock is running at half of the base clock frequency, so many count values will be half of what might be expected.

The clear and reset commands are generated by five blocks. The outputs of these blocks feed into a multiplexer and the “reset type” selection (and the pattern type) determines which block’s output is used. The blocks and their functions are:

- pgen_clear : generates global clear instructions
- pgen_pgs : generates single block reset instructions
- pgen_pgd : generates dual block reset instructions
- pgen_pgq : generates quad block reset instructions
- pgen_pgg : generates global reset instructions

A large process that includes two of case statements is used to generate the actual output pattern data for the A, B, C, and D data buses. The case statement was split into two sections to reduce fan out and help reduce timing closure issues. The active_cnt and a pipelined version of pgen_row signals are used to guide the data generation process.

As previously mentioned, it is important to remember that the clock being used in the data generation processes is half of the target clock frequency. Since the DDR data output is clocked on both the rising and falling edges at the target frequency, this means that the data generated on the half speed clock must be four times as wide as the data presented at the outputs. Ex: 64 bits at 200 MHz = 16 bits DDR at 400 MHz.

The patterns displayed by the pgen block are:

- Single pixel border box with small box in the pin-1 corner
- Thin vertical lines
- Small checkerboard
- Alternating thin and thick vertical lines
- Diagonal lines
- Horizontal herringbone
- Black screen (with global clear)

4.2 GUI Block

The GUI block receives data and control from a PC based GUI over a USB interface and provides the data and control information to the DLPC410 data interface in place of Pgen data and control.

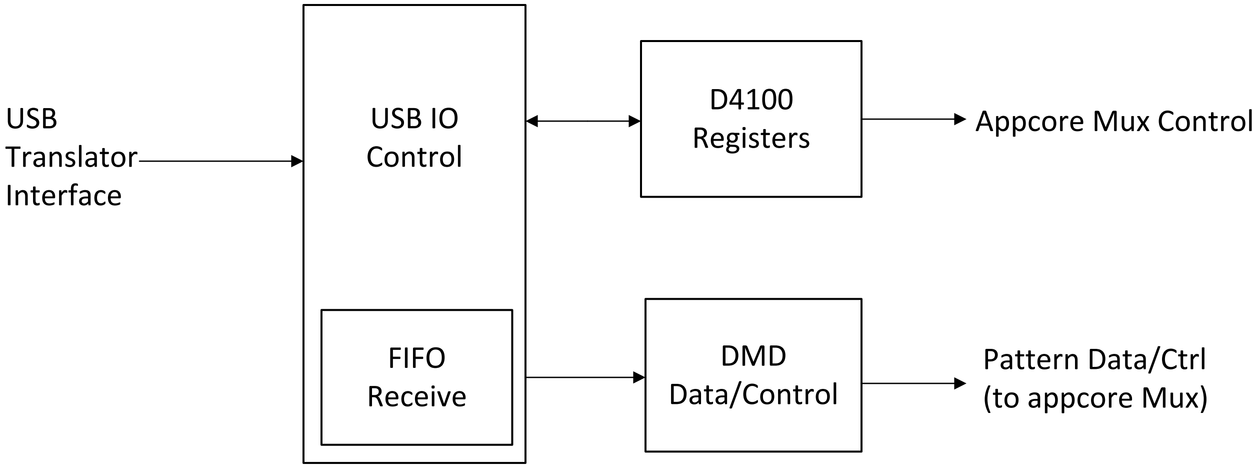


Figure 5. GUI Block

4.2.1 USB IO Control and D4100 Registers

The USB IO Control block provides USB read and write access from the PC based GUI to the controls and data interfaces within the APPSFPGA. The block accumulates the status of the D4100 registers and provides them to the GUI upon read accesses across the USB interface. It also receives incoming USB write accesses and loads the pertinent control information into the D4100 registers. Through the D4100 registers, one of the major functions is to allow the GUI to control the source of binary pattern data from the APPSFPGA to the DLPC410. Through register settings, this can be selected to be the output of the Pattern generator block (default), or for the data to be provided via USB from the GUI.

A mirrored soft copy of the DLPC410 on-board switches SW1(1-8) are contained in the D4100 registers. These registers provide an override capability of the switches to enable software control of these functions. There also exists a register to allow software override and selection of the pattern selections listed in Section 4.1.1.

4.2.1.1 USB Translator Interface

The USB Translator Interface is a section of VHDL code which translates the incoming and outgoing USB data as converted by the Cypress CY7C68013A device or the APPSFPGA. On one side of the Cypress device is the USB interface to the PC running the GUI. On the other side is the interface to/from the APPSFPGA. This interface logic utilizes the following signals for communication between the Cypress device and the APPSFPGA:

- ctl(2:0) - three control signals to identify the transaction
- 16 Bit Bi-directional Data Bus

Commands and Data are transferred across the 16 bit data bus. Command word decoding is performed by the USB Translator Interface to determine the type of transfer to take place. The encoding of this bus us shown in Table 2

Table 2. Control Signal Encoding

Signal	Idle	Address Latch	Data Write	Data Read	FIFO Burst[1]
ctl2	1	0	0	0	1
ctl1	1	1	1	0	1
ctl0	1	1	0	1	0

[1] Last word of data during burst not written.

4.2.2 FIFO Receive (FIFO RCV) and DMD Control

The FIFO RCV function provides a memory storage location for binary pattern data from the GUI. The GUI sends the pattern data blocks at a time to the APPSFPGA. The FIFO RCV block buffers the data prior to sending it across the 2xLVDS data buses to the DLPC410. The DMD Control block ensures the reading from the FIFO and sending to the DLPC410 includes the appropriate control signal framing for the selected modes of operation. Once the Data and Control information is successfully provided to the DLPC410, the DLPC410 will enable the binary pattern data to appear on the DMD micromirrors.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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, 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 (www.ti.com/legal/termsofsale.html) 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.

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