TI DN 2510445

August 2009

DDC4100 GUI/MEM Applications FPGA Sample Code Guide

This document describes the interface of the Texas Instruments DLP® Discovery™ 4100 chipset and gives an example Applications FPGA design that drives the DDC4100 system via USB/GUI interface and BIST testing of the DDR2 Memory Interface.

Revisions		
Rev	Descriptions	Date
A	Initial release	August 2009

IMPORTANT NOTICE

BEFORE USING TECHNICAL INFORMATION, THE USER SHOULD CAREFULLY READ THE FOLLOWING TERMS.

The term "Technical Information" includes reference designs, drawings, specifications, and other information relating to TI DLP® products or applications, contained herein or provided separately in any format or via any medium.

TI is providing Technical Information for the convenience of purchasers of DLP® products ("Users"), and will not accept any responsibility or liability arising from providing the Technical Information or its use. Any use or reliance on Technical Information is strictly the responsibility of the User.

1. **No Warranty.** ***THE TECHNICAL INFORMATION IS PROVIDED "AS IS".*** TI MAKES NO WARRANTIES OR REPRESENTATIONS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING LACK OF VIRUSES, ACCURACY, OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE TECHNICAL INFORMATION OR THE USE OF THOSE MATERIALS.

2. **Warranty for Products Not Affected.** The foregoing exclusion and disclaimer of warranty does not affect or diminish any warranty rights with regard to DLP® products. Such rights are governed exclusively by the terms of a written and signed purchase agreement with TI.

3. **Limitations and Exclusion of Damages.** IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF THE TECHNICAL INFORMATION OR THE USE OF THE TECHNICAL INFORMATION.

4. **No Engineering Services.** User is fully responsible for all design decisions and engineering with regard to its products, including decisions relating to application of DLP® products. By providing Technical Information TI does not intend to offer or provide engineering services or advice concerning User's design. If User desires engineering services, then User should rely on its retained employees and consultants and/or procure engineering services from a licensed professional engineer ("LPE").

5. **Compliance with Export Control Laws.** Unless prior authorization is obtained from the U.S. Department of Commerce, User may not export, re-export, or release, directly or indirectly, any Technical Information, or export, directly or indirectly, any direct product of such Technical Information to any destination or country to which the export, re-export or release of the Technical Information or direct product is prohibited by the Export Administration Regulations of the U.S. Department of Commerce ("EAR").

Abbreviations and Acronyms

The following lists abbreviations and acronyms used in this manual.

APPSFPGA	Xilinx Virtex 5 Field Programmable Gate Array for customer applications
CDS	Customer Data Sheet
DAD2000	DMD Power and Reset Driver
D4100	Discovery™ 4100
dc	Direct Current
DDR	Double Data Rate
DMD	Digital Micromirror Device
DLP	Digital Light Processing
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DVI	Digital Video Interface
EMI	Electromagnetic Interference
FCC	Federal Communications Commission
fps	Frames per Second
FPGA	Field Programmable Gate Array
Knowledge Base	Texas Instruments Extranet providing Discovery™ documentation, available after purchase only
LED	Light Emitting Diode
PROM	Programmable Read Only Memory
SCP	Serial Communications Port
SRAM	Static Random Access Memory
USB	Universal Serial Bus

Table of Contents

Table of Figures

Table of Tables

1 Overview

This document is a basic guide to the design of an Applications FPGA (AppsFPGA) that drives the DDC4100 chip. It explains the interface between the AppsFPGA and the DDC4100 and gives an example design (provided with Discovery 4100 starter kit). Figure 1 shows the system overview of an example design using the DDC4100.

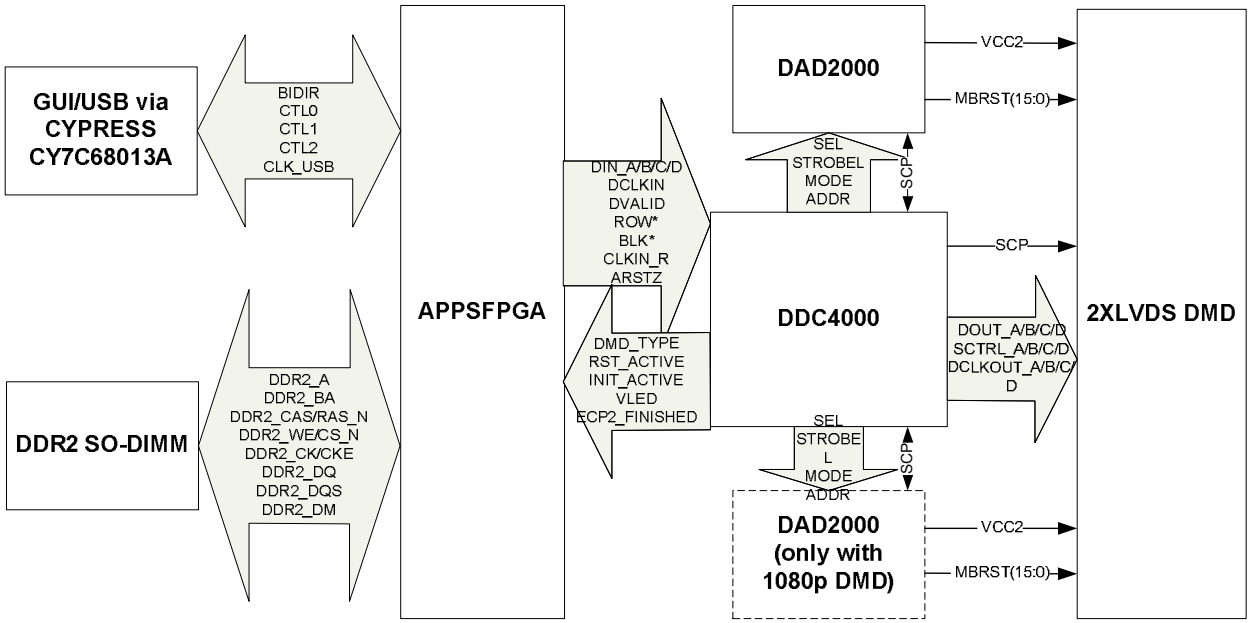

Figure 1 System Overview of Example Design

The GUI/MEM AppsFPGA contains the GUI/USB and DDR2 SO-DIMM memory interface Applications FPGA Sample Code for the DDC4100. The GUI and memory interface designs are mutually exclusive. The GUI interface design enables the Explorer 4100 GUI software to drive the DDC4100 and DMD. The sample code also contains a DDR2 memory BIST and controller for testing the DDR2 SO-DIMM interface as shown in Figure 2 below.

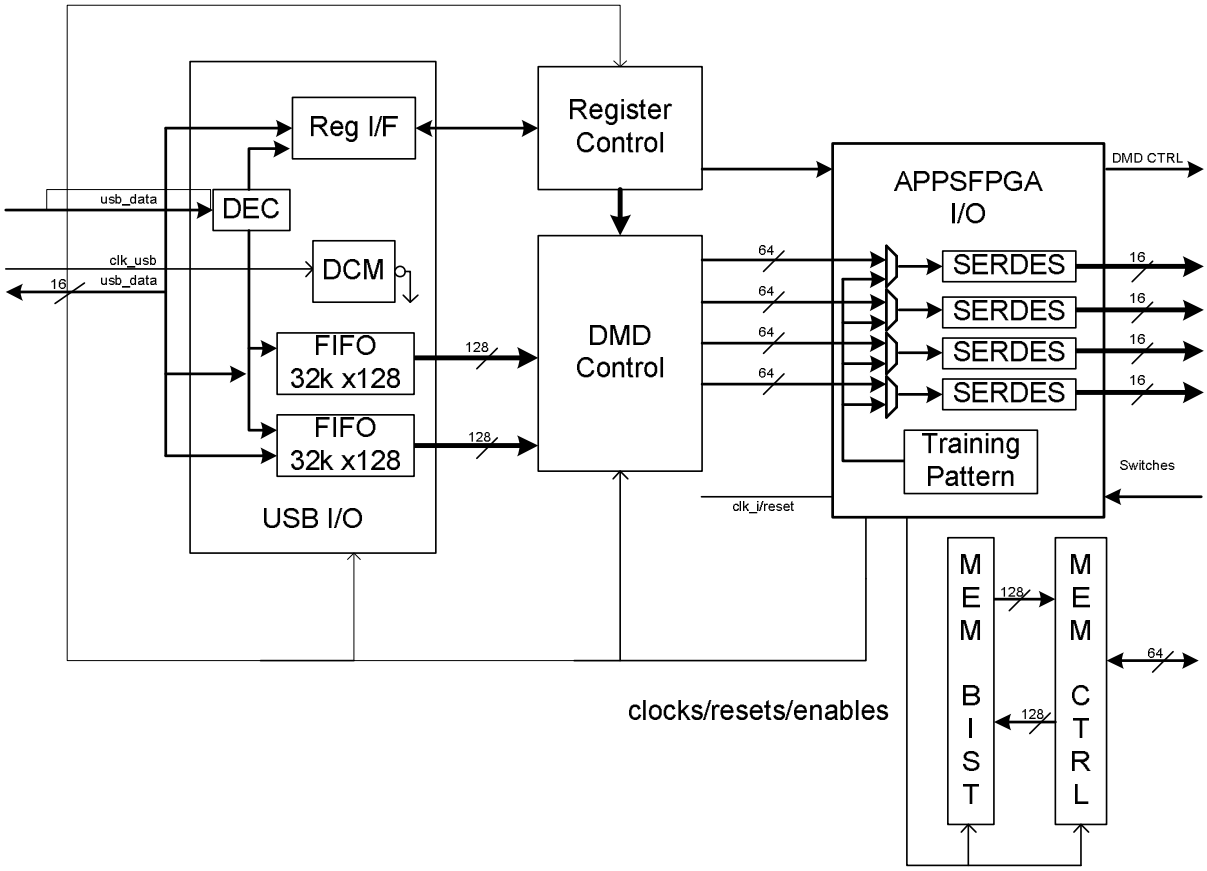

Figure 2 GUI/MEM APPS FPGA Block Diagram

2 Inputs and Outputs

Table 1describes the inputs and outputs of the GUI/MEM applications FPGA. Most of the output signals are part of the DDC 4100 interface. For more details on these signals, see the DDC 4100 data sheet.

Table 1 AppsFPGA Input/Output Description

Signal Name	IN/OUT/ INOUT	Description
clk_i	IN	Input clock (50 MHz)
reset_i	IN	Active high, asynchronous system reset (controlled by slide switch)
apps_logic_rstz	IN	Active low, asynchronous system reset (connected to push-button switch)
apps_mirror_floatz	IN	Float all mirrors in preparation for system shutdown (connect to push-button switch)
finished_iv_o	OUT	Indicates when applications FPGA has finished initialization (connected to LED)
in_rst_active_i	IN	Asserted while a mirror reset is being executed
in_init_active_i	IN	Asserted while DDC 4100 is initializing
in_dip_sw_i[7:0]	IN	Dip switch inputs
finished_iv_o	OUT	Indicates when applications FPGA has finished initialization (connected to LED)
clk_r_o	OUT	Reference clock to DDC4100 (50MHz)
arstz_o	OUT	Reset to DDC4100
dout_ap_o[15:0]	OUT	LVDS p output data A to DDC4100
dout_an_o[15:0]	OUT	LVDS n output data A to DDC4100
dout_bp_o[15:0]	OUT	LVDS p output data B to DDC4100
dout_bn_o[15:0]	OUT	LVDS n output data B to DDC4100
dout_cp_o[15:0]	OUT	LVDS p output data C to DDC4100
dout_cn_o[15:0]	OUT	LVDS n output data C to DDC4100
dout_dp_o[15:0]	OUT	LVDS p output data D to DDC4100
dout_dn_o[15:0]	OUT	LVDS n output data D to DDC4100
dclk_ap_o	OUT	LVDS p output data clock A to DDC4100

dclk_an_o	OUT	LVDS n output data clock A to DDC4100
dclk_bp_o	OUT	LVDS p output data clock B to DDC4100
dclk_bn_o	OUT	LVDS n output data clock B to DDC4100
dclk_cp_o	OUT	LVDS p output data clock C to DDC4100
dclk_cn_o	OUT	LVDS n output data clock C to DDC4100
dclk_dp_o	OUT	LVDS p output data clock D to DDC4100
dclk_dn_o	OUT	LVDS n output data clock D to DDC4100
dvalid_ap_o	OUT	LVDS p output data valid A to DDC4100 used to qualify data
dvalid_an_o	OUT	LVDS n output data valid A to DDC4100 used to qualify data
dvalid_bp_o	OUT	LVDS p output data valid B to DDC4100 used to qualify data
dvalid_bn_o	OUT	LVDS n output data valid B to DDC4100 used to qualify data
dvalid_cp_o	OUT	LVDS n output data valid C to DDC4100 used to qualify data
dvalid_cn_o	OUT	LVDS p output data valid C to DDC4100 used to qualify data
dvalid_dp_o	OUT	LVDS n output data valid D to DDC4100 used to qualify data
dvalid_dn_o	OUT	LVDS p output data valid D to DDC4100 used to qualify data
rowmd_o[1:0]	OUT	Output row mode to DDC4100
rowad_o[10:0]	OUT	Output row address to DDC4100
stepvcc_o	OUT	Output step vcc
comp_data_o	OUT	Output to cause DDC4100 to complement all data
ns_flip_o	OUT	Output to cause DDC4100 to reverse order of row loading
blkad_o	OUT	Output block address to DDC4100
blkmd_o	OUT	Output block mode to DDC4100
wdt_enablez_o	OUT	Output watch dog timer
ddc_version_i	IN	DDC version information from DDC4100.
dmd_type_i	IN	DMD type information from DDC4100 (1080P, .55 XGA, .7 XGA, etc.)
pwr_floatz_o	OUT	Float all mirrors command to DDC4100 in preparation for system shutdown
rst2blkz_o	OUT	Output RST2BLK to DDC4100
led0	OUT	Unused active low LED
led1	OUT	DDR2 BIST status

ddr2_a[13:0]	OUT	DDR2 SO-DIMM Address (2GB)
ddr2_ba[2:0]	OUT	DDR2 SO-DIMM Bank Address
ddr2_ras_n	OUT	DDR2 SO-DIMM Row Address Select (Active Low)
ddr2_cas_n	OUT	DDR2 SO-DIMM Column Address Select (Active Low)
ddr2_we_n	OUT	DDR2 SO-DIMM Write Enable (Active Low)
ddr2_cs_n[1:0]	OUT	DDR2 SO-DIMM Chip Select (Active Low)
ddr2_odt[1:0]	OUT	DDR2 SO-DIMM On-Die Termination
ddr2_cke[1:0]	OUT	DDR2 SO-DIMM Clock Enable
ddr2_ck[1:0]	OUT	DDR2 SO-DIMM Differential Clock P (150 MHz)
ddr2_ck_n[1:0]	OUT	DDR2 SO-DIMM Differential Clock N (150 MHz)
ddr2_dq[63:0]	INOUT	DDR2 SO-DIMM Data
ddr2_dqs[7:0]	OUT	DDR2 SO-DIMM Differential Data Strobe P
ddr2_dqs_n[7:0]	OUT	DDR2 SO-DIMM Differential Data Strobe N
ddr2_dm[7:0]	OUT	DDR2 SO-DIMM Data Mask
ddr2_scl	IN	DDR2 SO-DIMM Serial Clock (unused)
ddr2_sda	INOUT	DDR2 SO-DIMM Serial Data (unused)
clk_usb	IN	IFCLK from Cypress CY7C68013A (48MHz)
rdy0	OUT	RDY0 tied high
rdy1	OUT	RDY1 tied high
rdy2	OUT	RDY2 (tcexpire) tield low
ctl0	IN	USB Write Enable
ctl1	IN	USB Read Enable
ctl2	IN	USB Register Enable
gpio_a[2:0]	OUT	Unused
gpio_a_i[2:0]	IN	Unused
gpio_ext_rst_in	IN	Unused
gpio_reset_complete_o	OUT	Unused
bidir[15:0]	INOUT	USB Bidirectional Data
apps_testpt[30:0]	OUT	FPGA Test Points

A set of 8 dip switches are used to control the designs. Table 2 shows the assignment of these dip switches.

Table 2 Dip Switch Description

Switch Number	Effect
1	Inject error into the Memory BIST
2	Unused
3	Unused
4	Unused
5	Unused
6	Unused
7	Unused
8	Unused

3 Functionality and Structure

The sample code is written hierarchically, with five levels, as illustrated in Figure 3. The top level module, *AppsFPGA*, instantiates the modules *AppsFPGA_io* and *AppsCore*.

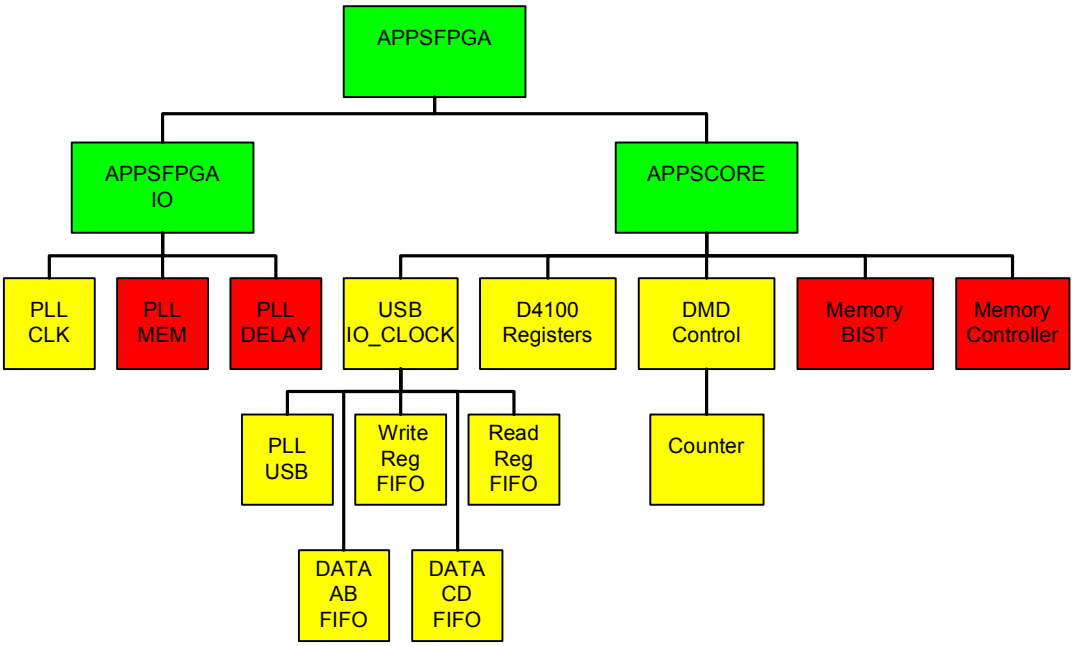

Figure 3 Sample GUI/MEM Applications FPGA Hierarchy

3.1 APPSFPGA_IO

The *AppsFPGA_IO* module contains the 3 Phase Locked Loops (PLLs) used in creating the system clocks for the DDC4100 and the clocks used in the DDR2 memory interface (see Table 3 below for details on clock speeds). This module also generates the system logic resets from the push button reset (apps_logic_rstz) with added debounce protection logic. The AppsFPGA_IO block also contains the ddr (4 to 1 SERDES) cells used to generate Double Data Rate (DDR) output data to drive the DDC4100 and the training pattern used to initialize this interface as described below in section 4.1 Initialization.

Table 3 PLL Clock Generation

PLL_CLK		
Clock	Speed	Phase Shift
clk0	200 MHz	-
clk2x	400 MHz	-
clk2x180	400 MHz	180 degrees
PLL_MEM		
Clock	Speed	Phase Shift
clk_mem	75 MHz	-
clk2x_mem	150 MHz	-
clk2x90_mem	150 MHz	90 degrees
PLL_DELAY		
Clock	Speed	Phase Shift
clock0_delay	200 MHz	-

3.2 APPSCORE

The *AppsCore* module instantiates five level-3 modules. These modules form two separate design entities. The first design contains the modules *USB_IO_Clock*, *D4100_registers* and *DMD_Control*, along with the AppsFPGA_io to form the USB/GUI interface to the DDC4100. The remaining two modules *Memory_BIST,* and *Memory Controller* create the DDR2 memory test design.

3.2.1 GUI APPS Design

The GUI APPS design consists of the three blocks *USB_IO_Clock*, *D4100_registers* and *DMD_Control* and the SERDES cells in the *AppsFPGA_IO* block. This design provides a sample of how USB can be used to interface the Explorer 4100 GUI software to send image data to the DDC4100.

3.2.1.1 USB_IO_Clock

The *USB_IO_Clock* module decodes the control signals from the Cypress CY7C68013A and routes the 16-bit data to either the data FIFOs or the D4100 registers. In the case of an XGA DMD the data goes entirely through the *Data AB FIFO*. With a 1080P DMD the data ping pongs back and forth between the *Data AB FIFO* and the *Data CD FIFO*. The control signals CTL0, CTL1 and CTL2 are decoded to determine the command issued by the USB interface as described in Table 4. The following table shows the encoding for the control signals. The data FIFOs are 32768x16 on the write side and 4096x128 on the read side. Data is read from the FIFOs 128 bits at a time at 200 MHz, for the AB channel and CD channel (in the case of 1080P operation) simultaneously to allow enough data to be provided to the 4 to 1 400 MHz SERDES to the DDC4100.

Table 4 Cypress Command Decode

CTL0 (FIFO_WEN)	CTL1(FIFO_REN)	CTL2 (FIFO_REGN)	Command
0	1	0	Register Write
1	0	0	Register Read
1	1	0	Register Address Setup
0	1	1	Data FIFO Write

3.2.1.2 D4100 Registers

The D4100_*Registers* module contains the USB R/W registers that control the USB/GUI interface to the DMD. Writes and reads of the register are a two step process. The first transaction sets the address to be written or read as designated by the command decoded from table 4 in the second transaction. During a register read, the data in the register is pre-fetched upon receipt of the register address setup command to be made available for when the read command arrives. Table 5 lists the registers available in the GUI APPS design and their addresses and functions:

Table 5 D4100 GUI Register Definitions

Address (Hex)	Data Bits	Bit descriptions	R/W
0x00	15:0	Discovery Version	R
0x01	15:0	APPSFPGA Code Version	R
0x02	15:0	ECHO	R/W
0x03	4:0	(4) FIFO Reset	W
		(3) Active Block Memory	
		(2) Block Reset	
		(1) Global Reset	
		(0) DMD Write Block	
0x10	3:0	DMD Type	R
0x11	2:0	DDC Version	R
0x14	1:0	Row Mode	R/W
0x15	10:0	Row Address	R/W
0x16	6:0	(6) RST2BLKZ	R/W
		(5) DMD External Reset	
		(4) Power Float	
		(3) Watch-Dog Timer	
		(2) North/South Flip	
		(1) Complement Data	
		(0) Step VCC	
0x17	1:0	Block Mode	R/W
0x18	3:0	Block Address	R/W
0x19	2:0	GPIO_out	R/W
0x20	15:0	DMD_RowLoads	R/W
0x21	0:0	Reset Complete	R
0x22	0:0	GPIO_reset_complete	W

3.2.1.3 DMD_Control

The *DMD_Control* module controls the timing of data being sent to the DMD. The timing is controlled by a state machine that is initiated by writing the DMD Write Block bit in register 0x03. Upon setting the write block bit the state machine loads a counter with the value loaded in the DMD_RowLoads register (0x20). Each time the state machine cycles through, it decrements this counter until the total number of rows to be loaded has reached zero. Each time a row is loaded to the DDC4100 the data is fetched from the data FIFOs in the *USB_IO_Clock* block. The data bit ordering must be manipulated so that it matches that required by the DDC4100, with respect to bit-order and AB or CD channel alignment. The state machine code is listed below.

```
IF system_reset = '1' THEN
  C_BLOCK_WRITE_STATE        <= S1;
  dvalid_f                   <= '0';
  row_write_pos_rst          <= '1';
  get_row_data          <= '0';
  decrement_row_load_counter <= '0';
ELSIF system_clk'EVENT and system_clk = '1' THEN
  CASE C_BLOCK_WRITE_STATE IS
    WHEN S1  =>
          IF ddc_init_active = '0' THEN
            C_BLOCK_WRITE_STATE <= S2 AFTER 1 PS;
          END IF;
    WHEN S2 =>
          IF rows_to_load > x"0000" THEN
            decrement_row_load_counter <= '1' AFTER 1 PS;
            C_BLOCK_WRITE_STATE <= S3 AFTER 1 PS;
          END IF;
    WHEN S3 =>
          decrement_row_load_counter <= '0'  AFTER 1 PS;
          IF (outclkphase = '1') THEN     -- force even # clocks between DVALID
            C_BLOCK_WRITE_STATE <= S4 AFTER 1 PS;
          END IF;
    WHEN S4  =>
          C_BLOCK_WRITE_STATE <= S5 AFTER 1 PS;
    WHEN S5  =>
          get_row_data <= '1' AFTER 1 PS;
          dvalid_f <= '1' AFTER 1 PS;
          row_write_pos_rst   <= '0' AFTER 1 PS;
          if (row_write_pos_cnt >= x"006") THEN
            if dmd_1080p_connected_2q = '1' AND every_other_row = '1' then
                    get_row_data <= '0';
            end if;
            C_BLOCK_WRITE_STATE <= S6 AFTER 1 PS;
          END IF;
    WHEN S6  =>
          row_write_pos_rst   <= '1' AFTER 1 PS;
          dvalid_f <= '0' AFTER 1 PS;
          get_row_data <= '0' AFTER 1 PS;
          C_BLOCK_WRITE_STATE <= S7 AFTER 1 PS;
   WHEN S7 =>
          dvalid_f <= '0' AFTER 1 PS;
          C_BLOCK_WRITE_STATE <= S8;
    WHEN s8 =>
          C_BLOCK_WRITE_STATE <= S9;
    WHEN s9 =>
      C_BLOCK_WRITE_STATE <= s1;
    WHEN OTHERS =>
          C_BLOCK_WRITE_STATE <= S1 AFTER 1 PS;
  END CASE;
END IF;

dmd_get_row_data <= get_row_data;
```

In the case of 1080P (1920x1080) operation, the DMD requires 2048 pixels to be loaded even though there are only 1920 visible pixels. To account for the extra bits, logic was added to insert 64 bits of zeros at the beginning and end of each row. The data retrieved from the AB and CD FIFOs is 128 bits wide (two 64 bit words). Because of the zero padding only one 64 bit word from a given FIFO retrieval is used, thus requiring the other 64 bit word to be stored for the next data transfer along with one of the 64 bit data from the next FIFO data retrieval. Table 6 illustrates how the data out of the FIFO is used.

Table 6 1080P Zero Pad Illustration

FIFO location	< 128 bit FIFO data>	
4	I	J
3	G	H
2	E	F
1	C	D
0	A	B

DMD Row Cycle #					
0	1	2	3	4	5
ZERO	B	D	F	H	J
A	C	E	G	I	…

The code demonstrating the zero padding is shown below.

```
if dmd_1080p_connected_2q = '1' then
    if get_row_data = '1' then
            fifo_ab_data_out_1q <= fifo_ab_data_out;
            fifo_cd_data_out_1q <= fifo_cd_data_out;
    end if;
   --1080p connected, got to zero pad
   if every_other_row = '0' then
            if row_write_pos_cnt = x"000" then
                    dmd_ab <= x"0000000000000000" & fifo_ab_data_out(127 downto 64);
                    dmd_cd <= fifo_cd_data_out;
            elsif row_write_pos_cnt = x"007" then
                    dmd_ab <= fifo_ab_data_out_1q(63 downto 0) & fifo_ab_data_out(127 downto 64);
                    dmd_cd <= fifo_cd_data_out(127 downto 64) & x"0000000000000000";
                    every_other_row <= '1' after 1 ps;
            else
                    dmd_ab <= fifo_ab_data_out_1q(63 downto 0) & fifo_ab_data_out(127 downto 64);
                    dmd_cd <= fifo_cd_data_out;
            end if;
   else
            if row_write_pos_cnt = x"000" then
                    dmd_ab <= x"0000000000000000" & fifo_ab_data_out_1q(63 downto 0);
                    dmd_cd <= fifo_cd_data_out_1q(63 downto 0) & fifo_cd_data_out(127 downto 64);
            elsif row_write_pos_cnt = x"007" then
                    dmd_ab <= fifo_ab_data_out_1q;
                    dmd_cd <= fifo_cd_data_out_1q(63 downto 0) & x"0000000000000000";
                    every_other_row <= '0';
            else
                    dmd_ab <= fifo_ab_data_out_1q;
                    dmd_cd <= fifo_cd_data_out_1q(63 downto 0) & fifo_cd_data_out(127 downto 64);
            end if;
```

```
    end if;
else
   --XGA DMD connected
    dmd_ab <= fifo_ab_data_out;
end if;
```

The data coming out of the FIFO is aligned 16 bits at a time alternating between the A channel data and the B channel data (like wise for the CD channels). Thus the A and the B channel data must be grouped together respectively before being sent to the 4 to 1 SERDES. The following code demonstrates this data manipulation.

```
dmd_ab_swap <= dmd_ab(127 DOWNTO 112) & dmd_ab(95 DOWNTO 80) & dmd_ab(63 DOWNTO 48)
                & dmd_ab(31 DOWNTO 16) & dmd_ab(111 DOWNTO 96) & dmd_ab(79 DOWNTO 64)
                & dmd_ab(47 DOWNTO 32) & dmd_ab(15 DOWNTO 0);

dmd_cd_swap <= dmd_cd(127 DOWNTO 112) & dmd_cd(95 DOWNTO 80) & dmd_cd(63 DOWNTO 48)
                & dmd_cd(31 DOWNTO 16) & dmd_cd(111 DOWNTO 96) & dmd_cd(79 DOWNTO 64)
                & dmd_cd(47 DOWNTO 32) & dmd_cd(15 DOWNTO 0);

dmd_dout_a <= dmd_ab_swap(127 DOWNTO 64);
dmd_dout_b <= dmd_ab_swap(63  DOWNTO  0);
dmd_dout_c <= dmd_cd_swap(127 DOWNTO 64);
dmd_dout_d <= dmd_cd_swap(63  DOWNTO  0);
```

The four 64 bit data busses (A, B, C, and D) are now sent to the DDC4100 via the 4 to 1 SERDES like the one shown below for the A channel.

```
-- create 16 ddr_lvds_io for each channel a,b,c,d
dat_gen_loop:
FOR i IN 0 TO 15 GENERATE
  BEGIN

      data_a_io : ddr_lvds_io
      PORT MAP (
    d1  => temp_dout_a_q(63-i),
    d2  => temp_dout_a_q(47-i),
    d3  => temp_dout_a_q(31-i),
    d4  => temp_dout_a_q(15-i),
    dout_dpp => appsfpga_io_dout_ap(i),    -- diff_p output (connect directly to top-level port)
    dout_dpn => appsfpga_io_dout_an(i),    -- diff_n output (connect directly to top-level port)
    clk2X => clk2x_b,
    clk1X => clk_b,
    reset => reset
    );
```

3.2.2 MEM APPS Design

The DDR2 MEM APPS FPGA design contains sample code for testing a 2GB DDR2 SO-DIMM module at 150 MHz with a burst length of four. The DDR2 module has a 64-bit data interface. The target device used in this design is the MT16HTF25664HY-667. The design consists of two blocks: *Memory_BIST* and *Memory Controller*.

3.2.2.1 Memory_Controller

The Memory_Controller used in this sample design was generated using the Xilinx Memory Interface Generator 2.3 (MIG 2.3). The parameters used in generating this controller can be found in Appendix A of this document and also in the mig.prj file located with the sample vhdl code.

Further information regarding implementing a DDR2 SO-DIMM memory interface with MIG 2.3 can be found at http://www.xilinx.com/support/documentation/ip_documentation/ug086.pdf.

3.2.2.2 Memory_BIST

The *Memory_BIST* block contains sample self-checking code that writes every address of the 2GB DDR2 module using pseudo-random address and data generators. After writing each address location, each address is then read using the same pseudo-random address generator pattern. The resulting data read is compared to data generated by an identical pseudo-random generator used in creating the write data and an error flag is set accordingly. The error status flag is connected to LED1 on the D4100 board.

When targeting the 2GB memory 28 bits of the 31 bit user address are used. The following is a map of how the user address bits map to the DDR2 memory address space. Identical mappings are implemented for the write and read address generators.

“000”	: Address(30:28)
1 Chip Select Bit (ddr2_cs_n)	: Address(27)
3 Bank Address Bits (ddr2_ba)	: Address(26:24)
14 Row Address Bits (ddr2_a)	: Address(23:10)
10 Column Address Bits (ddr2_a)	: Address(9:0)

The following code demonstrates how the above mapping is implemented utilizing a 26 bit pseudo-random address generator.

```
wr_addr       <= wr_addr(24 downto 0) & (wr_addr(25) XOR wr_addr(5) XOR wr_addr(1) XOR wr_addr(0))

wr_cs_bits    <= wr_addr(25);
wr_bank_bits <= wr_addr(24 DOWNTO 22);
wr_row_bits   <= wr_addr(21 DOWNTO 8);
wr_col_bits   <= wr_addr(7 DOWNTO 0) & "00";

address       <= "000" & wr_cs_bits & wr_bank_bits & wr_row_bits & wr_col_bits;
```

* Note: Address(1:0) are fixed at “00” given the Memory Controller is used with a burst length of 4 (thus the need for only a 26 bit generator), thus driven with two 128 bit words.

To target a different size memory such as a 256MB memory the address mapping could be changed to a 25 bit user address. that utilizes a 23 bit pseudo-random address generator.

“000000”	: Address(30:25)
2 Bank Address Bits (ddr2_ba)	: Address(24:23)
13 Row Address Bits (ddr2_a)	: Address(22:10)
10 Column Address Bits (ddr2_a)	: Address(9:0)

The following code demonstrates how the above mapping is implemented utilizing a 23 bit pseudo-random address generator.

```
wr_addr        <= wr_addr(21 downto 0) & (wr_addr(22) XOR wr_addr(17));

wr_bank_bits  <= wr_addr(22 DOWNTO 21);
wr_row_bits   <= wr_addr(20 DOWNTO 8);
wr_col_bits   <= wr_addr(7 DOWNTO 0) & "00";

address        <= "000000" & wr_bank_bits & wr_row_bits & wr_col_bits;
```

* Note: Address(1:0) are fixed at “00” given the Memory Controller is used with a burst length of 4 (thus the need for only a 23 bit generator), thus driven with two 128 bit words.

The data bus is implemented with 2 64-bit pseudo-random data generators with identical generators used for both the write data and the compare data.

```
bist_app_wdf_data_tmp <= bist_app_wdf_data_tmp(126 DOWNTO 64) & (bist_app_wdf_data_tmp(127)
                         XOR bist_app_wdf_data_tmp(126) XOR bist_app_wdf_data_tmp(124) XOR
                         bist_app_wdf_data_tmp(123)) &
                         bist_app_wdf_data_tmp(62 DOWNTO 0) &
                         (bist_app_wdf_data_tmp(63) XOR bist_app_wdf_data_tmp(62) XOR
                         bist_app_wdf_data_tmp(60) XOR bist_app_wdf_data_tmp(59));
```

4 Modes of Operation

4.1 Initialization

The init_active (Table 1) signal indicates that the DMD, DAD, and DDC are in an initialization state after power is applied. During this initialization period, the DDC is initializing the DMD and DAD(s) by setting all internal registers to their correct states. When this signal goes low, the system has completed initialization. System initialization takes approximately 5 ms to complete. Data and command write cycles should not be asserted during the initialization. This signal is driven by a CLK_R register and should be considered an asynchronous signal. Standard synchronization techniques should be applied if monitoring this signal with a synchronous circuit clocked by a clock other than CLK_R.

The DDC has an auto calibration feature that starts automatically after the system reset is de-asserted. A simple repeating test pattern must be supplied on all of the data inputs while the “init_active” output of the DDC4100 is high/active. The details of the training pattern are described in the DDC4100 document 2510443.

Below is an example of how this training pattern is applied:

```
dout_a <= "00000000000000000000000000000000111111111111111110000000000000000" WHEN (init_active_gq_int = '1') ELSE
appcore_dout_a_q;
dout_b <= "00000000000000000000000000000000111111111111111110000000000000000" WHEN (init_active_gq_int = '1') ELSE
appcore_dout_b_q;
dout_c <= "00000000000000000000000000000000111111111111111110000000000000000" WHEN (init_active_gq_int = '1') ELSE
appcore_dout_c_q;
dout_d <= "00000000000000000000000000000000111111111111111110000000000000000" WHEN (init_active_gq_int = '1') ELSE
appcore_dout_d_q;
dvalid <= "0100" WHEN (init_active_gq_int = '1') ELSE (appcore_data_valid_q & appcore_data_valid_q & appcore_data_valid_q &
appcore_data_valid_q);
```

The first dvalid should not be given until 64 clocks after the init_active signal is de-asserted.

In order to initialize the DMD properly, a *block clear* command followed by 2 consecutive No Operation commands are required. For more information, please refer to the DDC4100 document 2510443.

4.2 Power Down

To ensure long term reliability of the DMD, a shutdown procedure must be executed. Prior to power removal, assert the *pwr_float_o* (Table 1) signal and allow approximately 300µs for the procedure to complete. This procedure will assure the mirrors are in a flat state. For more details, please refer to the appropriate DMD document.

4.3 Driving the GUI interface to the DDC4100

In order to properly drive the DDC4100 a few important steps must be followed. The Explorer 4100 GUI software takes these requirements into account.

The first step is to load the image data into the data FIFOs. It is imperative to reset the FIFO pointers prior to loading new data into the FIFOs by setting the FIFO reset bit in register 0x03. The proper row mode, row address, block mode and block address values must also be set in the D4100 registers (see DDC4100 data sheet for proper mode settings). The number of rows to be loaded to the DMD must be set. Once each of the register values has been set then the DMD Write Block bit may be written, which enables the data to be fetched out of the FIFOs and sends it, along with the control signals to the DDC4100 via the 4 to 1 SERDES in the *APPSFPGA_IO* module. To prevent lost data or misaligned data it is best to have the amount of data loaded into the FIFO match the number of rows to be loaded to the DMD.

4.4 Memory BIST Operation

The Memory BIST is designed to run automatically with the removal of the system reset. Once reset has been removed the memory controller will begin a 4-stage calibration process. Upon the completion of the calibration process the phy_init_done signal out of the memory controller will be asserted. This signal will then enable the Memory BIST operation. The memory controller must successfully complete calibration for the BIST to be enabled.

Once enabled the BIST first writes address zero and then proceeds to run one complete cycle through the write address pseudo-random generator. Once the generator reaches its initial seed the BIST write is complete and then it moves on to the read side. The BIST will then read address zero followed by cycling through the read address pseudo-random generator until it reaches its initial seed. When read data is valid, it will be compared for errors, to the data in the pseudo-random read data generator. If an error occurs during any point in the read process, it will be latched and upon completion of the BIST will be checked to determine if the BIST passed or failed. A status signal is hooked to LED1. Table 6 describes the state of the LED during each state of the process.

Table 7 LED State Status

BIST Stage	**LED State**
Reset	OFF
Calibration Active	ON
Calibration Complete	OFF
BIST Active	OFF
BIST Pass	8 sec period 50% duty cycle
BIST Fail	4 sec ON followed by (4) 1 sec period 50% duty cycle

Note: Dip Switch 1 has been wired to insert an error into the data stream to verify BIST Fail operation.

The waveforms in figures 4 and 5 illustrate the LED patterns utilized on LED1 (D12) to signify BIST pass or fail status.

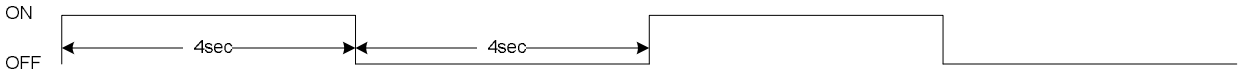

Figure 4 BIST Pass LED Pattern

Figure 5 BIST Fail LED Pattern

APPENDIX A MIG Project File Parameters


```
  <Version>2.3</Version>
   <SystemClock>Single-Ended</SystemClock>
   <IODelayHighPerformanceMode>HIGH</IODelayHighPerformanceMode>
   <Controller number="0" >
      <MemoryDevice>DDR2_SDRAM/SODIMMs/MT16HTF25664HY-667</MemoryDevice>
      <Frequency>150</Frequency>
      <DataWidth>64</DataWidth>
      <DeepMemory>2</DeepMemory>
      <DataMask>1</DataMask>
      <RowAddress>14</RowAddress>
      <ColAddress>10</ColAddress>
      <BankAddress>3</BankAddress>
      <TimingParameters>
         <Parameters tdha="300" tdhb="175" trfc="127.5" twtr="10" tis="400" tdqsq="240" twr="15" tac="450"
tjit_duty="125" trrd="7.5" tras="40" tfaw="37.5" tqhs="340" tjit="100" trtp="7.5" tdqsck="400" trc="55" tck_min="3000"
tmrd="2" tdsa="300" tdsb="100" trcd="15" twpre="0.35" tih="400" trp="15" txards="7" />
      </TimingParameters>
      <ECC>ECC Disabled</ECC>
      <BankSelection>
         <Bank Control="1" SysClk="0" Data="0" name="1" Address="1" wasso="19" />
         <Bank Control="1" SysClk="0" Data="1" name="16" Address="1" wasso="38" />
         <Bank Control="0" SysClk="0" Data="1" name="19" Address="0" wasso="38" />
         <Bank Control="1" SysClk="0" Data="1" name="20" Address="0" wasso="38" />
         <Bank Control="0" SysClk="1" Data="0" name="3" Address="0" wasso="19" />
         <Bank Control="0" SysClk="1" Data="0" name="4" Address="0" wasso="19" />
      </BankSelection>
      <mrBurstLength name="Burst Length" >4(010)</mrBurstLength>
      <mrBurstType name="Burst Type" >sequential(0)</mrBurstType>
      <mrCasLatency name="CAS Latency" >3(011)</mrCasLatency>
      <mrMode name="Mode" >normal(0)</mrMode>
      <mrDllReset name="DLL Reset" >no(0)</mrDllReset>
      <mrPdMode name="PD Mode" >fast exit(0)</mrPdMode>
      <mrWriteRecovery name="Write Recovery" >3(010)</mrWriteRecovery>
      <emrDllEnable name="DLL Enable" >Enable-Normal(0)</emrDllEnable>
      <emrOutputDriveStrength name="Output Drive Strength">Fullstrength(0)</emrOutputDriveStrength>
      <emrRTT name="RTT (nominal) - ODT" >75ohms(01)</emrRTT>
      <emrPosted name="Additive Latency (AL)" >0(000)</emrPosted>
      <emrOCD name="OCD Operation" >OCD Exit(000)</emrOCD>
      <emrDQS name="DQS# Enable" >Enable(0)</emrDQS>
      <emrRDQS name="RDQS Enable" >Disable(0)</emrRDQS>
      <emrOutputs name="Outputs" >Enable(0)</emrOutputs>
   </Controller>
</Project>
```

5 Related Documentation

This section lists related documents associated with the use of the DDC4100 Chipset. For more information, please visit the Knowledge Base.

Component Datasheets & Interface Drawings	
Document	**Drawing #**
DLP Discovery 4100 Starter Kit Assembly for .95" 1080p 2xLVDS Type A	2510450
DLP Discovery 4100 Starter Kit Assembly for .7" XGA 2xLVDS Type A	2510451
DLP Discovery 4100 Starter Kit Assembly for .55" XGA 2xLVDS Type X	2510452
.95" 1080p 2xLVDS Type A Datasheet	2509698
.95" 1080p Type A Mechanical ICD	2506491
.7" XGA 2x LVDS Type A Datasheet	2509699
.7" XGA 2x LVDS Type A Mechanical ICD	2507867
.55" XGA 2xLVDS Type X Data sheet	2509700
.55" XGA 2x LVDS Type X Mechanical ICD	2507499
DMD Discovery™ Digital Controller 4100 (DDC4100) Datasheet	2510443
DAD2000 (DMD Power & Reset Driver) Datasheet	2506593
DMD Glass Cleaning Procedure	2504640
DMD Handling Specification (Type-A)	2504641