TI DN 2510445

August 2009

DDC4100 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.

Revisions		
Rev	Descriptions	Date
A	Initial release	August 2009

TI DN 2510445

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 the DLP® Discover™ 4100 starter kit). Figure 1 shows the system overview of an example design using the DDC4100.

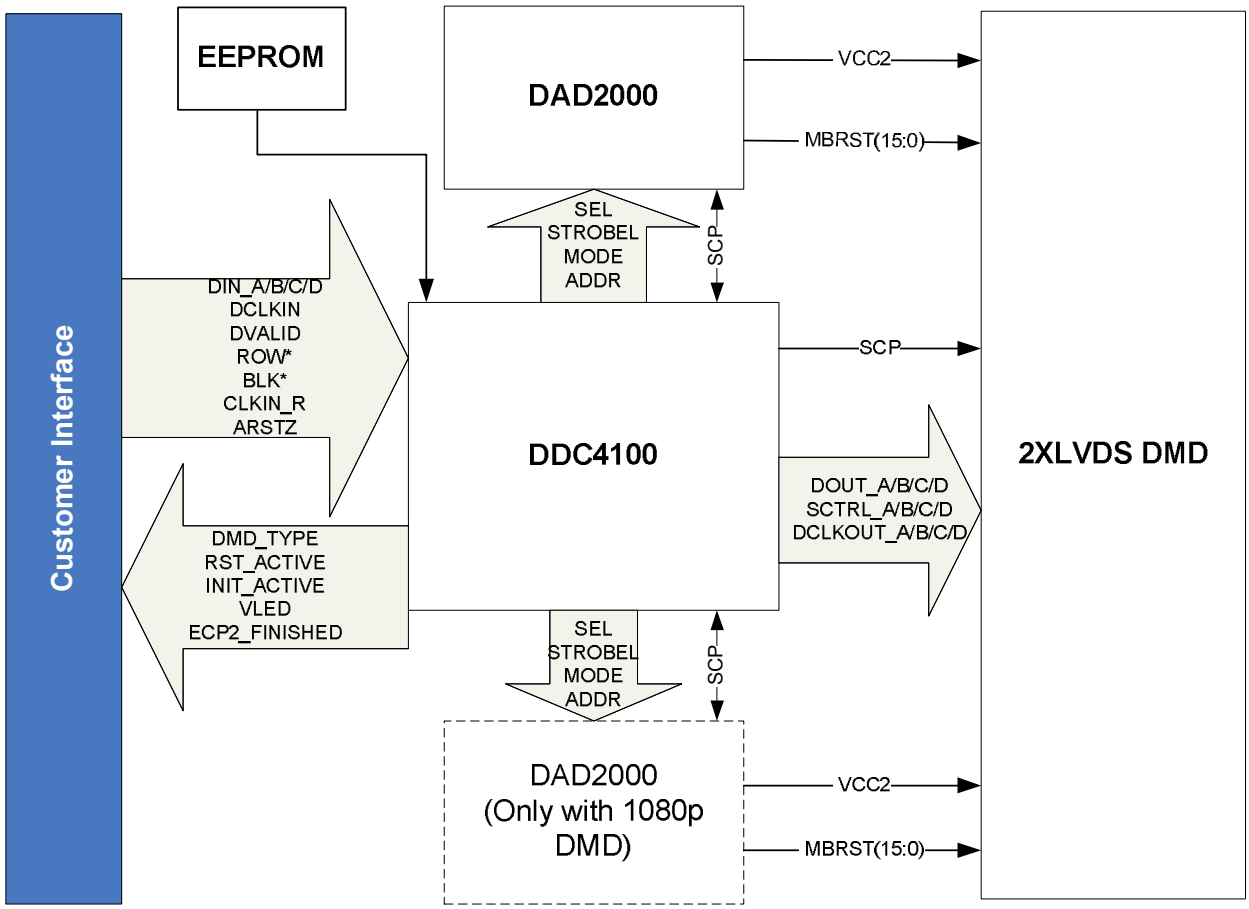

Figure 1. System Overview of Example Design

The APPSFPGA contains the Applications FPGA Sample Code for the DDC4100. This sample code cycles through test patterns and is meant to offer an example of code that meets the DDC4100 specification. It has been written to implement all features of the DDC4100, such as the complement function and all mirror reset types, as explained in later sections. This sample code also addresses additional operational requirements for the DDC4100 interface which should be observed.

2 Inputs and Outputs

Table 1 describes the inputs and outputs of the applications FPGA. Most of the output signals are part of the DDC4100 interface. For more details on these signals, see the DDC4100 data sheet.

Table 1. AppsFPGA Input/Output Description

Signal Name	Description
CLK_I	Input clock (50 MHz)
ARSTZ	Active low, asynchronous system reset (connected to flip switch)
IN_PWR_FLOAT_I	Float all mirrors in preparation for system shutdown (connect to push-button switch)
FINISHED_IV_O	Indicates when applications FPGA has finished initialization (connected to LED)
IN_RST_ACTIVE_I	Asserted while a mirror reset is being executed
IN_INIT_ACTIVE_I	Asserted while DDC4100 is initializing
IN_DIP_SW_I	Dip switch inputs
FINISHED_IV_O	Indicates when applications FPGA has finished initialization (connected to LED)
CLK_R	Reference clock to DDC4100 (50MHz)
DOUT_A[15:0]	Output data A to DDC4100 (400MHz DDR)
DOUT_B[15:0]	Output data B to DDC4100 (400MHz DDR)
DOUT_C[15:0]	Output data C to DDC4100 (400MHz DDR)
DOUT_D[15:0]	Output data D to DDC4100 (400MHz DDR)
DCLK_A	Output data clock to DDC4100 (400MHz)
DVALID_A	Output data valid to DDC4100 used to qualify data
ROWMD[1:0]	Output row mode to DDC4100
ROWAD[10:0]	Output row address to DDC4100
STEPVCC	Output to indicate status of vcc step
COMP_DATA	Output to cause DDC4100 to complement all data
NS_FLIP	Output to cause DDC4100 to reverse order of row loading
BLKAD	Output block address to DDC4100

Signal Name	Description
BLKMD	Output block mode to DDC4100
WDT_ENABLEZ	Output watch dog timer

A set of eight dip switches are used to control the DMD operations. Table 2 shows the assignment of these dip switches. For more information on the DMD operations, please refer to the appropriate DMD specification.

Table 2. Dip Switch Description

Switch Number	Effect
1	float – float all mirrors
2	counter halt – stop counter, this will freeze the image on the DMD
3	complement data – causes DDC4100 to complement all data it receives
4	north/south flip – causes the DDC4100 to reverse order of row loading, effectively flipping the image
6 and 5	Dictates the type of reset being used (where switch 6 is the MSB): 00 : single block phased reset 01 : dual block phased reset 10 : global reset 11 : quad block phased reset
7	Row Address Mode
8	WDT Enable (disables other resets)

NOTE : Single block reset for XGA DMD at 400 MHz is invalid

3 Functionality and Structure

The sample code is written hierarchically, with four levels, as illustrated in **Error! Reference source not found.**.

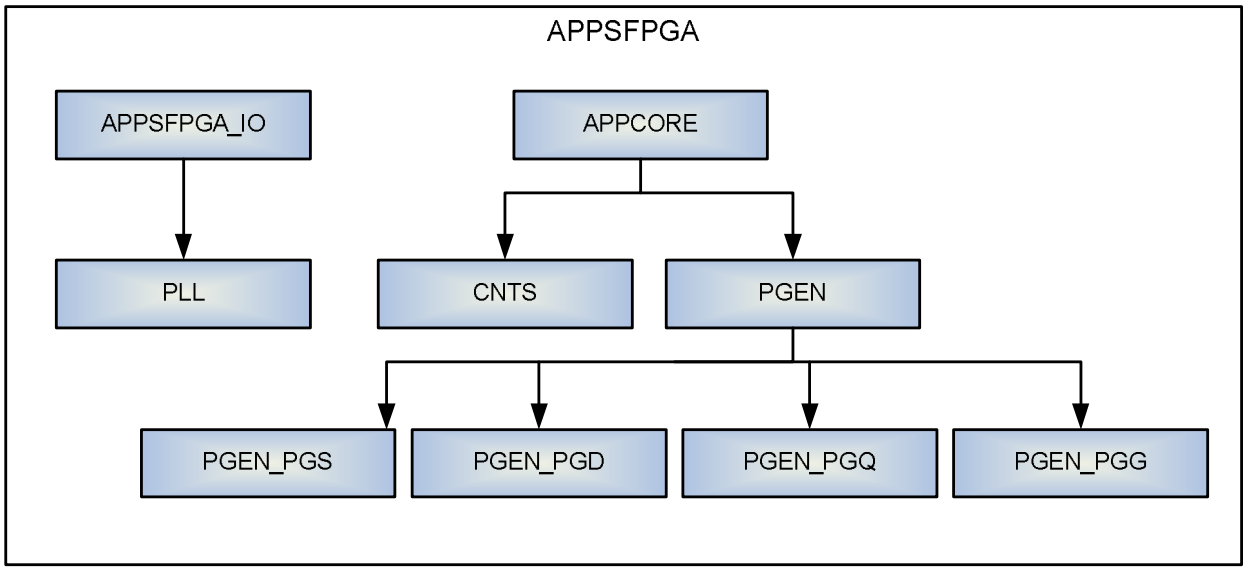

Figure 2. Sample APPSFPGA Hierarchy

The top level module (*APPSFPGA*) instantiates modules *APPSFPGA_IO* and *AppCore*. The *APPSFPGA_IO* module contains all input and output buffers which include DDR (SERDES) cells used to generate double data rate (DDR) output data to drive the DDC4100. In addition, a Phase Locked Loop (PLL) is used to create a 400 MHz and a 200 MHz global clock for the system. The *APPCORE* module instantiates two level-3 modules: *CNTS* and *PGEN*. These modules contain the logic for controlling the DDC4100.

Pattern generation is done in the *PGEN* module. There are many ways to generate a pattern. In this design, counters are used to keep track of the current row location.

Three counters, which form the core of the *CNTS* module, are used in this implementation. The active counter (active_cnt) is used to count 16 cycles for asserting data valid. The blank counter (blank_cnt) is used to count 16 cycles in which data valid is low. The pattern counter (pattern_cnt) is used to keep track of which row is currently being loaded. It is also used to generate the patterns to be displayed. A counter enable signal is associated with each of these counters. During normal operation, the active counter is used to form a window in which all other control signals can be passed to the DDC4100. Once this counter saturates, it enables the blank counter. After 16 cycles, the blank counter kicks off the active counter, and the pattern counter is incremented. Both the active counter and pattern counter values are passed into the pattern generation module, along with their respective counter enable signals.

The following sample code shows how to generate a simple horizontal line:

```
-- Each dout* is 4 DDR output cycles of data. Each 16 bits is reversed on the DMD - i.e. x0001 is the left bit of 16
-- 64 bits = 4 16 bit groups - x"AAAABBBBCCCCDDDD", a=c0 rising, b=c0 falling, c = c1 rising, d = c1 falling where c* is the DMD clock

        IF pgen_row_q1 = "00000000000" THEN
                douta <= x"FFFFFFFFFFFFFFFF" AFTER 1 PS; -- TOP BAR
                doutb <= x"FFFFFFFFFFFFFFFF" AFTER 1 PS;
                doutc <= x"FFFFFFFFFFFFFFFF" AFTER 1 PS;
                doutd <= x"FFFFFFFFFFFFFFFF" AFTER 1 PS;
```

```
ELSE

        douta <= x"0000000000000000" AFTER 1 PS;
        doutb <= x"0000000000000000" AFTER 1 PS;
        doutc <= x"0000000000000000" AFTER 1 PS;
        doutd <= x"0000000000000000" AFTER 1 PS;
END IF;
```

The bits are arranged to input to the DDR cells to convert the parallel signals to serial streams:

```
COMPONENT ddr_lvds_io IS
PORT (
    d1                                          : IN STD_LOGIC;     -- clock 0 rising edge
    d2                                          : IN STD_LOGIC;     -- clock 0 falling edge
    d3                                          : IN STD_LOGIC;     -- clock 1 rising edge
    d4                                          : IN STD_LOGIC;     -- clock 1 falling edge
    dout_dpp                                    : OUT STD_LOGIC;
    dout_dpn                                    : OUT STD_LOGIC;
    clk2X                                       : IN STD_LOGIC;     -- freq=2x
    clk1X                                       : IN STD_LOGIC;     -- freq=x
    reset                                       : IN STD_LOGIC
    );
END COMPONENT;

data_a_io : ddr_lvds_io
PORT MAP (
    d1  => appcore_dout_a_q(i+48),
    d2  => appcore_dout_a_q(i+32),
    d3  => appcore_dout_a_q(i+16),
    d4  => appcore_dout_a_q(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
    );

data_b_io : ddr_lvds_io
PORT MAP (
    d1  => appcore_dout_b_q(i+48),
    d2  => appcore_dout_b_q(i+32),
    d3  => appcore_dout_b_q(i+16),
    d4  => appcore_dout_b_q(i),
    dout_dpp => appsfpga_io_dout_bp(i),    -- diff_p output (connect directly to top-level port)
    dout_dpn => appsfpga_io_dout_bn(i),    -- diff_n output (connect directly to top-level port)
    clk2X => clk2x_b,
    clk1X => clk_b,
    reset => reset
    );

data_c_io : ddr_lvds_io
PORT MAP (
    d1  => appcore_dout_c_q(i+48),
    d2  => appcore_dout_c_q(i+32),
    d3  => appcore_dout_c_q(i+16),
    d4  => appcore_dout_c_q(i),
    dout_dpp => appsfpga_io_dout_cp(i),    -- diff_p output (connect directly to top-level port)
    dout_dpn => appsfpga_io_dout_cn(i),    -- diff_n output (connect directly to top-level port)
    clk2X => clk2x_b,
    clk1X => clk_b,
    reset => reset
    );

data_d_io : ddr_lvds_io
PORT MAP (
    d1  => appcore_dout_d_q(i+48),
    d2  => appcore_dout_d_q(i+32),
```

```
d3  => appcore_dout_d_q(i+16),
d4  => appcore_dout_d_q(i),
dout_dpp => appsfpga_io_dout_dp(i),    -- diff_p output (connect directly to top-level port)
dout_dpn => appsfpga_io_dout_dn(i),    -- diff_n output (connect directly to top-level port)
clk2X => clk2x_b,
clk1X => clk_b,
reset => reset
);
```

The *PGEN* module instantiates four level-4 modules. These modules create the control signals for each type of reset: single block phased reset (*PGEN_PGS*), dual block phased reset (*PGEN_PGD*), quad reset (*PGEN_PGQ*) and global block reset (*PGEN_PGG*). In each case, the counter values and counter enable signals are used to determine the values of data valid, row mode, block mode, and block address as per the DDC4100 data sheet. In pgen, these values are mux-ed together based upon the value of the reset type (Table 2) to decide the final values of the corresponding outputs.

3.1 Initialization

The INIT_ACTIVE signal (Table 1) 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 220ms 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 TIDN 2510443.

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

```
DOUT_A <= "00000000000000000000000000000001111111111111111000000000000000" WHEN (INIT_ACTIVE_GQ_INT = '1') ELSE
APPCORE_DOUT_A_Q;
DOUT_B <= "00000000000000000000000000000001111111111111111000000000000000" WHEN (INIT_ACTIVE_GQ_INT = '1') ELSE
APPCORE_DOUT_B_Q;
DOUT_C <= "00000000000000000000000000000001111111111111111000000000000000" WHEN (INIT_ACTIVE_GQ_INT = '1') ELSE
APPCORE_DOUT_C_Q;
DOUT_D <= "00000000000000000000000000000001111111111111111000000000000000" WHEN (INIT_ACTIVE_GQ_INT = '1') ELSE
APPCORE_DOUT_D_Q;
DVALID <= "0100" WHEN (INIT_ACTIVE = '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.

3.2 Power Down

To ensure long term reliability of the DMD, a shutdown procedure must be executed. Prior to power removal, assert the *PWR_FLOAT* (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 Additional Operational Requirements

DDC4100 has the ability to support both 1080p and XGA DMDs. When choosing the XGA output, only 768 rows are outputted. The pattern generator (PGEN) supports both the pattern generation and resets.

Due to the size difference between the XGA and the 1080p DMDs, the reset groups are different. This difference can be seen in the sample code. The reset modules (PGEN_PGG, PGEN_PGS, PGEN_PGD, and PGEN_PGQ) determine when a row is valid and when to send the reset. The PGEN automatically selects the appropriate reset groups according to the type of DMD attached. Table 3 illustrates the differences between the XGA and 1080p resets.

Table 3. Comparison of Sample Code for XGA and 1080p

Module	XGA	1080p

PGEN_PGG
(global reset)
	--determine if the current row is in the range of the DMD
	PROCESS (clk_g)
	BEGIN
	IF clk_g = '1' AND clk_g'event THEN
	CASE pgen_row IS
	WHEN "00000000000" => row_valid <= '1' AFTER 1 PS;
	WHEN **"01100000001"** => row_valid <= '0' AFTER 1 PS;
	WHEN OTHERS => row_valid <= row_valid AFTER 1 PS;
	END CASE;
	END IF;
	END PROCESS;
		--determine if the current row is in the range of the dmd
		PROCESS (clk_g)
		BEGIN
		IF clk_g = '1' AND clk_g'event THEN
		CASE pgen_row IS
		WHEN "00000000000" => row_valid <= '1' AFTER 1 PS;
		WHEN **"10000111001"** => row_valid <= '0' AFTER 1 PS;
		WHEN OTHERS => row_valid <= row_valid AFTER 1 PS;
		END CASE;
		END IF;
		END PROCESS;

PGEN_PGS
(single phased reset)
	--determine which set of blocks to reset
	--if the north/south flip switch is on, the blocks need to be reset in reverse order to avoid resetting blocks that have not yet been fully loaded
	
	PROCESS (clk_g)
	BEGIN
	IF clk_g = '1' AND clk_g'event THEN
	IF data_valid = '1' THEN
	CASE pgen_row IS
	WHEN **"00000110000"** =>
	blkmode <= "10" AFTER 1 PS;
	IF pgen_ns_flip = '0' THEN -- 48
	pgen_pgs_blkad_q <= "0000" AFTER 1 PS;
	ELSE
	pgen_pgs_blkad_q <= "1111" AFTER 1 PS;
	END IF;
	WHEN **"00001100000"** =>
	blkmode <= "10" AFTER 1 PS;
	IF pgen_ns_flip = '0' THEN -- 96
	pgen_pgs_blkad_q <= "0001" AFTER 1 PS;
	ELSE
	pgen_pgs_blkad_q <= "1110" AFTER 1 PS;
	END IF;
	WHEN **"00010010000"** =>
	blkmode <= "10" AFTER 1 PS;
	IF pgen_ns_flip = '0' THEN -- 144
	pgen_pgs_blkad_q <= "0010" AFTER 1 PS;
	ELSE
	pgen_pgs_blkad_q <= "1101" AFTER 1 PS;
	END IF;
	WHEN **"00011000000"** =>
	blkmode <= "10" AFTER 1 PS;
	IF pgen_ns_flip = '0' THEN -- 192
		--determine which set of blocks to reset
		--if the north/south flip switch is on, the blocks need to be reset in reverse order to avoid resetting blocks that have not yet been fully loaded
		
		PROCESS (clk_g)
		BEGIN
		IF clk_g = '1' AND clk_g'event THEN
		IF data_valid = '1' THEN
		CASE pgen_row IS
		WHEN **"00001001000"** =>
		blkmode <= "10" AFTER 1 PS;
		IF pgen_ns_flip = '0' THEN -- 72
		pgen_pgs_blkad_q <= "0000" AFTER 1 PS;
		ELSE
		pgen_pgs_blkad_q <= "1110" AFTER 1 PS;
		END IF;
		WHEN **"00010010000"** =>
		blkmode <= "10" AFTER 1 PS;
		IF pgen_ns_flip = '0' THEN -- 144
		pgen_pgs_blkad_q <= "0001" AFTER 1 PS;
		ELSE
		pgen_pgs_blkad_q <= "1101" AFTER 1 PS;
		END IF;
		WHEN **"00011011000"** =>
		blkmode <= "10" AFTER 1 PS;
		IF pgen_ns_flip = '0' THEN -- 216
		pgen_pgs_blkad_q <= "0010" AFTER 1 PS;
		ELSE
		pgen_pgs_blkad_q <= "1100" AFTER 1 PS;
		END IF;
		WHEN **"00100100000"** =>
		blkmode <= "10" AFTER 1 PS;
		IF pgen_ns_flip = '0' THEN -- 288

```
            pgen_pgs_blkad_q <= "0011" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1100" AFTER 1 PS;
            END IF;
        WHEN "00011110000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --240
            pgen_pgs_blkad_q <= "0100" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1011" AFTER 1 PS;
            END IF;
        WHEN "00100100000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --288
            pgen_pgs_blkad_q <= "0101" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1010" AFTER 1 PS;
            END IF;
        WHEN "00101010000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --336
            pgen_pgs_blkad_q <= "0110" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1001" AFTER 1 PS;
            END IF;
        WHEN "00110000000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --384
            pgen_pgs_blkad_q <= "0111" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1000" AFTER 1 PS;
            END IF;
        WHEN "00110110000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --432
            pgen_pgs_blkad_q <= "1000" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0111" AFTER 1 PS;
            END IF;
        WHEN "00111100000" =>
            blkmode <= "10" AFTER 1 PS;
            pgen_ns_flip = '0' THEN --480
            pgen_pgs_blkad_q <= "1001" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0110" AFTER 1 PS;
            END IF;
        WHEN "01000010000" =>
            blkmode <= "10" AFTER 1 PS;
            pgen_ns_flip = '0' THEN --528
            pgen_pgs_blkad_q <= "1010" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0101" AFTER 1 PS;
            END IF;
        WHEN "01001000000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --576
            pgen_pgs_blkad_q <= "1011" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0100" AFTER 1 PS;
            END IF;
        WHEN "01001110000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --624
            pgen_pgs_blkad_q <= "1100" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0011" AFTER 1 PS;
            END IF;
        WHEN "01010100000" =>
            blkmode <= "10" AFTER 1 PS;
```

```
            pgen_pgs_blkad_q <= "0011" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1011" AFTER 1 PS;
            END IF;
        WHEN "00101101000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --360
            pgen_pgs_blkad_q <= "0100" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1010" AFTER 1 PS;
            END IF;
        WHEN "00110110000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --432
            pgen_pgs_blkad_q <= "0101" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1001" AFTER 1 PS;
            END IF;
        WHEN "00111111000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --504
            pgen_pgs_blkad_q <= "0110" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "1000" AFTER 1 PS;
            END IF;
        WHEN "01001000000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --576
            pgen_pgs_blkad_q <= "0111" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0111" AFTER 1 PS;
            END IF;
        WHEN "01010001000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --648
            pgen_pgs_blkad_q <= "1000" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0110" AFTER 1 PS;
            END IF;
        WHEN "01011010000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --720
            pgen_pgs_blkad_q <= "1001" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0101" AFTER 1 PS;
            END IF;
        WHEN "01100011000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --792
            pgen_pgs_blkad_q <= "1010" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0100" AFTER 1 PS;
            END IF;
        WHEN "01101100000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --864
            pgen_pgs_blkad_q <= "1011" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0011" AFTER 1 PS;
            END IF;
        WHEN "01110101000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --936
            pgen_pgs_blkad_q <= "1100" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0010" AFTER 1 PS;
            END IF;
        WHEN "01111110000" =>
            blkmode <= "10" AFTER 1 PS;
```

```
            IF pgen_ns_flip = '0' THEN --672
            pgen_pgs_blkad_q <= "1101" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0010" AFTER 1 PS;
            END IF;
      WHEN "01011010000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --720
             pgen_pgs_blkad_q <= "1110" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0001" AFTER 1 PS;
            END IF;
      WHEN "00000000000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --0
            pgen_pgs_blkad_q <= "1111" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0000" AFTER 1 PS;
            END IF;
      WHEN OTHERS =>
            blkmode <= "00" AFTER 1 PS;
            pgen_pgs_blkad_q <= "0000" AFTER 1 PS;
   END CASE;
```

```
            IF pgen_ns_flip = '0' THEN --1008
            pgen_pgs_blkad_q <= "1101" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0001" AFTER 1 PS;
            END IF;
      WHEN "00000000000" =>
            blkmode <= "10" AFTER 1 PS;
            IF pgen_ns_flip = '0' THEN --0
            pgen_pgs_blkad_q <= "1110" AFTER 1 PS;
            ELSE
            pgen_pgs_blkad_q <= "0000" AFTER 1 PS;
            END IF;
      WHEN OTHERS =>
            blkmode <= "00" AFTER 1 PS;
            pgen_pgs_blkad_q <= "0000" AFTER 1 PS;
   END CASE;
```

The 1080p DMD will not display the left most and right most 64 bits. Therefore, a 1080p image will appear “shifted” when displayed on the XGA DMD.

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
DLP Discovery Digital Controller 4100 Datasheet	2510443
DAD2000 Datasheet	2506593
DMD Glass Cleaning Procedure	2504640
DMD Handling Specification (Type-A)	2504641