Juan Barroso
Embedded Systems Design
Final Report
5-8-07

Digital Camera

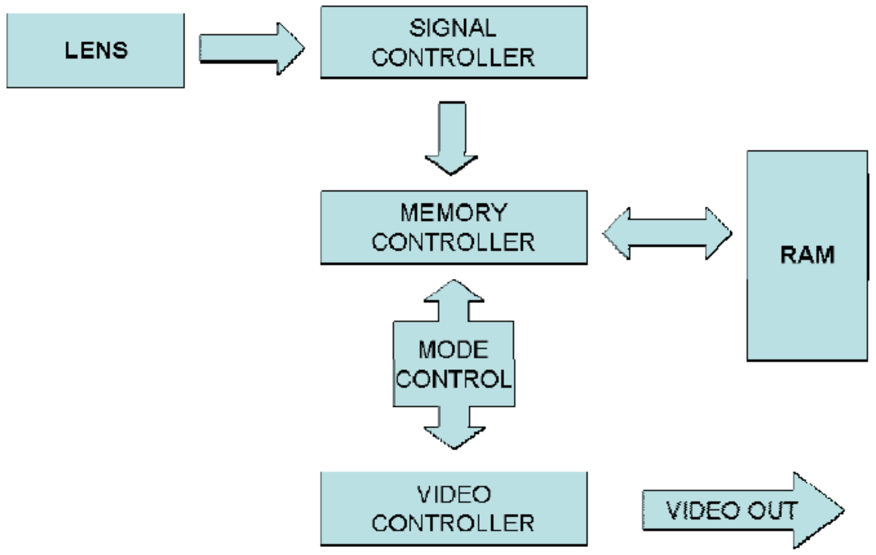
Intro:

For this design project, the idea was to create a digital camera using the FPGA and a CMOS Sensor. The system itself would require massive time, work, and dedication to implement such an elaborate system. Although the architecture behind this design is fairly simple in nature, the implementation is far from it.

When you imagine a digital camera, there are a certain things that come to mind. First, the ability to clearly see the image that you wish to capture a snapshot of plays an important aspect of the design. Clarity in both the image and the display of the photo is what makes for a well rounded camera. Another concept to keep in mind is storage. When looking at a digital camera, there are many different types of external media that can be used to store the image to, such as memory sticks, flash drives, and other forms of memory. The final thing to keep in mind is usability. A digital camera has to be not only easy to use, but also easy to understand its functionality. The user should easily understand how to not only take a photograph, but also how to view the images they have already taken.

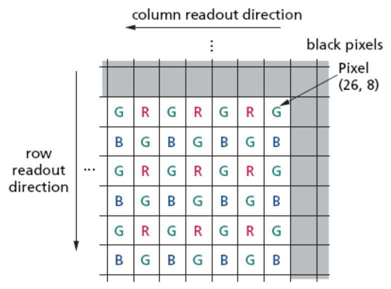
While others might view the usage of a camera differently, these concepts of the camera are what played a major role in the design of the camera's architecture. The intended system of the camera was built around these ideas as to how the camera should act, and it was meant to be implemented to the best of abilities on the FPGA.

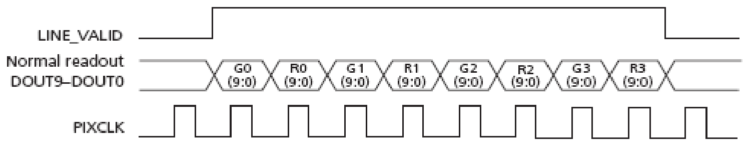
Design and Architecture:



The camera, although its architecture seems straight forward and relatively simple, is far more complex than one might imagine. Above is the original diagram of the camera's architecture. As stated before, this is a complex system. The main reasoning behind this is because the camera itself requires extensive signal processing to produce the image that the user wishes to see.

Before diving into the specifics, lets' take a look at the architecture diagram and go into what it does. The LENS block is the sensor and outputs the data that it is receiving for a given pulse of its clock. One thing to keep in mind is that the FPGA runs with a clock of 50MHz, while the LENS is working with a clock of 25MHz (half the speed). As the data comes out, it is read into the Signal Processor. Within this block alone, a large amount of work is done to the images to begin the process of producing an image to the screen. Within this block, the data is read in and then stripped to the into the color code. This consists of stripping from the data the Red, Green, and Blue values of the image that the lens is capturing.





As you can see, as the image comes in and the colors are stripped out, they are coming into the system in a certain order. This order must be maintained so that the image is read properly when being displayed. Also, the pixel order by which the image is being read is also running is the opposite direction that may seem intuitive (the rows are being read in right to left). If the image were displayed to the screen in this manner, it would appear as if you were looking at a mirror image of what you planned to grab a photograph of. After the pixels are read in, and the system begins filtering the RGB values, they are passed to another subsystem whose purpose is to reverse the order of the pixels per row. This would be the equivalent of reading in the pixels left to right.

Now that we have the data, the next order of business is one of the concepts behind the camera, storage. We have data, we want to take a picture, but where are we going to put it? With the architecture of this camera, the image will be stored on the FPGA itself, or more so, on the FPGA's SDRAM. The FPGA has a 512KB SDRAM that we will be using for the storage of the images. From the RAM, it is read into the VGA Controller that will then take the image of the sensor and display it to the screen. Now, although the image passes through the SDRAM initially, it is not stored unless an event is triggered by a button press. The VGA Controller handles the display, taking care of vertical and horizontal blanking to display the image at a resolution of 640 x 480. After the image is stored on the SDRAM, the user should be able to display the images that were taken on the screen. The subsystems Memory Mangement and Mode Control are the what control this feature. The way it was to be implemented was to allow for a switch event to determine the state of the over all system. If the switch was high, then the images would be read from SDRAM and displayed to the monitor; if the switch was low, the sensor would continue to read images and display them to screen in real time until the image was stored, in which case the image that was captured stayed on the screen until the user instructed the sensor to reset and begin reading data again.

This system was created and coded in VHDL, with a master file called Dig_Cam.vhd. Although the main file was in VHDL, as well as other protocols, many of the preexisting files, which were written in Verilog, were used in the creation of this system. The mix of these two types worked fine, it just caused more work to be done to understand the code itself. Next, we discuss the software that worked in the background over the system.

Software:

In the system that was written, the majority of the work was done in the hardware. This included the connections, signal processing, display, and storage of the images that were taken and to be taken. Software played a major and important role, how the image was stored and displayed afterwards. Typically, given the minimum resolution of a monitor to be 640 x 480 pixels, this means that the total number of pixels to be stored would be 307,200 pixels. With one pixel per byte, the image would require about

307KB, leaving only 205 KB of space left, which is insufficient space for the possibility of multiple images on the FPGA itself. So the purpose of the C-coding was to reduce the size of the image before it was stored and then increase the size afterwards.

As stated before, the minimum resolution of the monitor is 640 x 480 pixels. To allow for multiple images, we plan to reduce the size of the image by a factor of 4 (half the width and half the length). This would mean that the resolution would then be 320 x 240, which would only be 76,800 pixels or approximately 77 KB. This would allow the FPGA to then store up to 5 images. Once those images are stored, it would be nice to still see the large image you intended to see. To do this, given that the state of the system was to display the images on SDRAM, the software would then double each pixel as it would be read to then display the image to the screen. Although there might be some ambiguity from resizing, it was believed to be negligible.

Lessons:

In the course of this project, there were a few things that I learned first and most important, start early. The project requires massive time and much work must go into it. I saw very quickly that the more I thought I knew, the less I actually did. Also, communication with team members plays a massive role. The better communication the group, the more likely a chance the group will have of getting work done productively. I would also like to add that you should always ask for assistance if you truly do not understand something. Guidelines from the professor, TAs, and mainly other students assisted remarkably to the amount of work that got done on this project.

Source Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Dig_Cam is

  port (
    -- Clocks

    CLOCK_25,                                      -- 25 MHz
    CLOCK_50,                                      -- 50 MHz
    EXT_CLOCK : in std_logic;                      -- External Clock

    -- Buttons and switches

    KEY : in std_logic_vector(3 downto 0);         -- Push buttons
    SW : in std_logic_vector(17 downto 0);         -- DPDT switches

    -- LED displays

    HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
       : out std_logic_vector(6 downto 0);
    LEDG : out std_logic_vector(8 downto 0);       -- Green LEDs
    LEDR : out std_logic_vector(17 downto 0);      -- Red LEDs

    -- RS-232 interface

    UART_TXD : out std_logic;                      -- UART transmitter
    UART_RXD : in std_logic;                       -- UART receiver

     -- SDRAM

    DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
    DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
    DRAM_LDQM,                                     -- Low-byte Data Mask
    DRAM_UDQM,                                     -- High-byte Data Mask
    DRAM_WE_N,                                     -- Write Enable
    DRAM_CAS_N,                                    -- Column Address Strobe
    DRAM_RAS_N,                                    -- Row Address Strobe
    DRAM_CS_N,                                     -- Chip Select
    DRAM_BA_0,                                     -- Bank Address 0
    DRAM_BA_1,                                     -- Bank Address 0
    DRAM_CLK,                                      -- Clock
    DRAM_CKE : out std_logic;                      -- Clock Enable

    -- FLASH

    FL_DQ : inout std_logic_vector(7 downto 0);      -- Data bus
    FL_ADDR : out std_logic_vector(21 downto 0);  -- Address bus
    FL_WE_N,                                         -- Write Enable
    FL_RST_N,                                        -- Reset
    FL_OE_N,                                         -- Output Enable
    FL_CE_N : out std_logic;                         -- Chip Enable

    -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
    SRAM_UB_N,                                     -- High-byte Data Mask
    SRAM_LB_N,                                     -- Low-byte Data Mask
```

```
SRAM_WE_N,                                    -- Write Enable
SRAM_CE_N,                                    -- Chip Enable
SRAM_OE_N : out std_logic;                    -- Output Enable

-- USB controller

OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
OTG_ADDR : out std_logic_vector(1 downto 0);    -- Address
OTG_CS_N,                                       -- Chip Select
OTG_RD_N,                                       -- Write
OTG_WR_N,                                       -- Read
OTG_RST_N,                                      -- Reset
OTG_FSPEED,                    -- USB Full Speed, 0 = Enable, Z = Disable
OTG_LSPEED : out std_logic;    -- USB Low Speed, 0 = Enable, Z = Disable
OTG_INT0,                                       -- Interrupt 0
OTG_INT1,                                       -- Interrupt 1
OTG_DREQ0,                                      -- DMA Request 0
OTG_DREQ1 : in std_logic;                       -- DMA Request 1
OTG_DACK0_N,                                    -- DMA Acknowledge 0
OTG_DACK1_N : out std_logic;                    -- DMA Acknowledge 1

-- 16 X 2 LCD Module

LCD_ON,                     -- Power ON/OFF
LCD_BLON,                   -- Back Light ON/OFF
LCD_RW,                     -- Read/Write Select, 0 = Write, 1 = Read
LCD_EN,                     -- Enable
LCD_RS : out std_logic;     -- Command/Data Select, 0 = Command, 1 = Data
LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

-- SD card interface

SD_DAT,                     -- SD Card Data
SD_DAT3,                    -- SD Card Data 3
SD_CMD : inout std_logic;   -- SD Card Command Signal
SD_CLK : out std_logic;     -- SD Card Clock

-- USB JTAG link

TDI,                        -- CPLD -> FPGA (data in)
TCK,                        -- CPLD -> FPGA (clk)
TCS : in std_logic;         -- CPLD -> FPGA (CS)
TDO : out std_logic;        -- FPGA -> CPLD (data out)

-- I2C bus

I2C_SDAT : inout std_logic; -- I2C Data
I2C_SCLK : out std_logic;   -- I2C Clock

-- PS/2 port

PS2_DAT,                    -- Data
PS2_CLK : in std_logic;     -- Clock

-- VGA output

VGA_CLK,                                             -- Clock
VGA_HS,                                              -- H_SYNC
VGA_VS,                                              -- V_SYNC
VGA_BLANK,                                           -- BLANK
VGA_SYNC : out std_logic;                            -- SYNC
VGA_R,                                               -- Red[9:0]
VGA_G,                                               -- Green[9:0]
```

```
    VGA_B : out std_logic_vector(9 downto 0);           -- Blue[9:0]

    --  Ethernet Interface

    ENET_DATA : inout std_logic_vector(15 downto 0);    -- DATA bus 16Bits
    ENET_CMD,           -- Command/Data Select, 0 = Command, 1 = Data
    ENET_CS_N,                                          -- Chip Select
    ENET_WR_N,                                          -- Write
    ENET_RD_N,                                          -- Read
    ENET_RST_N,                                         -- Reset
    ENET_CLK : out std_logic;                           -- Clock 25 MHz
    ENET_INT : in std_logic;                            -- Interrupt

    -- Audio CODEC

    AUD_ADCLRCK : inout std_logic;                      -- ADC LR Clock
    AUD_ADCDAT : in std_logic;                          -- ADC Data
    AUD_DACLRCK : inout std_logic;                      -- DAC LR Clock
    AUD_DACDAT : out std_logic;                         -- DAC Data
    AUD_BCLK : inout std_logic;                         -- Bit-Stream Clock
    AUD_XCK : out std_logic;                            -- Chip Clock

    -- Video Decoder

    TD_DATA : in std_logic_vector(7 downto 0);  -- Data bus 8 bits
    TD_HS,                                      -- H_SYNC
    TD_VS : in std_logic;                       -- V_SYNC
    TD_RESET : out std_logic;                   -- Reset

    -- General-purpose I/O

    GPIO_0,                                      -- GPIO Connection 0
    GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1
    );

end Dig_Cam;

architecture datapath of Dig_Cam is

component CCD_Capture is
      port(  oDATA : out std_logic_vector(9 downto 0);
                    oDVAL : out std_logic;
                    oX_Cont,
                    oY_Cont : out std_logic_vector(10 downto 0);
                    oFrame_Cont : out std_logic_vector(31 downto 0);
                    iDATA : in std_logic_vector(9 downto 0);
                    iFVAL,
                    iLVAL,
                    iSTART,
                    iEND,
                    iCLK,
                    iRST : in std_logic );
end component;

  component I2C_CCD_Config is
      port( iCLK : in std_logic;
                 iRST_N : in std_logic;
                 iExposure : in std_logic_vector(15 downto 0);
                 I2C_SCLK : out std_logic;
                 I2C_SDAT : inout std_logic
               );

  end component;
```

```
component RAW2RGB is
       port( iX_Cont,
             iY_Cont : in std_logic_vector(10 downto 0);
                iDATA : in std_logic_vector(9 downto 0);
                iDVAL : in std_logic;
                oRed,
                oGreen,
                oBlue : out std_logic_vector(9 downto 0);
                oDVAL : out std_logic;

                iCLK,
                iRST : in std_logic
              );
end component;

component Reset_Delay is
       port( iCLK,
                iRST :in std_logic;
                signal oRST_0,
                          oRST_1,
                             oRST_2 : out std_logic
              );
end component;

component Mirror_Col is
       port( iCCD_R,
                iCCD_G,
                iCCD_B : in std_logic_vector(9 downto 0);
                iCCD_DVAL,
                iCCD_PIXCLK,
                iRST_N : in std_logic;

                oCCD_R,
                oCCD_G,
                oCCD_B : out std_logic_vector(9 downto 0);
                oCCD_DVAL: out std_logic
              );
end component;

  component VGA_Controller is
   port(iRed,
              iGreen,
              iBlue : in std_logic_vector( 9 downto 0);
              oRequest : out std_logic;

       oVGA_CLOCK,
       oVGA_H_SYNC,
       oVGA_V_SYNC,
       oVGA_BLANK,
       oVGA_SYNC : out std_logic;
              oVGA_R,
       oVGA_G,
              oVGA_B : out std_logic_vector(9 downto 0);

              iRST_N : in std_logic;
       iCLK   : in std_logic
  );
  end component;

  component sram_controller is
       port(  REF_CLK, RESET_N : in std_logic;
                    WR1_Data : in std_logic_vector(15 downto 0);
                    WR1 : in std_logic;
                    WR1_ADDR, WR1_MAX_ADDR : in std_logic_vector(21 downto 0);
                    WR1_Length : in std_logic_vector(8 downto 0);
                    WR1_Load, WR1_CLK : in std_logic;

                    WR2_Data : in std_logic_vector(15 downto 0);
                    WR2 : in std_logic;
                    WR2_ADDR, WR2_MAX_ADDR : in std_logic_vector(21 downto 0);
                    WR2_Length : in std_logic_vector(8 downto 0);
                    WR2_Load, WR2_CLK : in std_logic;

                    RD1_Data : out std_logic_vector(15 downto 0);
                    RD1 : in std_logic;
                    RD1_ADDR, RD1_MAX_ADDR : in std_logic_vector(21 downto 0);
                    RD1_Length : in std_logic_vector(8 downto 0);
                    RD1_Load, RD1_CLK : in std_logic;
                    RD1_Empty : out std_logic;
                    RD1_Use : out std_logic_vector(8 downto 0);

                    RD2_Data : out std_logic_vector(15 downto 0);
                    RD2 : in std_logic;
                    RD2_ADDR, RD2_MAX_ADDR : in std_logic_vector(21 downto 0);
                    RD2_Length : in std_logic_vector(8 downto 0);
                    RD2_Load, RD2_CLK : in std_logic;

                    SA : out std_logic_vector(11 downto 0);
                    BA, CS_N : out std_logic_vector(1 downto 0);
                    CKE, RAS_N, CAS_N, WE : out std_logic;
                    DQ : inout std_logic_vector(15 downto 0);
                    DQM : out std_logic_vector(1 downto 0);
                    SDR_CLK : out std_logic
             );
       end component;


signal clk25 : std_logic := '1';
signal read : std_logic;
signal Vals : std_logic_vector(9 downto 0);
signal CCD_SDATA : std_logic;
signal CCD_SCLK,
       CCD_Mclk : std_logic;
signal CCD_Data,
       mCCD_DATA : std_logic_vector(9 downto 0);
signal CCD_SDAT,
          CCD_FVAL,
          CCD_LVal,
          CCD_PIXCLK,
          mCCD_Dval_d,
          mCCD_Dval : std_logic;

signal Read_Data1,
          Read_Data2 : std_logic_vector(15 downto 0);

signal VGA_CTRL_CLK,
          AUD_CTRL_CLK : std_logic;

signal X_ADDR,
          green_data,
          mCCD_R,
          mCCD_G,
          mCCD_B,
          sCCD_R,
          sCCD_G,
          sCCD_B,
          rCCD_Data : std_logic_vector(9 downto 0);

signal DRAM_BA_3, DRAM_DQM : std_logic_vector(1 downto 0);

signal DLY_RST_0,
          DLY_RST_1,
          DLY_RST_2,
          rCCD_LVAL,
          rCCD_FVAL,
          sCCD_DVAL : std_logic;

signal X_Cont,
       Y_Cont : std_logic_vector(10 downto 0);

signal Frame_Cont : std_logic_vector(31 downto 0);
signal c : std_logic_vector( 1 downto 0);
begin

process(CLOCK_50)
begin

CCD_Data(0) <= GPIO_1(0);
CCD_Data(1) <= GPIO_1(1);
CCD_Data(2) <= GPIO_1(5);
CCD_Data(3) <= GPIO_1(3);
CCD_Data(4) <= GPIO_1(2);
CCD_Data(5) <= GPIO_1(4);
CCD_Data(6) <= GPIO_1(6);
CCD_Data(7) <= GPIO_1(7);
CCD_Data(8) <= GPIO_1(8);
CCD_Data(9) <= GPIO_1(9);

GPIO_1(11) <= CCD_Mclk;
GPIO_1(15) <= CCD_SDAT;
GPIO_1(14) <= CCD_SCLK;

CCD_FVal <= GPIO_1(13);
CCD_LVal <= GPIO_1(12);
CCD_PIXCLK <= GPIO_1(10);

green_data(9 downto 5) <= Read_Data1(14 downto 10);
green_data(4 downto 0) <= Read_Data2(14 downto 10);
DRAM_BA_1 <= DRAM_BA_3(1);
DRAM_BA_0 <= DRAM_BA_3(0);
DRAM_UDQM <= DRAM_DQM(1);
DRAM_LDQM <= DRAM_DQM(0);
DRAM_CS_N <= c(0);
DRAM_CS_N <= c(1);
end process;

process(CCD_PIXCLK)
begin
       rCCD_DATA <= CCD_DATA;
       rCCD_LVAL <= CCD_LVAL;
       rCCD_FVAL <= CCD_FVAL;
end process;


CCD : CCD_Capture port map(
       oDATA => mCCD_DATA,
       oDVAL => mCCD_DVAL,
       oX_Cont => X_Cont,
       oY_Cont => Y_Cont,
       oFrame_Cont => Frame_Cont,
       iDATA => CCD_Data,
       iFVAL => CCD_FVal,
       iLVAL => CCD_LVal,
       iSTART => not key(3),
       iEND => not key(2),
       iCLK => CCD_PIXCLK,
       iRST => DLY_RST_1
);

IC_Config : I2C_CCD_Config port map(
iCLK => CLOCK_50,
iRST_N => key(1),
iExposure => sw(15 downto 0),
I2C_SCLK => CCD_SCLK,
I2C_SDAT => CCD_SDATA
);

VGA: VGA_Controller port map (
  iRed => Read_Data2(9 downto 0),
  iGreen => green_data,
  iBlue => Read_Data1(9 downto 0),
  oRequest => read,

  oVGA_CLOCK => VGA_CLK,
  oVGA_H_SYNC => VGA_HS,
  oVGA_V_SYNC => VGA_VS,
  oVGA_BLANK => VGA_BLANK,
  oVGA_SYNC => VGA_SYNC,
  oVGA_G => VGA_G,
  oVGA_R => VGA_R,
  oVGA_B => VGA_B,

  iCLK => VGA_CTRL_CLK,
  iRST_N => DLY_RST_2
);

Res : Reset_Delay port map(
  iCLK => CLOCK_50,
  iRST => key(0),
  oRST_0 => DLY_RST_0,
  oRST_1 => DLY_RST_1,
  oRST_2 => DLY_RST_2
);

Colors : RAW2RGB port map(
  oRed => mCCD_R,
  oGreen => mCCD_G,
  oBlue => mCCD_B,
  oDVAL => mCCD_DVAL_d,
  iX_Cont => X_Cont,
  iY_Cont => Y_Cont,
  iDATA => mCCD_Data,
  iDVAL => mCCD_DVAL,
  iCLK => CCD_PIXCLK,
  iRST => DLY_RST_1
);

Pic : Mirror_Col port map(
  iCCD_R => mCCD_R,
  iCCD_G => mCCD_G,
  iCCD_B => mCCD_B,
  iCCD_DVAL => mCCD_DVAL_d,
  iCCD_PIXCLK => CCD_PIXCLK,
  iRST_N => DLY_RST_1,
  oCCD_R => sCCD_R,
  oCCD_G => sCCD_G,
  oCCD_B => sCCD_B,
  oCCD_DVAL => sCCD_DVAL
);

SRAM : sram_controller port map(
       REF_CLK => CLOCK_50,
       RESET_N => '1',

       WR1_Data => (sCCD_G(9 downto 5) & sCCD_B(9 downto 0) & '0'),
       WR1 => sCCD_DVAL,
       WR1_ADDR => "0000000000000000000000",
       WR1_MAX_ADDR => "0001010000000000000000",
       WR1_Length => "100000000",
       WR1_Load => not DLY_RST_0,
       WR1_CLK => CCD_PIXCLK,

       WR2_Data => (sCCD_G(4 downto 0) & sCCD_R(9 downto 0) & '0'),
       WR2 => sCCD_DVAL,
       WR2_ADDR => "0100000000000000000000",
       WR2_MAX_ADDR => "0101010000000000000000",
       WR2_Length => "100000000",
       WR2_Load => not DLY_RST_0,
       WR2_CLK => CCD_PIXCLK,

       RD1_Data => Read_Data1,
       RD1 => Read,
       RD1_ADDR => "0000000010100000000000",
       RD1_MAX_ADDR => "0001001101100000000000",
       RD1_Length => "100000000",
       RD1_Load => not DLY_RST_0,
       RD1_CLK => VGA_CTRL_CLK,

       RD2_Data => Read_Data2,
       RD2 => Read,
       RD2_ADDR => "0100000010100000000000",
       RD2_MAX_ADDR => "0101001101100000000000",
       RD2_Length => "100000000",
       RD2_Load => not DLY_RST_0,
       RD2_CLK => VGA_CTRL_CLK,

       SA => DRAM_ADDR,
       BA => DRAM_BA_3,
       CS_N => c,
       CKE => DRAM_CKE,
       RAS_N => DRAM_RAS_N,
       CAS_N => DRAM_CAS_N,
       WE => DRAM_WE_N,
       DQ => DRAM_DQ,
       DQM => DRAM_DQM,
       SDR_CLK => DRAM_CLK
);

--  HEX7     <= "0001001"; -- Leftmost
--  HEX6     <= "0000110";
--  HEX5     <= "1000111";
--  HEX4     <= "1000111";
--  HEX3     <= "1000000";
--  HEX2     <= (others => '1');
--  HEX1     <= (others => '1');
--  HEX0     <= (others => '1');          -- Rightmost
--  LCD_ON   <= '1';
--  LCD_BLON <= '1';

  -- Set all bidirectional ports to tri-state
--  DRAM_DQ     <= (others => 'Z');
--  FL_DQ       <= (others => 'Z');
--  SRAM_DQ     <= (others => 'Z');
-- OTG_DATA    <= (others => 'Z');
--  LCD_DATA    <= (others => 'Z');
--  SD_DAT      <= 'Z';
--  I2C_SDAT    <= 'Z';
--  ENET_DATA   <= (others => 'Z');
--  AUD_ADCLRCK <= 'Z';
--  AUD_DACLRCK <= 'Z';
--  AUD_BCLK    <= 'Z';

  process (CLOCK_50)
  begin
    if CLOCK_50'event and CLOCK_50 = '1' then
      clk25 <= not clk25;
    end if;
  end process;

end datapath;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sram_controller is
       port(  REF_CLK, RESET_N : in std_logic;

                    WR1_Data : in std_logic_vector(14 downto 0);
                    WR1 : in std_logic;
                    WR1_ADDR, WR1_MAX_ADDR : in std_logic_vector(21 downto 0);
                    WR1_Length : in std_logic_vector(8 downto 0);
                    WR1_Load, WR1_CLK : in std_logic;
                    WR1_Full : out std_logic;
                    WR1_Use : out std_logic_vector(8 downto 0);

                    WR2_Data : in std_logic_vector(14 downto 0);
                    WR2 : in std_logic;
                    WR2_ADDR, WR2_MAX_ADDR : in std_logic_vector(21 downto 0);
                    WR2_Length : in std_logic_vector(8 downto 0);
                    WR2_Load, WR2_CLK : in std_logic;
                    WR2_Full : out std_logic;
                    WR2_Use : out std_logic_vector(8 downto 0);

                    RD1_Data : out std_logic_vector(14 downto 0);
                    RD1 : in std_logic;
                    RD1_ADDR, RD1_MAX_ADDR : in std_logic_vector(21 downto 0);
                    RD1_Length : in std_logic_vector(8 downto 0);
                    RD1_Load, RD1_CLK : in std_logic;
                    RD1_Empty : out std_logic;
                    RD1_Use : out std_logic_vector(8 downto 0);

                    RD2_Data : out std_logic_vector(14 downto 0);
                    RD2 : in std_logic;
                    RD2_ADDR, RD2_MAX_ADDR : in std_logic_vector(21 downto 0);
                    RD2_Length : in std_logic_vector(8 downto 0);
                    RD2_Load, RD2_CLK : in std_logic;
                    RD2_Empty : out std_logic;
                    RD2_Use : out std_logic_vector(8 downto 0);

                    SA : out std_logic_vector(11 downto 0);
                    BA : out std_logic_vector(1 downto 0);
                    CS_N : out std_logic;
                    CKE, RAS_N, CAS_N, WE : out std_logic;
                    DQ : inout std_logic_vector(15 downto 0);
                    DQM : out std_logic_vector(1 downto 0);
                    SDR_CLK : out std_logic
             );
end sram_controller;

architecture datapath of sram_controller is

       component Sdram_PLL is
             port(  inclk : in std_logic;
                          c0, c1 : out std_logic
             );
       end component;

       component command is
             port(  CLK, RESET_N, NOP,
                          READA, WRITEA, REFRESH,
                          PRECHARGE, LOAD_MODE, REF_REQ,
                          INIT_REQ, PM_STOP, PM_DONE : in std_logic;

                          SADDR : in std_logic_vector(22 downto 0);

                          REF_ACK, CM_ACK, OE, CKE,
                          RAS_N, CAS_N, WE_N : out std_logic;

                          BA, CS_N : out std_logic_vector(1 downto 0);
                          SA : out std_logic_vector(11 downto 0)
             );
       end component;

       component control_interface is
             port(  CLK, RESET_N, REF_ACK,
                          CM_ACK : in std_logic;

                          CMD : in std_logic_vector (2 downto 0);
                          ADDR : in std_logic_vector(22 downto 0);

                          NOP, READA, WRITEA, REFRESH, PRECHARGE,
                          LOAD_MODE, REF_REQ, INIT_REQ,
                          CMD_ACK : out std_logic;

                          SADDR : out std_logic_vector(22 downto 0)
             );
       end component;

       component sdr_data_path is
             port(  CLK, RESET_N : in std_logic;
                          DATAIN : in std_logic_vector(15 downto 0);
                          DM : in std_logic_vector(1 downto 0);

                          DQOUT : out std_logic_vector(15 downto 0);
                          DQM : out std_logic_vector(1 downto 0)
             );
       end component;
```

```
        component Sdram_FIFO is
                port(  aclr, rdclk, rdreq,
                                wrclk, wrreq : in std_logic;
                                data : in std_logic_vector(15 downto 0);

                                rdempty, wrfull : out std_logic;
                                rdusedw, wrusedw : out std_logic_vector(8 downto 0);
                                q : out std_logic_vector(15 downto 0)
                );

        end component;

        signal          mADDR, rWR1_ADDR, rWR1_MAX_ADDR,
                        rWR2_ADDR, rWR2_MAX_ADDR, saddr,
                        rRD1_ADDR, rRD1_MAX_ADDR, rRD2_ADDR,
                        rRD2_MAX_ADDR : std_logic_vector(22 downto 0);

        signal  mDATA_OUT, mDATA_IN, mDATA_IN1,
                        mDATA_IN2, DQOUT : std_logic_vector(15 downto 0);

        signal          mLength, rWR1_Length, rWR2_Length,
                        rRD1_Length, rRD2_Length,
                        write_side_fifo_rusedw1,
                        read_side_fifo_wusedw1,
                        write_side_fifo_rusedw2,
                        read_side_fifo_wusedw2: std_logic_vector(8 downto 0);

        signal  WR_Mask, RD_Mask, CMD, br,
                        cs_l, IBA, ICS_N, IDQM : std_logic_vector(1 downto 0);

        signal  mWR_Done, mRD_Done, mWR, Pre_WR,
                        mRD, Pre_RD, PM_Stop, PM_Done,
                        Read, Write, CMDACK, ClE, RlS_N,
                        ClS_N, Wl_N, ICKE, IRAS_N, ICAS_N,
                        IWE_N, OUT_Valid, IN_REQ, load_mode,
                        sop, readi, writei, refresh, precharge,
                        oe, ref_ack, ref_req, init_req, cm_ack,
                        active, CLK : std_logic;

        signal  SADD, ISA : std_logic_vector(11 downto 0);

        signal  ST : std_logic_vector(9 downto 0);

begin
        sdram_pll1 : Sdram_PLL port map(
                inclk => REF_CLK,
                c0 => CLK,
                c1 => SDR_CLK
        );

        control1 : control_interface port map(
                CLK => CLK,
                RESET_N => RESET_N,
                CMD => CMD,
                ADDR => mADDR,
                REF_ACK => ref_ack,
                CM_ACK => cm_ack,
                NOP => sop,
                READA => readi,
                WRITEA => writei,
                REFRESH => refresh,
                PRECHARGE => precharge,
        LOAD_MODE => load_mode,
        SADDR => saddr,
        REF_REQ => ref_req,
        INIT_REQ => init_req,
        CMD_ACK => CMDACK
);

command1 : command port map(
        CLK => CLK,
        RESET_N => RESET_N,
        SADDR => saddr,
        NOP => sop,
        READA => readi,
        WRITEA => writei,
        REFRESH => refresh,
        PRECHARGE => precharge,
        LOAD_MODE => load_mode,
        REF_REQ => ref_req,
        INIT_REQ => init_req,
        REF_ACK => ref_ack,
        CM_ACK => cm_ack,
        OE => oe,
        PM_STOP => PM_Stop,
        PM_DONE => PM_Done,
        SA => ISA,
        BA => IBA,
        CS_N => ICS_N,
        CKE => ICKE,
        RAS_N => IRAS_N,
        CAS_N => ICAS_N,
        WE_N => IWE_N
);

data_path1 : sdr_data_path port map(
        CLK => CLK,
        RESET_N => RESET_N,
        DATAIN => mDATA_IN,
        DM => "00",
        DQOUT => DQOUT,
        DQM => IDQM
);

write_fifo1 : Sdram_FIFO port map(
        data => WR1_Data,
        wrreq => WR1,
        wrclk => WR1_CLK,
        aclr => WR1_Load,
        rdreq => (IN_REQ and WR_Mask(0)),
        rdclk => CLK,
        q => mDATA_IN1,
        wrfull => WR1_Full,
        wrusedw => WR1_Use,
        rdusedw => write_side_fifo_rusedw1
);

write_fifo2 : Sdram_FIFO port map(
        data => WR2_Data,
        wrreq => WR2,
        wrclk => WR2_CLK,
        aclr => WR2_Load,
        rdreq => (IN_REQ and WR_Mask(1)),
        rdclk => CLK,
        q => mDATA_IN1,
        wrfull => WR2_Full,
        wrusedw => WR2_Use,
        rdusedw => write_side_fifo_rusedw2
);

process(mData_IN, mDATA_IN1, mData_IN2)
begin
        if(WR_Mask(0)= '1') then
                mData_IN <= mData_IN1;
        else
                mData_IN <= mData_IN2;
        end if;
end process;

read_fifo1 : Sdram_FIFO port map(
        data => mData_OUT,
        wrreq => (OUT_Valid and RD_Mask(0)),
        wrclk => CLK,
        aclr => RD1_Load,
        rdreq => RD1,
        rdclk => RD1_CLK,
        q => RD1_Data,
        wrusedw => read_side_fifo_wusedw1,
        rdempty => RD1_Empty,
        rdusedw => RD1_Use
);

read_fifo2 : Sdram_FIFO port map(
        data => mData_OUT,
        wrreq => (OUT_Valid and RD_Mask(1)),
        wrclk => CLK,
        aclr => RD2_Load,
        rdreq => RD2,
        rdclk => RD2_CLK,
        q => RD2_Data,
        wrusedw => read_side_fifo_wusedw2,
        rdempty => RD2_Empty,
        rdusedw => RD2_Use
);

process(CLK)
begin
        if (ST = (mLength sla 3)) then
                SADD <= "000000000000";
        else
                SADD <= ISA;
        end if;

        br <= IBA;
        cs_l <= ICS_N;
        ClE <= ICKE;

        if (ST = (mLength sla 3)) then
                RlS_N <= '0';
        else
                RlS <= IRAS_N;
        end if;

        if (ST = (mLength sla 3)) then
                ClS_N <= '1';
        else
                ClS_N <= ICAS_N;
        end if;
        if (ST = (mLength sla 3)) then
                Wl_N <= '0';
        else
                Wl_N <= IWE_N;
        end if;

        if (ST = (mLength sla 3)) then
                PM_Stop <= '1';
        else
                PM_Stop <= '0';
        end if;

        if (ST = ((mLength sla 3) sla 2)) then
                PM_Done <= '1';
        else
                PM_Done <= '0';
        end if;

        if(active and (ST >= "0000000011")) then
                if ((ST = (mLength sla 3)) and Write) then
                        DQM <= "11";
                else
                        DQM <= "00";
                end if;
        else
                DQM <= "11";
        end if;
        mData_OUT <= DQ;
end process;

process(oe)
begin
        if(oe) then
                DQ <= DQOUT;
        else
                DQ <= "zzzzzzzzzzzzzzzz";
        end if;

        active <= Read or Write;
end process;

process(CLK, RESET_N)
begin
        if(RESET_N = 0) then
                CMD <= "00";
                ST <= "0000000000";
                Pre_RD <= '0';
                Read <= '0';
                OUT_Valid <= '0';
                IN_REQ <= '0';
                mWR_Done <= '0';
                mRD_Done <= '0';
        else
                Pre_RD <= mRD;
                Pre_WR <= mWR;
                case ST is
                        when 0 =>
                                if((Pre_RD&mRD) = "01")then
                                        Read <= 1;
                                        Write <= 0;
                                        CMD <= "01";
                                        ST <= 1;
```

```
                            elsif((Pre_WR&mWR) = "01")then
                                   Read <= 0;
                                   Write <= 1;
                                   CMD <= "10";
                                   ST <= 1;
                            end if;

                     when 1 =>
                            if( CMDACK = 1) then
                                   CMD <= "00";
                                   ST <= 2;
                            end if;

                     when others =>
                            if(ST /= (((mLength sla 3) sla 3) sla 1)) then
                                   ST <= ST sla 1;
                            else
                                   ST <= 0;
                            end if;
              end case;

              if(Read) then
                     if( ST = (((mLength sla 3) sla 3) sla 1)) then
                            OUT_Valid <= '1';
                     elsif(ST = (((mLength sla 3) sla 3) sla 1))then
                            OUT_Valid <= '0';
                            Read <= '0';
                            mRD_Done <= '1';
                     end if;
              else
                     mRD_Done <= '0';
                     if(ST = ("000000011" sra 1))then
                            IN_REQ <= '1';
                     elsif(ST = ((mLength sla 3) sra 1)) then
                            IN_REQ <= '0';
                     elsif(ST = (mLength sla 3))then
                            Write <= '0';
                            mWR_Done <= '1';
                     else
                            mWR_Done <= '0';
                     end if;
              end if;
       end if;
end process;

process(CLK,RESET_N)
begin
       if(not RESET_N) then
              rWR1_ADDR <= "0000000000000000000000";
              rWR2_ADDR <= "0100000000000000000000";
              rRD1_ADDR <= "0000000010100000000000";
              rRD2_ADDR <= "0100000010100000000000";
              rWR1_MAX_ADDR <= "0001010000000000000000";
              rWR2_MAX_ADDR <= "0101010000000000000000";
              rRD1_MAX_ADDR <= "0001001101100000000000";
              rRD2_MAX_ADDR <= "0101001101100000000000";
              rWR1_Length <= "100000000";
              rWR2_Length <= "100000000";
              rRD1_Length <= "100000000";
              rRD2_Length <= "100000000";
       else
              if(WR1_Load)then
                     rWR1_ADDR <= WR1_ADDR;
```

```
                                        rWR1_Length <= WR1_Length;
                                elsif(mWR_Done and WR_Mask(0)) then
                                        if(rWR1_ADDR < (rWR1_MAX_ADDR - rWR1_Length)) then
                                                rWR1_ADDR <= rWR1_ADDR+rWR1_Length;
                                        else
                                                rWR1_ADDR <= WR1_ADDR;
                                        end if;
                                end if;

                                if(WR2_Load)then
                                        rWR2_ADDR <= WR2_ADDR;
                                        rWR2_Length <= WR2_Length;
                                elsif(mWR_Done and WR_Mask(1)) then
                                        if(rWR2_ADDR < (rWR2_MAX_ADDR - rWR2_Length)) then
                                                rWR2_ADDR <= rWR2_ADDR+rWR1_Length;
                                        else
                                                rWR2_ADDR <= WR2_ADDR;
                                        end if;
                                end if;

                                if(RD1_Load)then
                                        rRD1_ADDR <= RD1_ADDR;
                                        rRD1_Length <= RD1_Length;
                                elsif(mRD_Done and RD_Mask(0)) then
                                        if(rRD1_ADDR < (rRD1_MAX_ADDR - rRD1_Length)) then
                                                rRD1_ADDR <= rRD1_ADDR+rRD1_Length;
                                        else
                                                rRD1_ADDR <= RD1_ADDR;
                                        end if;
                                end if;

                                if(RD2_Load)then
                                        rRD2_ADDR <= RD2_ADDR;
                                        rRD2_Length <= RD2_Length;
                                elsif(mRD_Done and RD_Mask(1)) then
                                        if(rRD2_ADDR < (rRD2_MAX_ADDR - rRD2_Length)) then
                                                rRD2_ADDR <= rRD2_ADDR+rRD2_Length;
                                        else
                                                rRD2_ADDR <= RD2_ADDR;
                                        end if;
                                end if;
                        end if;
                end process;

                process(CLK,RESET_N)
                begin
                        if(not RESET_N)then
                                mWR <= '0';
                                mRD <= '0';
                                mADDR <= "0000000000000000000000";
                                mLength <= "0000000000";
                        else
                                if(     (mWR=0)and(mRD=0)and(ST=0)and(WR_Mask=0)and
                                        (RD_Mask=0)and(WR1_Load)and(WR2_Load=0)and
                                        (RD1_Load=0)and(RD2_Load=0)) then

                                        if((write_side_fifo_rusedw1 >=
        rWR1_Length)and(rWR1_Length/=0))then
                                                mADDR <= rWR1_ADDR;
                                                mLength <= rWR1_Length;
                                                WR_Mask <= "01";
                                                RD_Mask <= "00";
                                                mWR <= '1';
                                                mRD <= '0';
                                        elsif((write_side_fifo_rusedw2 >= rWR2_Length) and
        (rWR2_Length /= 0)) then
                                                mADDR <= rWR2_ADDR;
                                                mLength <= rWR2_Length;
                                                WR_Mask <= "10";
                                                RD_Mask <= "00";
                                                mWR <= '1';
                                                mRD <= '0';
                                        elsif((read_side_fifo_wusedw1 < rRD1_Length)) then
                                                mADDR <= rRD1_ADDR;
                                                mLength <= rRD1_Length;
                                                WR_Mask <= "00";
                                                RD_Mask <= "01";
                                                mWR <= '0';
                                                mRD <= '1';
                                        elsif((read_side_fifo_wusedw2 < rRD2_Length)) then
                                                mADDR <= rRD2_ADDR;
                                                mLength <= rRD2_Length;
                                                WR_Mask <= "00";
                                                RD_Mask <= "10";
                                                mWR <= '0';
                                                mRD <= '1';
                                        end if;
                                end if;

                                if(mWR_Done)then
                                        WR_Mask <= "00";
                                        mWR <= '0';
                                end if;

                                if(mRD_Done) then
                                        RD_Mask <="00";
                                        mRD <= '0';
                                end if;
                        end if;
                end process;
        end datapath;
```

```
module CCD_Capture( oDATA,
                                oDVAL,
                                oX_Cont,
                                oY_Cont,
                                oFrame_Cont,
                                iDATA,
                                iFVAL,
                                iLVAL,
                                iSTART,
                                iEND,
                                iCLK,
                                iRST   );

input  [9:0]  iDATA;
input               iFVAL;
input               iLVAL;
input               iSTART;
input               iEND;
input               iCLK;
input               iRST;
output [9:0]  oDATA;
output [10:0] oX_Cont;
output [10:0] oY_Cont;
output [31:0] oFrame_Cont;
output              oDVAL;
reg                         Pre_FVAL;
reg                         mCCD_FVAL;
reg                         mCCD_LVAL;
reg           [9:0]  mCCD_DATA;
reg           [10:0] X_Cont;
reg           [10:0] Y_Cont;
reg           [31:0] Frame_Cont;
reg                         mSTART;

assign oX_Cont              =      X_Cont;
assign oY_Cont              =      Y_Cont;
assign oFrame_Cont   =      Frame_Cont;
assign oDATA         =      mCCD_DATA;
assign oDVAL         =      mCCD_FVAL&mCCD_LVAL;

always@(posedge iCLK or negedge iRST)
begin
      if(!iRST)
      mSTART <=     0;
      else
      begin
            if(iSTART)
            mSTART <=     1;
            if(iEND)
            mSTART <=     0;
      end
end

always@(posedge iCLK or negedge iRST)
begin
      if(!iRST)
      begin
            Pre_FVAL     <=     0;
            mCCD_FVAL    <=     0;
            mCCD_LVAL    <=     0;
            mCCD_DATA    <=     0;
            X_Cont       <=     0;
            Y_Cont       <=     0;
      end
      else
      begin
            Pre_FVAL     <=     iFVAL;
            if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
            mCCD_FVAL    <=     1;
            else if({Pre_FVAL,iFVAL}==2'b10)
            mCCD_FVAL    <=     0;
            mCCD_LVAL    <=     iLVAL;
            mCCD_DATA    <=     iDATA;
            if(mCCD_FVAL)
            begin
                  if(mCCD_LVAL)
                  begin
                        if(X_Cont<1279)
                        X_Cont <=     X_Cont+1;
                        else
                        begin
                              X_Cont <=     0;
                              Y_Cont <=     Y_Cont+1;
                        end
                        end
                end
                else
                begin
                        X_Cont <=     0;
                        Y_Cont <=     0;
                end
        end
end

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        Frame_Cont    <=     0;
        else
        begin
                if( ({Pre_FVAL,iFVAL}==2'b01) && mSTART )
                Frame_Cont    <=     Frame_Cont+1;
        end
end
```

```
module I2C_CCD_Config (    //     Host Side
                                        iCLK,
                                        iRST_N,
                                        iExposure,
                                        //     I2C Side
                                        I2C_SCLK,
                                        I2C_SDAT      );
//     Host Side
input               iCLK;
input               iRST_N;
input  [15:0] iExposure;
//     I2C Side
output        I2C_SCLK;
inout         I2C_SDAT;
//     Internal Registers/Wires
reg    [15:0] mI2C_CLK_DIV;
reg    [23:0] mI2C_DATA;
reg                 mI2C_CTRL_CLK;
reg                 mI2C_GO;
wire          mI2C_END;
wire          mI2C_ACK;
reg    [15:0] LUT_DATA;
reg    [5:0]  LUT_INDEX;
reg    [3:0]  mSetup_ST;

//     Clock Setting
parameter     CLK_Freq      =      50000000;    //     50     MHz
parameter     I2C_Freq      =      20000;       //     20     KHz
//     LUT Data Number
parameter     LUT_SIZE      =      17;

/////////////////////      I2C Control Clock   ////////////////////////
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                mI2C_CTRL_CLK <=     0;
                mI2C_CLK_DIV  <=     0;
        end
```

```
	else
	begin
		if( mI2C_CLK_DIV	< (CLK_Freq/I2C_Freq) )
		mI2C_CLK_DIV	<=	mI2C_CLK_DIV+1;
		else
		begin
			mI2C_CLK_DIV	<=	0;
			mI2C_CTRL_CLK	<=	~mI2C_CTRL_CLK;
		end
	end
end
////////////////////////////////////////////////////////////////////
I2C_Controller 	u0	(	.CLOCK(mI2C_CTRL_CLK),		//
	Controller Work Clock
						.I2C_SCLK(I2C_SCLK),		//
	I2C CLOCK
						.I2C_SDAT(I2C_SDAT),		//
	I2C DATA
						.I2C_DATA(mI2C_DATA),		//
	DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
						.GO(mI2C_GO),			//
	GO transfor
						.END(mI2C_END),
	//	END transfor
						.ACK(mI2C_ACK),
	//	ACK
						.RESET(iRST_N)	);
////////////////////////////////////////////////////////////////////
//////////////////////	Config Control	////////////////////////////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
begin
	if(!iRST_N)
	begin
		LUT_INDEX	<=	0;
		mSetup_ST	<=	0;
		mI2C_GO		<=	0;
	end
	else
	begin
		if(LUT_INDEX<LUT_SIZE)
		begin
			case(mSetup_ST)
			0:	begin
					mI2C_DATA	<=	{8'hBA,LUT_DATA};
					mI2C_GO		<=	1;
					mSetup_ST	<=	1;
				end
			1:	begin
					if(mI2C_END)
					begin
						if(!mI2C_ACK)
						mSetup_ST	<=	2;
						else
						mSetup_ST	<=	0;

						mI2C_GO		<=	0;
					end
				end
			2:	begin
					LUT_INDEX	<=	LUT_INDEX+1;
					mSetup_ST	<=	0;
				end
			endcase
		end
	end
end
/////////////////////////////////////////////////////////////////////
/////////////////////	Config Data LUT	  //////////////////////////
always
begin
	case(LUT_INDEX)
	0	:	LUT_DATA	<=	16'h0000;
	1	:	LUT_DATA	<=	16'h2000;
	2	:	LUT_DATA	<=	16'hF101;	//	Mirror Row and
Columns
	3	:	LUT_DATA	<=	{8'h09,iExposure[15:8]};// Exposure
	4	:	LUT_DATA	<=	{8'hF1,iExposure[7:0]};
	5	:	LUT_DATA	<=	16'h2B00;	//	Green 1 Gain
	6	:	LUT_DATA	<=	16'hF1B0;
	7	:	LUT_DATA	<=	16'h2C00;	//	Blue Gain
	8	:	LUT_DATA	<=	16'hF1CF;
	9	:	LUT_DATA	<=	16'h2D00;	//	Red Gain
	10	:	LUT_DATA	<=	16'hF1CF;
	11	:	LUT_DATA	<=	16'h2E00;	//	Green 2 Gain
	12	:	LUT_DATA	<=	16'hF1B0;
	13	:	LUT_DATA	<=	16'h0500;	//	H_Blanking
	14	:	LUT_DATA	<=	16'hF188;
	15	:	LUT_DATA	<=	16'h0600;	//	V_Blanking
	16	:	LUT_DATA	<=	16'hF119;
	default:LUT_DATA	<=	16'h0000;
	endcase
end
/////////////////////////////////////////////////////////////////////
endmodule
```

```
module I2C_Controller (
	CLOCK,
	I2C_SCLK,//I2C CLOCK
	I2C_SDAT,//I2C DATA
	I2C_DATA,//DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
	GO,      //GO transfor
	END,     //END transfor
	W_R,     //W_R
	ACK,      //ACK
	RESET,
	//TEST
	SD_COUNTER,
	SDO
);
	input  CLOCK;
	input  [23:0]I2C_DATA;
	input  GO;
	input  RESET;
	input  W_R;
	inout  I2C_SDAT;
	output I2C_SCLK;
	output END;
	output ACK;

//TEST
	output [5:0] SD_COUNTER;
	output SDO;


reg SDO;
reg SCLK;
reg END;
reg [23:0]SD;
reg [5:0]SD_COUNTER;

wire I2C_SCLK=SCLK | ( ((SD_COUNTER >= 4) & (SD_COUNTER <=30))? ~CLOCK :0 );
wire I2C_SDAT=SDO?1'bz:0 ;

reg ACK1,ACK2,ACK3;
wire ACK=ACK1 | ACK2 |ACK3;

//--I2C COUNTER
always @(negedge RESET or posedge CLOCK ) begin
if (!RESET) SD_COUNTER=6'b111111;
else begin
if (GO==0)
	SD_COUNTER=0;
	else
	if (SD_COUNTER < 6'b111111) SD_COUNTER=SD_COUNTER+1;
end
end
//----

always @(negedge RESET or  posedge CLOCK ) begin
if (!RESET) begin SCLK=1;SDO=1; ACK1=0;ACK2=0;ACK3=0; END=1; end
else
case (SD_COUNTER)
	6'd0  : begin ACK1=0 ;ACK2=0 ;ACK3=0 ; END=0; SDO=1; SCLK=1;end
	//start
	6'd1  : begin SD=I2C_DATA;SDO=0;end
	6'd2  : SCLK=0;
	//SLAVE ADDR
	6'd3  : SDO=SD[23];
	6'd4  : SDO=SD[22];
	6'd5  : SDO=SD[21];
	6'd6  : SDO=SD[20];
	6'd7  : SDO=SD[19];
	6'd8  : SDO=SD[18];
	6'd9  : SDO=SD[17];
	6'd10 : SDO=SD[16];
	6'd11 : SDO=1'b1;//ACK

	//SUB ADDR
	6'd12  : begin SDO=SD[15]; ACK1=I2C_SDAT; end
	6'd13  : SDO=SD[14];
	6'd14  : SDO=SD[13];
	6'd15  : SDO=SD[12];
	6'd16  : SDO=SD[11];
	6'd17  : SDO=SD[10];
	6'd18  : SDO=SD[9];
	6'd19  : SDO=SD[8];
	6'd20  : SDO=1'b1;//ACK

	//DATA
	6'd21  : begin SDO=SD[7]; ACK2=I2C_SDAT; end
	6'd22  : SDO=SD[6];
	6'd23  : SDO=SD[5];
	6'd24  : SDO=SD[4];
	6'd25  : SDO=SD[3];
```

```
		6'd26  : SDO=SD[2];
		6'd27  : SDO=SD[1];
		6'd28  : SDO=SD[0];
		6'd29  : SDO=1'b1;//ACK


		//stop
	6'd30 : begin SDO=1'b0;       SCLK=1'b0; ACK3=I2C_SDAT; end
	6'd31 : SCLK=1'b1;
	6'd32 : begin SDO=1'b1; END=1; end

endcase
end
```

```
module Mirror_Col(  //     Input Side
                                 iCCD_R,
                                 iCCD_G,
                                 iCCD_B,
                                 iCCD_DVAL,
                                 iCCD_PIXCLK,
                                 iRST_N,
                                 //     Output Side
                                 oCCD_R,
                                 oCCD_G,
                                 oCCD_B,
                                 oCCD_DVAL    );
//     Input Side
input  [9:0]  iCCD_R;
input  [9:0]  iCCD_G;
input  [9:0]  iCCD_B;
input               iCCD_DVAL;
input               iCCD_PIXCLK;
input               iRST_N;
//     Output Side
output [9:0]  oCCD_R;
output [9:0]  oCCD_G;
output [9:0]  oCCD_B;
output              oCCD_DVAL;
//     Internal Registers
reg           [9:0]  Z_Cont;
reg                        mCCD_DVAL;

assign oCCD_DVAL    =      mCCD_DVAL;

always@(posedge iCCD_PIXCLK or negedge iRST_N)
begin
       if(!iRST_N)
       begin
              mCCD_DVAL    <=     0;
              Z_Cont       <=     0;
       end
       else
       begin
              mCCD_DVAL    <=     iCCD_DVAL;
              if(Z_Cont<640)
              begin
                     if(iCCD_DVAL)
                     Z_Cont <=     Z_Cont+1'b1;
```

```
            end
            else
            Z_Cont <=     0;
      end
end

Stack_RAM (
                  .clock(iCCD_PIXCLK),
                  .data(iCCD_R),
                  .rdaddress(639-Z_Cont),
                  .wraddress(Z_Cont),
                  .wren(iCCD_DVAL),
                  .q(oCCD_R));

Stack_RAM (
                  .clock(iCCD_PIXCLK),
                  .data(iCCD_G),
                  .rdaddress(639-Z_Cont),
                  .wraddress(Z_Cont),
                  .wren(iCCD_DVAL),
                  .q(oCCD_G));

Stack_RAM (
                  .clock(iCCD_PIXCLK),
                  .data(iCCD_B),
                  .rdaddress(639-Z_Cont),
                  .wraddress(Z_Cont),
                  .wren(iCCD_DVAL),
                  .q(oCCD_B));

Endmodule
```

```
module RAW2RGB(     oRed,
                        oGreen,
                        oBlue,
                        oDVAL,
                        iX_Cont,
                        iY_Cont,
                        iDATA,
                        iDVAL,
                        iCLK,
                        iRST   );

input  [10:0] iX_Cont;
input  [10:0] iY_Cont;
input  [9:0]  iDATA;
input               iDVAL;
input               iCLK;
input               iRST;
output [9:0]  oRed;
output [9:0]  oGreen;
output [9:0]  oBlue;
output              oDVAL;
wire   [9:0]  mDATA_0;
wire   [9:0]  mDATA_1;
reg          [9:0]  mDATAd_0;
reg          [9:0]  mDATAd_1;
reg          [9:0]  mCCD_R;
reg          [10:0] mCCD_G;
reg          [9:0]  mCCD_B;
reg                       mDVAL;
```

```
assign oRed   =      mCCD_R[9:0];
assign oGreen =      mCCD_G[10:1];
assign oBlue  =      mCCD_B[9:0];
assign oDVAL  =      mDVAL;

Line_Buffer   u0     (      .clken(iDVAL),
                                          .clock(iCLK),
                                          .shiftin(iDATA),
                                          .taps0x(mDATA_1),
                                          .taps1x(mDATA_0)    );

always@(posedge iCLK or negedge iRST)
begin
       if(!iRST)
       begin
              mCCD_R <=     0;
              mCCD_G <=     0;
              mCCD_B <=     0;
              mDATAd_0<=    0;
              mDATAd_1<=    0;
              mDVAL  <=     0;
       end
       else
       begin
              mDATAd_0      <=     mDATA_0;
              mDATAd_1      <=     mDATA_1;
              mDVAL         <=     {iY_Cont[0]|iX_Cont[0]}    ?      1'b0   :
       iDVAL;
              if({iY_Cont[0],iX_Cont[0]}==2'b01)
              begin
                     mCCD_R <=     mDATA_0;
                     mCCD_G <=     mDATAd_0+mDATA_1;
                     mCCD_B <=     mDATAd_1;
              end
              else if({iY_Cont[0],iX_Cont[0]}==2'b00)
              begin
                     mCCD_R <=     mDATAd_0;
                     mCCD_G <=     mDATA_0+mDATAd_1;
                     mCCD_B <=     mDATA_1;
              end
              else if({iY_Cont[0],iX_Cont[0]}==2'b11)
              begin
                     mCCD_R <=     mDATA_1;
                     mCCD_G <=     mDATA_0+mDATAd_1;
                     mCCD_B <=     mDATAd_0;
              end
              else if({iY_Cont[0],iX_Cont[0]}==2'b10)
              begin
                     mCCD_R <=     mDATAd_1;
                     mCCD_G <=     mDATAd_0+mDATA_1;
                     mCCD_B <=     mDATA_0;
              end
       end
end
```

```
module VGA_Controller(     //     Host Side
                                          iRed,
                                          iGreen,
                                          iBlue,
                                          oRequest,
                                          //     VGA Side
                                          oVGA_R,
                                          oVGA_G,
                                          oVGA_B,
                                          oVGA_H_SYNC,
                                          oVGA_V_SYNC,
                                          oVGA_SYNC,
                                          oVGA_BLANK,
                                          oVGA_CLOCK,
                                          //      Control Signal
                                          iCLK,
                                          iRST_N );

  `include "VGA_Param.h"

  //      Host Side
  input         [9:0]  iRed;
  input         [9:0]  iGreen;
  input         [9:0]  iBlue;
  output reg                 oRequest;
  //      VGA Side
  output        [9:0]  oVGA_R;
  output        [9:0]  oVGA_G;
  output        [9:0]  oVGA_B;
  output reg                 oVGA_H_SYNC;
  output reg                 oVGA_V_SYNC;
  output                     oVGA_SYNC;
  output                     oVGA_BLANK;
  output                     oVGA_CLOCK;
  //      Control Signal
  input                      iCLK;
  input                      iRST_N;

  //      Internal Registers and Wires
  reg           [9:0]        H_Cont;
  reg           [9:0]        V_Cont;
  reg           [9:0]        Cur_Color_R;
  reg           [9:0]        Cur_Color_G;
  reg           [9:0]        Cur_Color_B;
  wire                       mCursor_EN;
  wire                       mRed_EN;
  wire                       mGreen_EN;
  wire                       mBlue_EN;

  assign oVGA_BLANK    =     oVGA_H_SYNC & oVGA_V_SYNC;
  assign oVGA_SYNC     =     1'b0;
  assign oVGA_CLOCK    =     iCLK;

  assign oVGA_R =      (     H_Cont>=X_START     && H_Cont<X_START+H_SYNC_ACT &&
                                          V_Cont>=Y_START     &&
  V_Cont<Y_START+V_SYNC_ACT )
                                          ?      iRed   :      0;
  assign oVGA_G =      (     H_Cont>=X_START     && H_Cont<X_START+H_SYNC_ACT &&
                                          V_Cont>=Y_START     &&
  V_Cont<Y_START+V_SYNC_ACT )
                                          ?      iGreen :      0;
  assign oVGA_B =      (     H_Cont>=X_START     && H_Cont<X_START+H_SYNC_ACT &&
                                          V_Cont>=Y_START     &&
  V_Cont<Y_START+V_SYNC_ACT )
                                          ?      iBlue  :      0;

  //      Pixel LUT Address Generator
  always@(posedge iCLK or negedge iRST_N)
  begin
          if(!iRST_N)
```

```
		oRequest	<=	0;
		else
		begin
			if(	H_Cont>=X_START-2 && H_Cont<X_START+H_SYNC_ACT-2 &&
				V_Cont>=Y_START && V_Cont<Y_START+V_SYNC_ACT )
			oRequest	<=	1;
			else
			oRequest	<=	0;
		end
	end

	//	H_Sync Generator, Ref. 25.175 MHz Clock
	always@(posedge iCLK or negedge iRST_N)
	begin
		if(!iRST_N)
		begin
			H_Cont		<=	0;
			oVGA_H_SYNC	<=	0;
		end
		else
		begin
			//	H_Sync Counter
			if( H_Cont < H_SYNC_TOTAL )
			H_Cont	<=	H_Cont+1;
			else
			H_Cont	<=	0;
			//	H_Sync Generator
			if( H_Cont < H_SYNC_CYC )
			oVGA_H_SYNC	<=	0;
			else
			oVGA_H_SYNC	<=	1;
		end
	end

	//	V_Sync Generator, Ref. H_Sync
	always@(posedge iCLK or negedge iRST_N)
	begin
		if(!iRST_N)
		begin
			V_Cont		<=	0;
			oVGA_V_SYNC	<=	0;
		end
		else
		begin
			//	When H_Sync Re-start
			if(H_Cont==0)
			begin
				//	V_Sync Counter
				if( V_Cont < V_SYNC_TOTAL )
				V_Cont	<=	V_Cont+1;
				else
				V_Cont	<=	0;
				//	V_Sync Generator
				if(	V_Cont < V_SYNC_CYC )
				oVGA_V_SYNC	<=	0;
				else
				oVGA_V_SYNC	<=	1;
			end
		end
	end

	endmodule
```