VGA Graphics Controller using SRAM Memory

TSTE12

Computer Technology, ISY

Linköpings Universitet

September 11, 2024

Table of Contents

1 VGA Graphics Controller Using SRAM Memory

1.1 Background

This lab will introduce the use of IP units as well as hierarchical schematic entry.

This section discusses the design of a graphics controller driving a computer monitor. Included is a description of the timing for the signals that drive a monitor and a description of an VHDL module that will let you drive a monitor with a picture stored in the SRAM memory.

The top level description of the design is shown below in Figure 1. The system consists of the computer monitor connected to the FPGA board. The FPGA board contains the SRAM with the image inside, the FPGA, and a 50 MHz clock source. Inside the FPGA a PLL structure (PLL65M) will create a 65 MHz clock used by all the logic (VGA Controller and Memory Interface). Two green LED will also indicate clock rate and reset state.

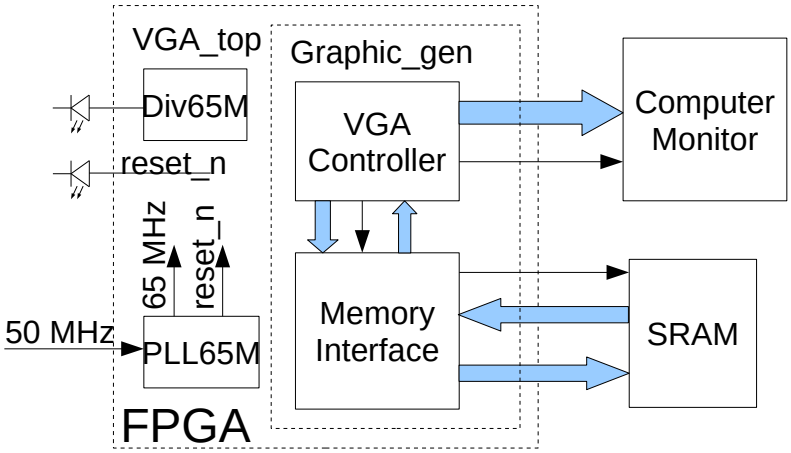

Figure 1: Overview of the complete system

2 VGA Graphics Controller (VGA_top)

The XGA resolution computer monitor is connected to the FPGA board through a 15 pin mini-DSUB VGA connector. This connector contains three analog color information signal, ground signals, and two synchronization signal HSYNC and VSYNC. The image to be shown is stored in the SRAM memory.

2.1 Requirements

Here are the requirements to pass the laboratory

- Implement the design using hdl_designer.
- The two leftmost 7-segment displays shall indicate your lab group number.
- The rightmost green LED should blink with a rate of 1 s and 50% duty cycle. The second to right should indicate current lock signal of the PLL65M. The blink rate should be controlled by the internal 65 MHz clock (not the 50MHz clock).
- Declare a symbol and corresponding VHDL view.
- Create the model in hdl designer with one block for each process.

- Call the top level VGA_top. Use at least three hierarchy levels including the top level symbol VGA_top. For example, divide the unit into a sync generation part and a color generation part.
- It is not allowed to have more than one clock domain (disregarding the PLL65M). The best way to check this is to not have more than one signal appearing on statements using ’event or the rising_edge() function call.
- Synthesize the design and demonstrate the function on the DE2-115 FPGA board.

2.2 VGA Color Signals

There are three signals -- red, green, and blue -- that send color information to a VGA monitor. In an CRT monitor drives each of these three signals an electron gun that emits electrons, which paint one primary color at a point on the monitor screen. Analog levels between 0 (completely dark) and 0.7 V (maximum brightness) on these control lines tell the monitor what intensities of these three primary colors to combine to make the color of a dot (or pixel) on the monitor's screen.

Each individual analog color input can be set to one of 2^8 (=256) levels by controlling the corresponding digital 8-bit input vector (either vga_r, vga_g, or vga_b to the digital-to-analog converters in the VGA DAC chip. The 256 possible levels on each analog input are combined by the monitor to create a pixel with one of 256x256x256 = 16 M different colors.

2.3 VGA Signal Timing

The monitor image is painted by controlling the focal point of the electron gun focus point (the color) using deflection circuits. The focal point is moved line by line on the monitor, starting from the top left corner and ending at the bottom right corner. The number of lines and the number of pixels on each line defines the resolution of the image, in this lab it will be 1024x768 pixels (XGA resolution). The deflection circuits require two synchronization signals in order to start and stop the deflection circuits at the right times so that a line of pixels is painted across the monitor and the lines stack up from the top to the bottom to form an image. The waveforms sent to the VGA is shown in Figure 2, with the expected frequency and times given in the DE2-115 user manual (see also Table 1, XGA(60Hz) is used in this lab).

Negative pulses on the horizontal sync signal mark the start and end of a line and ensure that the monitor displays the pixels between the left and right edges of the visible screen area. The actual pixels are sent to the monitor within a 15.75 us (1024/65e6) window. The horizontal sync signal

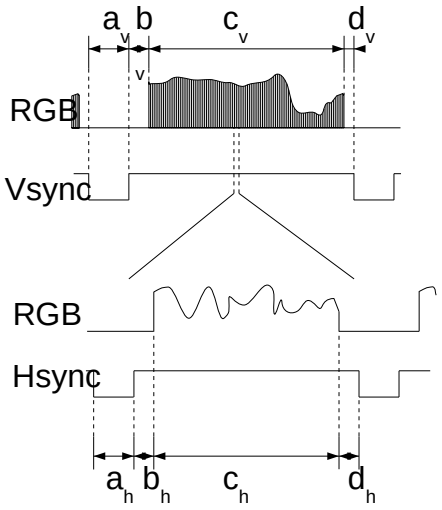

Figure 2: VGA waveforms

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(MHz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25
VGA(85Hz)	640x480	1.6	2.2	17.8	1.6	36
SVGA(60Hz)	800x600	3.2	2.2	20	1	40
SVGA(75Hz)	800x600	1.6	3.2	16.2	0.3	49
SVGA(85Hz)	800x600	1.1	2.7	14.2	0.6	56
XGA(60Hz)	1024x768	2.1	2.5	15.8	0.4	65
XGA(70Hz)	1024x768	1.8	1.9	13.7	0.3	75
XGA(85Hz)	1024x768	1.0	2.2	10.8	0.5	95
1280x1024(60Hz)	1280x1024	1.0	2.3	11.9	0.4	108

VGA mode		Vertical Timing Spec				
Configuration	Resolution(HxV)	a(lines)	b(lines)	c(lines)	d(lines)	Pixel clock(MHz)
VGA(60Hz)	640x480	2	33	480	10	25
VGA(85Hz)	640x480	3	25	480	1	36
SVGA(60Hz)	800x600	4	23	600	1	40
SVGA(75Hz)	800x600	3	21	600	1	49
SVGA(85Hz)	800x600	3	27	600	1	56
XGA(60Hz)	1024x768	6	29	768	3	65
XGA(70Hz)	1024x768	6	29	768	3	75
XGA(85Hz)	1024x768	3	36	768	1	95
1280x1024(60Hz)	1280x1024	3	38	1024	1	108

Table 1: Detailed timing for various screen resolutions

drops low a minimum of 0.37 us (24/65e6) after the last pixel and stays low for 2.1 us (136/65e6). A new line of pixels can begin a minimum of 2.46 us (160/65e6) after the horizontal sync pulse ends. So a single line occupies 15.75 us of a 20.68 us interval. The other 4.93 us of each line is the horizontal blanking interval during which the screen is dark.

In an analogous fashion, negative pulses on a vertical sync signal mark the start and end of a frame made up of video lines and ensure that the monitor displays the lines between the top and bottom edges of the visible monitor screen. The lines are sent to the monitor within a 15.88 ms window. The vertical sync signal drops low a minimum of 62 us (3 lines) after the last line and stays low for 0.124 ms (6 lines). The first line of the next frame can begin a minimum of 0.60 ms (29 lines) after the vertical sync pulse ends. So a single frame occupies 15.88 ms of a 16.67 ms interval. The other 0.79 ms of the frame interval is the vertical blanking interval during which the screen is dark.

2.4 VGA Signal Generator Algorithm

We now have to figure out a process that will send pixels to the monitor with the correct timing and framing. We can store a picture in the SRAM of the DE2-115 Board. Then we can retrieve the data from the SRAM, format it into lines of pixels, and send the lines to the monitor with the appropriate pulses on the horizontal and vertical sync pulses.

An example of pseudocode for a single frame of this process is shown in Figure 3. The pseudocode has two outer loops: one which displays the L lines of visible pixels, and another which inserts the V blank lines and the vertical sync pulse. Within the first loop, there are two more loops: one which

sends the P pixels of each video line to the monitor, and another which inserts the H blank pixels and the horizontal sync pulse.

```
for line_cnt=1 to L             /* send L lines of video to the monitor */
        for pixel_cnt=1 to P            /* send P pixels for each line */
                data = RAM(address)         /* get pixel data from the memory */
                address = address + 1       /* FLASH data word contains 4 pixels */
                color = COLOR_MAP(data)     /* get the color for the right-bit pixel */
                send color to monitor
                pixel_cnt = pixel_cnt + 1
        for horiz_blank_cnt=1 to H          /* blank the monitor for H pixels */
                color = BLANK
                send color to monitor
                /* pulse the horizontal sync at the right time */
                if horiz_blank_cnt>HB0 and horiz_blank_cnt<HB1
                        hsync = 0
                else
                        hsync = 1
                horiz_blank_cnt = horiz_blank_cnt + 1
        line_cnt = line_cnt + 1
for vert_blank_cnt=1 to V           /* blank the monitor for V lines and insert vertical sync */
        color = BLANK
        send color to monitor
        /* pulse the vertical sync at the right time */
        if vert_blank_cnt>VB0 and vert_blank_cnt<VB1
                vsync = 0
        else
                vsync = 1
        vert_blank_cnt = vert_blank_cnt + 1
/* go back to start of picture in memory */
address = 0
```

Figure 3: VGA signal generation pseudocode

Within the pixel display loop, there are statements to get the next word from the SRAM. Each word contains one pixel. Since it has only 16 bits, each pixel can store one of 65536 levels of color or gray. The mapping from the 16 bit pixel value to the actual values required by the monitor electronics is done by the COLOR_MAP() routine. In this design we only make use of 65536 colours, with 6 bits red (bits 15 downto 10) and 5 bits each for green (bits 9 downto 5) and blue (4 downto 0).

Reading memory will take one clock cycle (applying the address at one rising clock edge, and reading data at the next rising clock edge). Applying the color mapping and sending the pixel to the pins may take additional time. It is therefore important to understand the timing of the design, especially related to the blanking signals. It may therefore be necessary to add additional delay to the blanking signals. Failing this may cause the image to lack color on the edges of the image or have duplicated pixels on the image edges.

2.5 Problem definition

Here are the definitions listed that are needed to complete the design.

2.5.1 Port definitions

The inputs and outputs of the circuit as defined are as follows:

Name	Mode	Type and Range	Description
fpga_clk	IN	std_logic	The system clock
fpga_reset_n	IN	std_logic	The circuit reset signal. Reset is active low, i.e., fpga_reset_n='0' gives reset
vga_clk	OUT	std_logic	The DAC clock signal. Typically pixel clock signal.
vga_sync	OUT	std_logic	Not in use. Inactivate this with a logical '0'
vga_blank_n	OUT	std_logic	The blank signal from the design.
vga_r	OUT	std_logic_vector 7 downto 0	The red component of the display rgb signal. Always set the unused lower 2 bits to '0'
vga_g	OUT	std_logic_vector 7 downto 0	The green component of the display rgb signal. Always set the unused lower 3 bits to '0'
vga_b	OUT	std_logic_vector 7 downto 0	The blue component of the display rgb signal. Always set the unused lower 3 bits to '0'
vga_hsync_n	OUT	std_logic	The display horizontal sync pulse, active low
vga_vsync_n	OUT	std_logic	The display vertical sync pulse, active low
sram_data	IN	std_logic_vector 15 downto 0	The display data from SRAM
sram_address	OUT	std_logic_vector 19 downto 0	The address to display data SRAM. Always set unused bits to '0'
sram_we_n	OUT	std_logic	SRAM write enable. Set to '1' while reading image
sram_oe_n	OUT	std_logic	SRAM output enable. Set to '0' while reading image
sram_ce_n	OUT	std_logic	SRAM chip select. Set to '0' while reading image
sram_lb_n	OUT	std_logic	SRAM lower byte strobe. Set to '0' while reading image
sram_ub_n	OUT	std_logic	SRAM upper byte strobe. Set to '0' while reading image
HEX7	OUT	std_logic_vector 6 downto 0	Most Significant Digit of your lab group number. See lab1 for more information
HEX6	OUT	std_logic_vector 6 downto 0	Least Significant Digit of your lab group number. See lab1 for more information.
GLED0	OUT	std_logic	Green LED indicating PLL65M lock state
GLED1	OUT	std_logic	Green LED blinking once every second

Pin location for these signals can be found in the board documentation available at /courses/TSTE12/material/DE2-115_SystemCD_v3.0.6/DE2_115_user_manual/DE2_115_User_manual.pdf.

2.5.2 Interface

The synthesis tool needs to know how the FPGA is connected to the external world. An attribute description included in the the VHDL description will be used for this purpose. How to include this into the design was described in the keyboard exercise.

With guidance from the port definitions each group has to create its own attribute description. All necessary information is found in the documentation of the "DE2-115 education board - users manual".

2.5.3 Top-level Structure of the VGA Signal Generator (VGA_top)

The pseudocode and pipeline timing in the last section will help us to understand the structure and create the VHDL code for a VGA signal generator.

2.5.3.1 Inputs and outputs

fpga_clk

The input for the 50 MHz clock of the DE2-115 board. It should only be used as input to the PLL65M that will use this clock as a reference and create a new 65 MHz clock that will be used in the rest of the design.

fpga_reset_n

Reset the PLL65M. Will stop the clock generation when fpga_reset_n = '0'. Connect this to one of the push buttons on the board. Note that the PLL65M areset input is active high, which means that the fpga_reset_n signal must be inverted before it is used by the PLL65M.

vga_clk

The vga_clk is used by the DAC to clock the individual pixels. Create this by inverting the internal 65 MHz clock. The pixel and sync signals should then be stable when the positive edge of vga_clk reaches the DAC chip.

vga_hsync_n, vga_vsync_n

The outputs for the horizontal and vertical sync pulses.

vga_blank_n

The output for the display blank signal. A '0' on this signal forces the DAC-chip to blank its outputs.

vga_r, vga_g, vga_b

The outputs that is used by the DAC chip to create the red, green, and blue analog color gun signals. Each pixel in the memory encodes the pixel color as three parts, <r5...r0 g4...g0 b4...g0>, the 6 leftmost bits (MSB bits) should be used as MSB bits for vga_r, etc. The remaining unused lower bits in the vga_r, vga_g and vg_b output should be set to '0'.

SRAM_address, SRAM_data

The outputs for driving the address lines of the SRAM memory and the inputs for receiving the data from the SRAM.

SRAM_oe_n, SRAM_we_n, SRAM_ce_n,SRAM_lb_n, SRAM_ub_n

Control signals for the SRAM interface. Forces the SRAM to be read all the time, using 16 bit data interface. More information is available in the DE2-115 users manual and SRAM datasheet.

2.5.3.2 VGA_top Internal Signals

Signal can be added directly into the block diagram. The name and type of these signals are automatically set to be equal to the name and type of the port that they are connected to. Names can be changed by double-clicking on the name in the block diagram or on the wire itself.

fpga_clock_65M

Internal FPGA clock generated by the PLL65M component. This should be the only clock signal used by the rest of the design (excluding the 50 MHz clock used as input to the PLL65M component). It is connected to the c0 output of the PLL65M.

reset_n

Internal reset signal. This is a copy of the locked signal from the PLL65M. The rest of the design should be reset when reset_n is '0'. This should also be shown on one of the green LEDs.

2.5.3.3 Subblocks in the VGA_top block diagram

These three components should be placed in the top level, that is, the structural description of the VGA_top component. The two components Div65M and Graphic_gen is added by simply selecting the blue component block symbol, while the PLL65M must first be created separately, and then added to the structure.

Div65M

Divides the 65MHz clock frequency by 65 000 000, thereby producing a alternating output signal blinking with a 1 second cycle. Build this as a counter that counts from 65 000 000/2 downto 0, and when reaching zero inverts the output bit and restarts counting.

Graphic_gen

The part of the design that generates monitor control signals, generates an SRAM memory address, and output pixel color based on the data from the SRAM memory. Details are described later.

PLL65M

This is an example of an IP block. This component creates a 65 MHz clock signal used in the rest of the design. It is based on a hardware block in the FPGA consisting of both analog and digital circuitry such as voltage controlled oscillators, dividers, filters etc. It is able to create and synchronize multiple clocks using one input clock as a reference.

The IP block will not be synthesized is the usual way by creating a netlist of lookup tables and flip flops. The IP will instead consist of a simulation model useful for verifying the function using simulation, and a post-synthesis configuration information used to configure the PLL hardware in the FPGA.

The IP block has been created separately in Quartus Prime, and will be imported into the lab design. The import will be done from within HDL designer by first pressing on Tasks/templates on the right edge of the Design manager window. Double click on the Quartus Prime Import icon as shown in Figure 4.

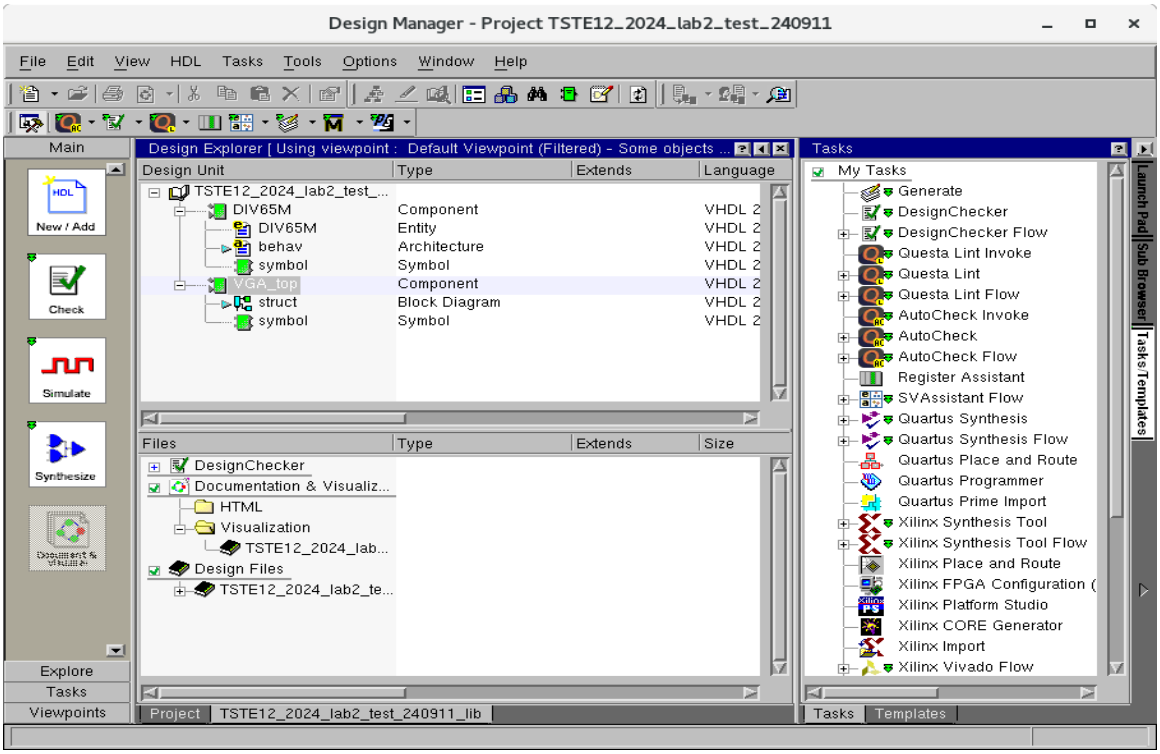

Figure 4: Quartus Prime Import available in the middle of the right column

The dialog window shown in Figure 5 appears. Press the Browse button and find the PLL65M.qip file in the folder /courses/TSTE12/material/PLL65M, and then press Open. Finally press OK to perform the import.

The new component PLL65M should now be visible in project library in the design manager.

Adding the generated PLL into the block diagram of VGA_top is done by first selecting the green component symbol at the top of the VGA_top struct block diagram window, then drag and drop the PLL component from the component browser window into the block diagram. Select no if asked about adding an additional library definition of altera_mf.

The altera_mf library is used in the simulation model of the PLL65M component, but due to the complex simulation model the simulation time will be long. You are therefore not expected to run simulations that include the PLL65M component.

2.5.4 Structure of the Graphics_gen component

The Graphic_gen component contains the logic that controls the memory and the computer monitor. It should consist of the two blocks Memory_interface and VGA_controller. The Memory interface shall produce the SRAM address and control signals, while VGA_controller generates monitor control signals and translates the incoming sram_data into red, green and blue color data.

The SRAM controller needs information about where on the screen the pixel should be located. There are two main approaches to this problem; either calculate an index based on current pixel and line counters, or use the fact that pixels are located sequentially in memory and use a simple counter.

The following description shows one way to implement this functionality.

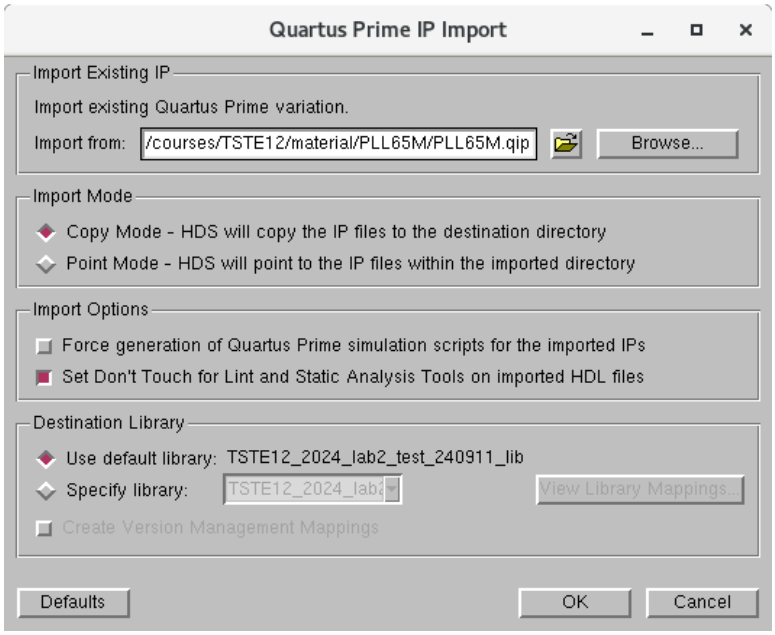

Figure 5: Quartus Prime Import window with the PLL component selected

2.5.4.1 Graphics_gen Internal Signals

hcnt, vcnt or offset or vsync, blank

Send current pixel position using hcnt and vcnt (require address calculation in the receiving block) or send a precalculated offset, or send blank and vsync signals to allow a counter in the receiving block reset when vsync is activated and increment when not blanked.

2.5.4.2 Subblocks in Graphics_gen

The Graphic_gen schematic should contain at least two components, VGA_controller and Memory_Interface.

VGA_controller

The VGA_controller block generates the control signals to the screen (sync, pixelvalues etc),

Memory_Interface

The Memory_Interface block generates control signals to the SRAM. These consists of static control signals such as SRAM_ce_n, SRAM_oe_n etc that will make the SRAM always read. The Memory_Interface block is generating the SRAM_address.

2.5.5 Structure of the VGA_Controller

The VGA_Controller implements most of the functionality shown in Figure 3: VGA signal generation pseudocode. This must first be translated into a number of different processes.

2.5.5.1 VGA_Controller internal signals

Some of the signals may be needed as outputs from the VGA_Controller.

hcnt, vcnt

The counters that store the current horizontal position within a line of pixels and the vertical position of the line on the screen. We will call these the horizontal or pixel counter, and the vertical

or line counter, respectively. The line period is 20.68 µs that is 1344 vga_clk cycles, so the pixel counter needs at least eleven bits of resolution. Each frame is composed of 806 video lines (only 768 are visible, the other 38 are blanked), so a ten bit counter is needed for the line counter.

pixrg

The 16-bit register that stores the pixel received from the SRAM.

hblank, vblank

The video blanking signal and its registered counterpart that is used in the next pipeline stage. These signals indicate if a pixel should be output on the screen at any given moment.

2.5.5.2 Subblocks in VGA_Controller

The VGA_Controller should be described using a block diagram, where each block is a separate process. The block diagram will then clearly show how the different processes communicate with eachother.

pixelcounter

This process describes the operation of the horizontal pixel counter. The counter is asynchronously set to zero when the fpga_reset_n is applied. The counter increments on the rising edge of each fpga_clock_65M cycle. The range for the horizontal pixel counter is [0,1343]. When the counter reaches 1343, it rolls over to zero on the next cycle. Thus, the counter has a period of 1344 pixel clocks. With a pixel clock of 65 MHz, this translates to a period of 20.68 µs.

linecounter

This process describes the operation of the vertical line counter. The counter is asynchronously set to zero when the reset input is low. The counter increments when the rising edge of the horizontal sync pulse is detected, that is, after a line of pixels is completed. The range for the vertical line counter is [0,805]. When the counter reaches 805, it rolls over to zero on the next cycle. Thus, the counter has a period of 806 lines. Refer to the the tables in the end of 2.3, “ VGA Signal Timing” for more information on the length of each subpart.

hsyncr

This process describes the operation of the horizontal sync pulse generator. The horizontal sync is set to its inactive high level when the reset is activated. During normal operations, the horizontal sync output is updated on every pixel clock. Refer to the the tables in the end of 2.3, “ VGA Signal Timing” for more information on the length of each subpart.

Here is also the hblank signal created. The video is blanked after the 1024 pixels of a line are displayed. The blanking signal is active high.

vsyncr

This process describes the operation of the vertical sync pulse generator. The vertical sync is set to its inactive high level when the reset is activated. During normal operations, the vertical sync output is updated after every line of pixels is completed. Refer to the the tables in the end of 2.3, “ VGA Signal Timing” for more information on the length of each subpart.

Here is also the vblank signal created. The video is blanked after 768 lines are displayed. The blanking signal is active high.

blank_syncr

This process describes the operation of the pipelined video blanking signal. Within the process, the blanking signal is stored in a register so it can be used during the next stage of the pipeline when the color is computed.

Computation of the combinatorial blanking signal. The total blank is calculated as vga_blank_n = not (hblank OR vblank).

pixel_reg

This process describes the operation of the register that holds the word (16 bits) of pixel data read from SRAM. The register is asynchronously cleared when the VGA circuit is reset. The register is updated on the rising edge of each fpga_clock_65M.

vga_gen

This process describes the process by which the current active pixel is mapped into the 24 bits that drive the red, green and blue color guns. The register is set to zero (which displays as the color black) when the reset input is low. The color register is clocked on the rising edge of the fpga_clock_65M clock since this is the rate at which new pixel values arrive. The value clocked into the register is a function of the pixel value and the blanking input.

When the pipelined blanking input is low (inactive), the color displayed on the monitor is a color value depending the input value from SRAM. This means we send different parts of the SRAM value to all the RGB inputs to achieve a color image. When the pipelined blanking input is high, the color register is loaded with zero (black).

2.5.6 Structure of the SRAM_controller

The SRAM is permanently selected and writing to the SRAM is disabled. It stores the video data. The image we will display is stored in a linear sequence of words inside the SRAM. The address to the SRAM is easiest to create by use of a linear counter that is updated every clock cycle whenever the hblank is not active. The counter can be reset to 0 whenever the vertical blanking is active.

2.6 Tips and tricks

To aviod a lot of trouble in the design phase a few hints are useful

- Do not use multiple clock definitions, i.e. , use only one signal (fpga_clock_65M) with ’EVENT or rising_edge() to define a clock. The use of ’EVENT and ’LAST_VALUE are not useful in the context of synthesis except for defining the rising or falling edge of the clock fpga_clock_65M.
- Let the top level design only contain the PLL, the LED driver, and one block containing the total VGA controller. This allows the VGA simulation to be run without having to simulate the PLL (the PLL is time consuming to simulate and require special libraries)
- Use a simple model of the SRAM contents to figure out if the timing is correct. The simulation model of the SRAM should have a signal update delay of approximately 4 ns, and give different data outputs for each address, e.g., copy the 16 least significant bits of the address to the data. Use this in a testbench (that you create) and look at the red, green and blue signals together with the blank and sync signals.
- The system asks if you want to add altera_mf as a library in the design. Do not let it to do this (answer no).

- The simulation complains about missing altera_mf library. This is caused by the PLL structure requiring this library. The solution is to not simulate the PLL. Simulate only the graphics_gen block, and feed it a 65MHz clock directly.
- The synthesis may complain about attribute values in std_logic_vector pin assignment. This may be caused by mixing different length strings such as “B3”,”A23”. This problem is solved by making all substrings the same length, by adding a space at the end. E.g., “B3 “,”A23”.

2.7 Synthesis flow (using Precision)

The synthesis should in this lab target the altera DE2-115 board. The setup is therefore slightly different compared to the setup in lab1.

2.7.1 Adding the pin attributes

The pin attributes are added in the same way as in lab1, but the definitions are slightly different. Use the definitions in the file /courses/TSTE12/material/Lab_Keyboard/keyboard_DE2-115_attributes.txt and remove non-used attributes and add attributes for the new signals. Exact pin locations are found in the board documentation at /courses/TSTE12/material/DE2-115_SystemCD_v3.0.6/DE2_115_user_manual/DE2_115_User_manual.pdf.

2.7.2 Synthesize the Design

The design flow buttons include several steps in one command. There are flows defined for simulation and synthesis. The synthesis assume a validated and compiled design.

Select the VGAtop design unit in the design browser.

Start synthesis by choosing the synthesis flow button. You find this next to the compile and generate toolbar buttons. Remember to select the version with two green arrows/triangels in the symbol.

The Precision Synthesis settings appear. Make sure to change the following in the setup (see also Figure 6).

Technology:	Altera/Cyclone IV E
Device:	EP4CE115F29C
Speed Grade:	7
Design frequency:	50 MHz
Run Options:	Run Integrated Place and Route should be selected
Implementation options	Single Implementation Mode

Press ok. The synthesis tool is now started.

The synthesis now starts and generates a netlist, various reports including used pins, timing estimates etc., and a sof FPGA configuration file. The synthesis results are located in a subfolder to /courses/TSTE12/labs/labgrp<nn> depending of what names you chose for the library and design. The sof programming file is located in a subdirectory inside the synthesis folder (ps).

2.8 Downloading the Design

The board is always programmed from a computer using an USB connection. The programming is carried out using a special programming software from the FPGA vendor.

One approach to program the board is therefore to login and run the programming software on the computer connected to the FPGA board. This is however not very useful if the board is shared by multiple lab groups.

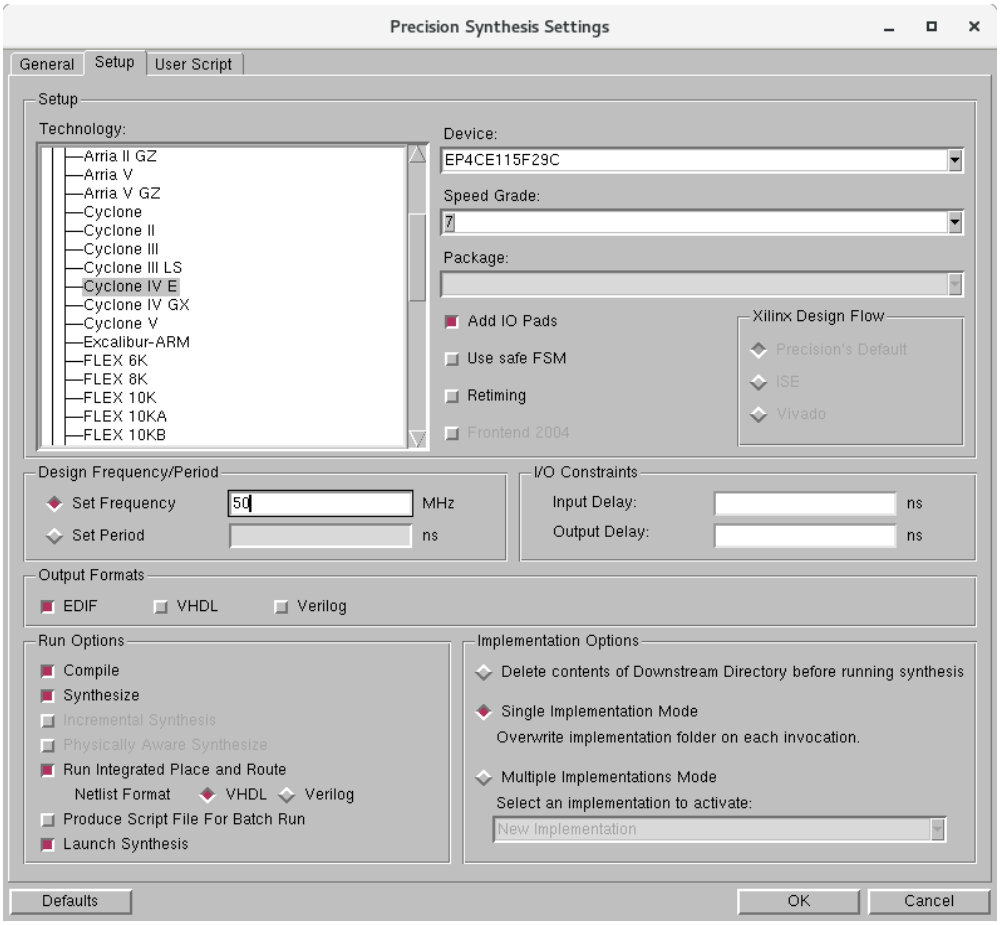

Figure 6: Synthesis settings dialog window

The board may also be accessed over the network. The programming software is then run on the same computer that is used for development. Instead of selecting the USB port on the development computer is a connection made from within the programming software to the other computer, thereby controlling the board remotely. The only difference is how to select the USB port inside the programming software.

2.8.1 Programming the FPGA

These steps are for programming the DE2-115 FPGA board.

1. Start Quartus II Programmer by entering the following command inside the mentorskal window
   quartus_pgmw
2. Press the Add file button. Select your sof programming file.
3. Then make sure that the USB-blaster and JTAG has been selected as the hardware to use. This is shown close to the the top. If the correct board is shown go to step 7 below.
4. If the wrong board or no board is connected, press the “Hardware Setup” button.
5. If a networked board is used select the JTAG Settings tab. Press Add Server to connect to the computer with the board.

6. Remember to select a board in the “currently selected hardware entry” before closing the Hardware setup window.
7. Select and mark the box Programming/Configure on the right of the file.
8. Then press start button to make the FPGA programmed. The new design will be downloaded within seconds.
9. Try out the hardware. Verify that it is your group number showing on the leftmost 7-segment LEDs.
10. Press keys and/or observe results on LEDs and the attached VGA screen.

3 Document history

240911

Support new HDL designer 2024.1, quartus 23.1, modifying PLL import

200912

Add details about synthesis and programming

190913

Change to new CentOS 7 installation and Quartus 17.1. Remove megacore creation of PLL, use only copy of existing IP.

170825

Restructure description, detail Graphic_gen structure

120911

Small fixes. Do not add altera_mf to default package list. Separate green led pins.

120620

Add PLL, change resolution to 1024x768 with 1 pixel/word, switch to DE2-115 board using SRAM.

100909 V0.2

Updated pin names to avoid clash with reset_n and clarify active level by adding _n

Fix pseudo code to indicate 4 pixels / word

Clarify the polarity of vga_blank_n

100902 V0.01

First version addressing the DE2-70 using FLASH