EITF35 - Introduction to Structured VLSI Design (Fall 2013)

Course projects

v.1.0.0

1 Introduction

This document describes the course projects provided in EITF35 “Introduction to Structured VLSI Design” conducted at EIT, LTH. The student may choose from two different projects to obtain grade 4 and has to complete both of them to obtain a grade 5. Both projects are extensions to the lab assignments 2 and 3, where the ALU and PS/2 keyboard controller need to be integrated to the whole system along with additional components. The basic requirement for projects is that the result obtained has to be displayed on a computer screen interfaced to the FPGA using the VGA port. A block diagram for the top level is shown in Fig 1.

* Project 1 (*Grade 4*) - A calculator with memory:
  Deadline Oct. 18

  Instantiate an 8 kB, 8 bit wide RAM in your design using Xilinx LogiCORE IP generator. Integrate the keyboard, ALU, VGA and the newly created IP into your design. The design should be able to input operands from the keyboard, store them into the RAM and later calculate the result and display the result on the VGA.

* Project 2 (*Grade 5*) - Integrated ALU with memory and a square root unit:
  Deadline Oct. 30

  The design should be able to get operands from the keyboard and store them in the RAM. Along with performing the already implemented operations, the ALU should be able to compute the square root of one operand with upto 3 digits in accuracy after the decimal point. The result should be displayed on the seven segment display as well as the VGA screen.

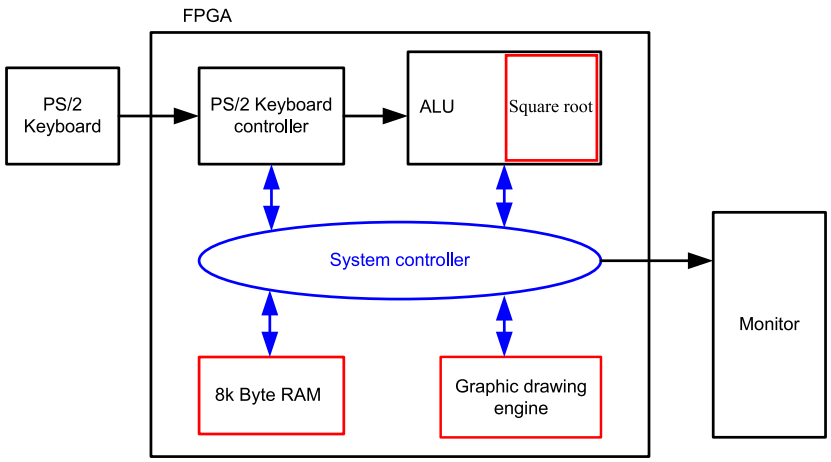


Figure 1: An overview on the course projects

To ease the start of the project, a reference design of a VGA controller on the target FPGA board is provided, where a course welcome message is loaded from FPGA's block memories and displayed on the monitor. The student can modify this design to suit the requirements of whichever project he/she chooses to implement.

Lab preparation

- Read this manual and try to understand the given tasks. Make sure that you have understood what is expected from both projects before choosing a topic. Consult the lab assistants, if the functionality or any task is not expressed clear enough.
- Choose a course project.
- Read VGA section of the FPGA user manual, and go through the provided VGA controller reference design. Understand how a VGA controller works, and read about generation of IP cores using Xilinx LogiCORE IP generator.

Equipments

- A Xilinx Spartan-3 FPGA board, with a mounted FPGA device - Xilinx "XC3S200", package type "FT256", and speed grade "-4".
- A PC monitor with a standard VGA port.
- A PS/2 interfaced keyboard.

2 The VGA reference design

In both course projects, a VGA display with pixel resolution of 640×480@60 Hz is used. The VGA port connections, VGA color signals and basic timing specification may be found in the user guide of the FPGA board provided by Xilinx (please find the file under:

"S:\course_projects\datasheets\S3BOARD_RM.pdf",

on page 21). Therefore, descriptions for these parts are not repeated in this manual, whereas only the VGA signal timing diagram is illustrated here as shown in Fig. 2.

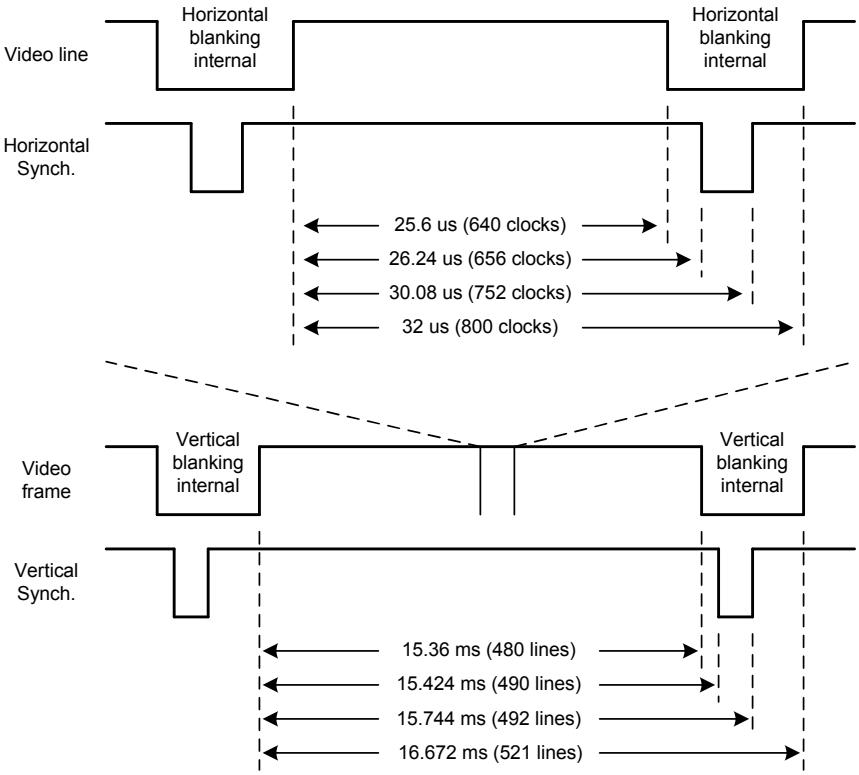


Figure 2: Signal timing diagram for a 60 Hz, 640×480 VGA display.

To illustrate the use of the given signal timing information, a reference design of the VGA controller is provided in this course and is briefly described in this manual. The reference design displays a course welcome message on a VGA display, where the message is saved as an image file stored in the block memories of the FPGA. An overview of the provided VGA controller is shown in Fig. 3.

A) DCM (Digital Clock Management): This module divides the input clock frequency by a factor of 2, as the provided VGA controller is designed based on a system clock of 25 MHz. The DCM unit is a primitive component available in Xilinx's FPGAs, which may be generated from Xilinx ISE environment with the use of IP core generator. The way of generating and properly configuring the Xilinx DCM core is shown as a video clip, placed under:

"*S:\tutorials\ise_clock_rom.wmv*",

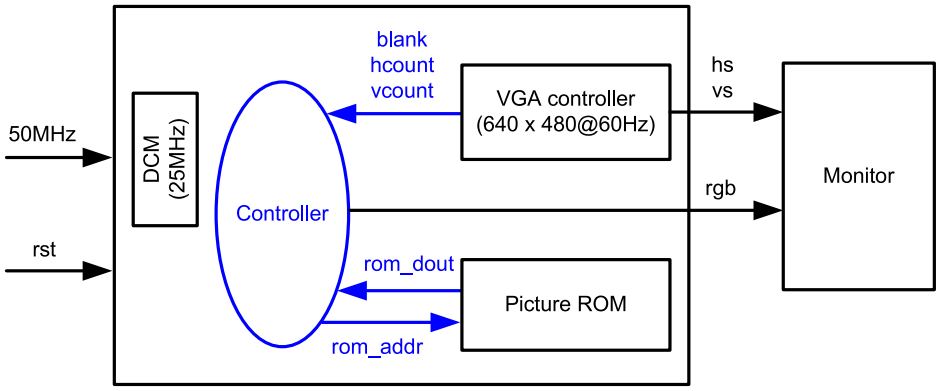


Figure 3: Block diagram of the VGA controller reference design.

B) Picture ROM: This is the place where the welcome message is stored. The message is saved as a bitmap image and is stored inside FPGA's data ROMs. Data ROMs may be generated with the use of Xilinx IP cores, however, the input data files have to be loaded in a ".coe" file format during the ROM generation. This may be accomplished by using the software provided - "imageConverter", placed under:

   "*S:\course_projects\imageConverter*".

   A bitmap image conversion is shown in a video clip "*image_converter.wmv*", and ROM generation is shown in "*ise_clock_rom.wmv*", both placed under:

   "*S:\tutorials*".

C) VGA controller: This module contains two binary counters, used for tracking on the horizontal video pixels and vertical video lines, respectively. Horizontal and vertical synchronization pulses for the VGA display are generated based on the counters, and an additional blank signal is provided as an output to indicate the VGA blanking time interval.

D) Controller: The system controller keeps tracking on the current VGA pixel position by using the horizontal and vertical counter values provided from the VGA controller. This module also controls the address of picture ROM, and reads out the image data at the desired pixel locations. 3-bit color codes with one bit each for red, green, and blue are sent to the VGA display, resulting in having 8 different color tones.

Notice that physical pins mappings of the system I/O signals on an FPGA are accomplished with the use of a constraint file, namely the ".ucf" file, which is added in the project structure.

3 Course project 1 (*Grade 4*) - A calculator with memory

In this project, the ALU implemented in lab assignment 3 and PS/2 keyboard controller designed in lab assignment 2 will be reused. A new IP will be generated using the Xilinx IP generator tool.

3.1 Task 1

Start by first understanding how the given VGA controller works. Try assigning your own rgb colors to the display instead of ROM data. Figure out how the vertical and horizontal counters can be used in order to emulate the seven segment display on the LCD. An illustration of the LCD display required is shown in a screen capture, placed under:

"*S:\course_projects\rtl_ref_designs\project_1.jpg*".

Integrate the VGA controller to the keyboard and the ALU. Use a top level file to instantiate these three IPs as components in order to keep them functionally in separate files. Reuse the binary to bcd function/component which was used to convert the inputs from the keyboard into seven segment display. Reuse as much code as possible.

3.2 Task 2

Once Task 1 is done you can move onto generating your own memory module. The basic steps of generating an IP core are listed below.

* Right click in the design hierarchy window -> New Source.
* Choose IP(Core generator and architecture wizard). Name your memory module.
* In Memories & Storage elements, choose *RAMs & ROMs*. Then choose Block memory generator.
* In the new window that opens up, examine the memory block that will be generated.
* Choose a Single port RAM with the Algorithm set to Minimum Area.
* Set Memory write width to 8 bits and write depth to 8 kB.
* Leave all other options unchanged. Generate memory.

Once this is done, a new IP will appear in your design hierarchy window. Examine the HDL files generated by clicking on the IP and choosing the View HDL functional model. The component instantiation that needs to be used in your calculator design can be found in the HDL instantiation file.

If the memory module is generated as specified above, it will have 5 ports. Clock, write enable, address, data in and data out. The memory generated will also be a positive edge triggered memory, meaning that data will be written to the specified address on the positive edge of the clock signal if write enable is set to high.When write enable is low, the data stored in the address specified is read out.

3.3 Task 3

Integrate the memory module into your design by instantiating it as a component. Once this is done, the next step will be test whether the integration has succeeded. Refer to Fig 4 and the following steps for some suggested ways to start working on the memory controller. The goal of the following would be to perform read and write operations to the memory using the basic pins and switches available on the board.

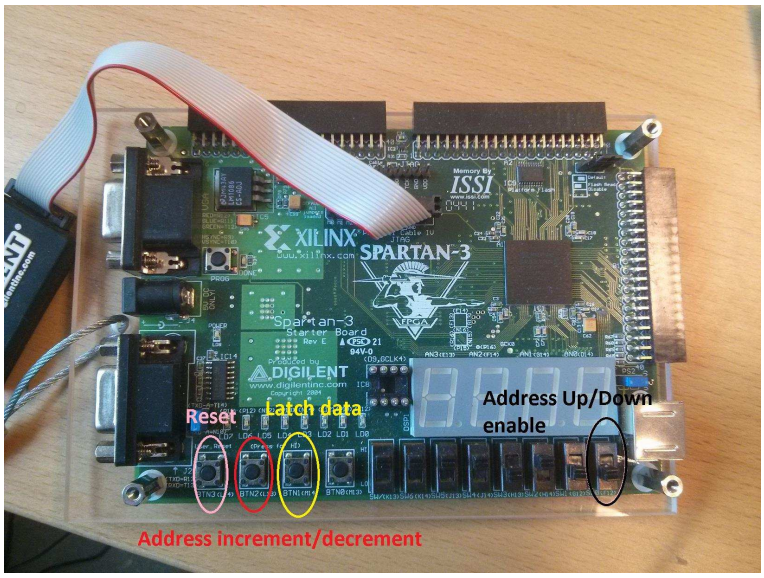


Figure 4: FPGA with memory controls

* Assign BTN[3] to reset your system.

* Even though the mem_data bus generated from the IP will be 8 bits, for testing we will now use only 4 bits. Assign the mem_data[3 downto 0] bits to your keyboard out data. Assign the upper 4 bits to zero.

* Design a counter and connect the mem_address to this counter. The idea is that when BTN[2] is pressed if SWITCH[0] is set to 0, the address should increment and decrement if BTN[2] is pressed when SWITCH[0] is set to 1. Remember to use debouncing logic on the BTN, if not the memory address will increment by more than one at each press of BTN[2]. It would be a good idea to also connect the mem_write_enable to this button.

* Try and use the LED0-7 present on the board for debugging. Check whether after adding debouncing logic the address increments by the required steps.

* Assign BTN[1] to enable data latching. Essentially the keyboard data should be registered to the memory input when BTN[1] is pressed.

* Connect the memory_out data to the seven segment display, either on the FPGA or on the LCD screen for debugging.

It is always a good idea to look at the warnings tab when synthesizing the design. Understand the warnings shown and see if they are OK for your design. It may happen

that the memory block is not connected properly and your system does not work. Clean the project files to obtain updated warnings on the next synthesis runs. This is done from the Project menu in the ISE Project manager.

3.4 Task 4

Once you have the above things working, now it would be time to write code to enable data to be stored in the memory along with the operators. Use the same BTN[2-1] logic designed above to store a string of data into the memory along with the operands. Make sure you are able to store 3 data digits into the memory. The input data range is from 0 to 255. At the end of entering data values along with operands, the ALU should be started. This can be done by pressing the <Enter> key. At the press of every <Enter> key the memory controller should be able to pop the top three memory locations(the two data operands and the operator), compute the result and display it on the VGA screen. On the next <Enter> key the next two data operands and the operator have to popped out from the memory and result should be displayed on the VGA screen. For the mod 3 operator we need to enter only one data and the operand. Remember the result could be either a positive or a negative number. Therefore it is required to display the sign of the result before the result as shown in the example in Fig. 5. Since the data RAM created will be 8 bits wide and we need to store some operands along with data, some of the bit patterns can be assigned for these operators(e.g. “+”, “-”, “=” “mod”). Choose the range of 130 to 135 for operators. This also means that input data in the range of 130 to 135 shall not be considered as operands. Think about storing the values in the RAM only when proper operands and operators have been entered, meaning there should be an option for the back space key. If a mistake is done while entering the operands, one could use the backspace key to delete the already entered numbers and start over. Detailed requirements for this project is stated as following:

- Both data operands must be represented in at least 3 digits (hundreds, tens, units), and the computation results must be represented in three digits (hundreds, tens, units) along with the sign. The operands, computation result along with the operator must be shown on the emulated 7-segments on a VGA monitor. For example if one has to compute the sum of 98 and 99 the VGA display should look like. Notice that there is a sign operator before the result. The inputs will be entered in the 3 digits format, meaning if one wants to use 9 as an operand, the input from the keyboard shall be 009. If the data entered is above the limit, then the number shall be stored as 255. For example if the user enters 1234 as the first input operator, the calculator shall store this number as 255 if the data latch button is pressed. Note that the backspace key could be used to fix the data before the data latch button is pressed if desired.

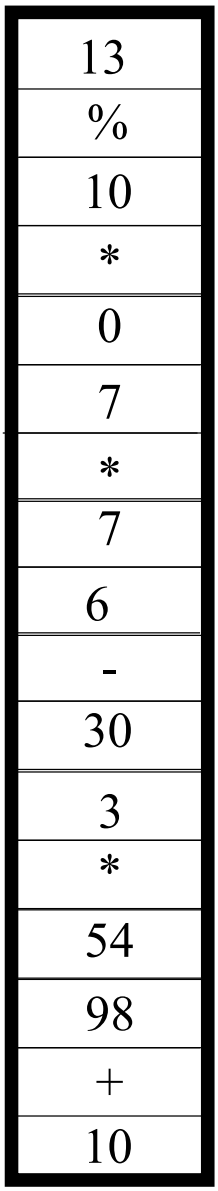
Figure 5: Example VGA Output

Remember that the result is signed and the operands are unsigned. This will enable one to design a simple state machine to accept the right amount of inputs before storing them in the memory.

- The design must be able to perform the following different computation operations: addition, subtraction, multiplication and modulo 3. An indication of overflow/underflow should also be displayed when it happens.

- The emulated 7-segments have to be shown in a visible size. It is allowed to load digits and operators from data ROMs, however, you have to consider the available memory capacity in the FPGA. It is recommended to design a display engine for one 7-segment, and use it to generate digits at all locations during system run-time. Using either logics or data memories is always a design trade off, where a common practice is to use a mixed design approach to find a balanced point between them. You may, for instance, store all data operators (e.g. “+”, “-”, “=”) in ROMs, and generate all digits by using one 7-segment display engine.

An example output for a list of operands is shown below. Assume that the memory is filled and looks as shown Fig 6



On first enter key
013%003 = 001
On second enter key
100*000 = 000
On third enter key
007*007 = 049
On fourth enter key
006-030 = -024
On fifth enter key
003*054 = 162
On sixth enter key
098+010 = 108

Figure 6: Example output

4 Course project 2 (*Grade 5*) - ALU with square root and Memory

In this project an additional operation will be added to the ALU. The ALU should be able to compute the square root of an unsigned number and display the result on the emulated seven segment display. The result obtained should be displayed with upto three digits in accuracy after the decimal point. The detailed requirements are as follows

- Interface the keyboard, ALU and the VGA as explained in the previous sections. Emulate a seven segment display on the VGA screen.

- The square root unit is to be designed which should accept an unsigned integer as its input and produce the square root of the number with at least three digits in accuracy after the decimal point. The input range for the square root number will be [0-255]. The square root unit has to be integrated into the ALU. The design should be capable of accepting data from keyboard, compute results for different operands like addition, multiplication and square root, then display the result on the VGA screen.

- Find an algorithm to calculate the square root like Newton-Rhaphson method and implement it in hardware. An introduction to algorithms implementing square root can be found on Wikipedia. Use a lookup table to find the closest square root and start with that as the seed to the algorithm. The following steps can be done to obtain a reasonably accurate square root of a number with Newton's method.

- Start by understanding the Matlab code provided in the appendix which uses the fixed point notation tool. Understand what widths of inputs and outputs are required to produce the desired accuracy in the final result. Think about number of bits needed by the divisor and choose correct widths appropriately. Below is an example of how one would compute the square root of 245 with an initial estimate of 16.

$$\begin{aligned} S &= 245 \\ x_0 &= 16 \\ x_1 &= 0.5 \times (x_0 + \frac{S}{x_0}) \\ &= 15.65625 \\ x_2 &= 0.5 \times (x_1 + \frac{S}{x_1}) \\ &= 15.65247 \end{aligned} \tag{1}$$

- As it can be seen from the above equations, you will need a divider unit to perform the square root algorithm. Instantiate a divider core generator IP from the IP generator tool. Select the algorithm to be of type "fixed" with dividend and divisor widths to be of **X** and **Y** bits respectively. The values of **X** and **Y** should be obtained from Matlab simulations or by calculations. Since you need 3 decimal digits of accuracy, this would correspond to a fraction binary width of 10 bits. Choose the number of clocks per division as 1. Generate the IP.

- Read the divider IPs manual to understand the number of cycles it takes to produce one division output. Construct a small testbench and verify that you understand the divider's operation.

- The next step is to design the above algorithm and integrate it into the system. From simulations done in Matlab you will be able to understand the number of iterations required to reach the desired accuracy for all the numbers in the range from [0-255]. Once this is fixed, design a state machine which will start when lets say BTN[0] is pressed and process the square root of the input number. For testing purposes, you can input the number from the SWITCH buttons. Remember to wait for the division operation to complete before proceeding to the next iteration. Construct a testbench and verify that the state machine is functioning as required. The divider IP returns the integer and fractional part of the quotient. As it can be seen in the above equation, x_1 is a fixed point number. Make sure you design your adders to take care that the fractional bits are added correctly and the integer bits are updated. Think about whether you need to shift up the divisor to make the fractional bits into integer bits, or whether it is ok to use the fractional divisor as it is.

- The final result obtained will contain an integer part and a fractional part. The integer part needs to be converted into BCD code. Use the old code that you must have written by now. The fractional part needs to be converted to be displayed as a BCD number also. Design a small function to do this.

- Integrate the square root unit to the ALU with memory designed in Project 1. The final result should be displayed on the LCD monitor using the VGA controller and the input should come from the keyboard instead of the SWITCH keys.

- Remember to do things step by step. Create modules based on functionality and integrate in the top level. For example, you should have a separate module which accepts a seven segment coded number and displays on the VGA, one separate module to perform square root, one to do all the other ALU operations etc.

5 Appendix

```
%%sqrt_vlsi project code
clear,clc
for k = 1:255
S = k;
%integer and fractional part bits
INT = 15;
FRA = 10;
run_len = 5;
x_col = zeros(1,run_len);

%Look up table
if(S < 32)
    x_col(1) = 5;
elseif(S<81)
    x_col(1) = 9;
else
    x_col(1) = 16;
end

for i = 2:run_len
    pre_calc = double(fi(x_col(i-1),0,INT,FRA));
    div_calc = double(fi((S*2^FRA/(pre_calc*2^FRA)),0,INT,FRA));
    x_col(i) = 0.5*(pre_calc+div_calc);
    x_col(i) = double(fi(x_col(i),0,INT,FRA));
end
actual_sqrt = sqrt(S)*ones(1,run_len);
err_iter(:,k) = (actual_sqrt-x_col);

end
surf(err_iter);
max(abs(err_iter(1,:)))
max(abs(err_iter(2,:)))
max(abs(err_iter(3,:)))
max(abs(err_iter(4,:)))
max(abs(err_iter(5,:)))
```