ECE 574: Modeling and synthesis of digital systems using Verilog and VHDL
Fall Semester 2019

Design and implementation (in Verilog) of a memory interface along with a Microblaze embedded processor
Report and Signoff due Week 10 (November 6)

This project involves the design and implementation of a digital system design using Verilog. It involves interfacing to external devices (ALU and VGA monitor) and includes a Microblaze embedded processor programmed in C. It shows how an existing IP written in VHDL (the VGA controller) can be integrated into the rest of the Verilog design and how software can be used to control custom logic. Part 1-2-3 and Part 4 are two different projects. You don't need to combine them into a single design.

Part a (background):

- Read the DS865 data sheet
- Complete the Microblaze MCS tutorial including the extra steps to add switches and leds.
    - Modify the size of the MicroBlaze memory from 16KB to 32KB
    - Make the UART BAUD 9600 so it will work easily with the lab PCs

Part 1: VGA Display and connection to the Microblaze

- Add a MMCM to create 100MHz for the Microblaze and 25MHz for the VGA controller (and other logic in your design)
- Instantiate the Digilent VGA Controller as a *VHDL module* (do not modify any of the code or convert to Verilog)
- Assume the screen is divided into blocks, with each block 32 pixels high by 32 pixels wide, and the top left corner block is at block x,y position (0,0) and the bottom right block is at x,y position (19, 14)
- Create a block and place it at a starting position determined by the value of one GPO port from the Microblaze (use some of the bits for the X position and some for the Y position).
- Use another GPO port to set the 12-bits of color for the block
- When the Microblaze updates the X, Y port values the colored block should move to the new location

Part 2: Hardware additions to MicroBlaze MCS

- Connect the X,Y block position to a single GPO port on the Microblaze
- Connect the color of the block (12-bits) to a GPO port
- Connect three of the slider switches to a GPI port to select the color of the block: Red, Green, Yellow, Blue, White.

Part 3: Software additions to MicroBlaze MCS - Modify the C program to:

- Display a message on the PC terminal window showing the date and your name
- Generate the X,Y starting position of the block (can be 0,0).

- Read the slider switches to determine the color of the block and then set the 12-bit GPO with the color information
    - You should be able to change the color of the block at any time
- Use four keys on the keyboard to move the block up, down, left, right by modifying the X and Y position
    - Use the inbyte() function to read the keyboard characters
    - When the keys are pressed move the block around the screen. Just move the block by one location for each key press. Stop the block from 'wrapping' around the screen.
- Display the x and y coordinates of the block position (0,0) to (19,14) in decimal on the PC terminal window

Part 4: Hardware Multiplier and Divider connected to MicroBlaze MCS

- Create a hardware multiplier and a hardware divider that will be controlled using the terminal and GPI/GPO ports of the Microblaze. The numbers will be entered through the terminal and the hardware will do the computation. Once the computations are completed, you will read and display the results on the terminal. Also, you should reset the hardware for new multiplication or division.

Multiplier:

- Create a hardware multiplier that multiplies 128-bit numbers. You are allowed to use only a single 32x32-bit multiplier. You have use state machine to calculate the multiplication in multiple clock cycles.
- Use terminal to enter two 128-bit numbers and use GPO ports of Microblaze to write the numbers to the registers of the Hardware Multiplier. Later, start he multiplication by setting a control register as '1'. Here, the control register can be used to start, stop and reset the computation.
- Once the multiplication is completed, read it using the GPI port.
  Hint:One way to read/write from GPO/GPI ports is that to use two of them. Since GPI/GPO ports are limited on bitsize, you may use one to hold the value and one to hold the index location. For example, say we can write 16 bits at a time, we have eight 16-bit locations {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}. We can set registers as {index, value} = {0, 5}, {1, 3}, {2, 9} in sequence. The 128-bit number would be {0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x03, 0x09}.

Unsinged Divider:

- Create a 64-bit division hardware. Simply you need to compute:

$$N/D = (Q, R)$$

  which Q is the quotient and R is the remainder for N divided by D.
  Create a state machine to handle the division steps in multiple clock cycles. Use a similar approach to load the numbers and compute the division as you did in multiplication, e.g. using a register for index setting and having a control register to start and reset the hardware. A pseudo code for division algorithm is given below:

Pseudo Code:

We will divide *N* by *D*, placing the quotient in *Q* and the remainder in *R*. In the following code, all values are treated as unsigned integers with n-bit size.

```
Q = 0
R = 0
for i = n-1 to 0:
        R = R << 1
        R(0) = N(i)       -- set bit location zero of R with i^th location of N

        if(R >= D)
                R = R – D
                Q(i) = 1
        endif
endfor
```

return(*Q*, *R***)**

Example:

If we take $N=1100_2$ (12_{10}) and $D=100_2$ (4_{10})
Step 1: Set R=0 and Q=0
Step 2: Take i=3 (one less than the number of bits in N)
Step 3: R=00 (left shifted by 1)
Step 4: R=01 (setting R(0) to N(i))
Step 5: R<D, so skip statement
Step 2: Set i=2
Step 3: R=010
Step 4: R=011
Step 5: R<D, statement skipped
Step 2: Set i=1
Step 3: R=0110
Step 4: R=0110
Step 5: R>=D, statement entered
Step 5b: R=10 (R−D)
Step 5c: Q=10 (setting Q(i) to 1)
Step 2: Set i=0
Step 3: R=100
Step 4: R=100
Step 5: R>=D, statement entered
Step 5b: R=0 (R−D)
Step 5c: Q=11 (setting Q(i) to 1)
end
$Q=11_2$ (3_{10}) and R=0.

- The quotient and remainders should be returned and displayed on terminal with a similar approach using GPI ports.

Combining Multiplier and Divider:

- Once the multiplier and divider hardware are completed combine them in a single project. You may need to use same GPO/GPI ports for inputs, so use the control registers to select between the operands and the operation.

As usual, this is not a complete description – make whatever additions you think are necessary.

Prepare a sign-off sheet and demo your system (parts1to 4) along with your Verilog source files, C program, and write your report before the deadline.

Grading Guidelines

- [50 pts] Implementation
    - [40 pts] Design works on board and meets requirements
- [20 pts] Source Code – Verilog and C program in Appendix
    - Code style and comments (well-commented and tab-indented code!)
    - Use of *case* vs. *if*, spaghetti code vs. structured, etc.
    - Recognizable implementation of "standard" elements (state machines, counters, clock dividers, decoders)
    - Sensible use of modularity
    - No latches or other synthesis problems
- [30 pts] Lab Report
    - [5 pts] Brief Introduction / Problem Statement
    - [15 pts] General Overview of approach to solution and description (include Block and State Diagrams with descriptions). Include pictures of VGA display showing the block and PC window showing text. For multiplier and divider, include pictures of some example computations PC window.
    - [5 pts] FPGA Resource usage (# flip-flops with explanation) and listing and explanation of warning messages (don't copy all the Xilinx reports – just the relevant sections)
    - [5 pts] Conclusions
        - Problems faced in implementation
        - Solutions used to solve problems
        - Lessons learned from the project
        - Suggestions for further improvements and extensions
- [10 pts] Extra points
    - Possible extra points for good additional features or capabilities (need to demo on board and include description in report)