**Lab 7 – VHDL for Digital System Design**

*Pay special attention to red and read each and every step thoroughly to avoid stall in your work due to missing out something trivial*

**Author**: Poulami Das

**Owner**: Dr. Nina Telang

**Summary** The goal of this lab is to familiarize you with the notion of designing a control state machine; it draws heavily on the divider example in Section 18.3.

**Problem Assignment**

Your assignment for Lab 7 is given in the table below, with small modifications which are specified here

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1st Letter of Last Name** | **A-F** | **G-L** | **M-R** | **S-Z** |
| **Assigned Problem** | **20.A** | **20.B** | **20.C** | **20.D** |

The 4 problems we have assigned are division problems, and relatively easy.  If you are up for a challenge, you can try out a multiply-add problem. Specifically, you will **get 10% bonus points** in addition to the early submission bonus if:

* Your assignment is 20.A and you work out 20.T instead
* Your assignment is 20.B and you work out 20.U instead
* Your assignment is 20.C and you work out 20.V instead
* Your assignment is 20.D and you work out 20.W instead

Bonus clarifications:

* Your assigned problem is 20.A and you work out 20.A and turn it in, you will get 150
* If your assigned problem is 20.A and you work out 20.T instead, and turn it in, you will get 150 + 15 = 165

***We have made changes, specified below, to the book problems. This is done to counter the use of old solutions.  If you turn in a solution that solves the unmodified problem, and you cannot explain how you came up with the solution, we will treat your solution as copied and immediately refer the matter to***[***Student Judicial Services***](http://deanofstudents.utexas.edu/sjs/downloads/disc_ref.pdf)***. (As per the SJS website, an “F” in the course is a common penalty for plagiarism on a major paper.)***

**Modified problems**

The changes specified for each design problem are as follows

**20.A:** same as in the book except divide a 7 bit dividend by a 3 bit divisor to give a 4 bit quotient and 3bit remainder. Overflow occurs when quotient requires more than 4 bits.

**20.B:** same as 20A except divide a 6 bit dividend by a 3 bit divisor to give a 3 bit quotient and 3 bit remainder. Overflow occurs when quotient requires more than 3 bits.

**20.C:** same as 20A except divide a 8 bit dividend by a 4 bit divisor to give a 4 bit quotient. Overflow occurs when quotient requires more than 4 bits.

**20.D:** same as 20A except divide 8 bit dividend by a 3 bit divisor to to give a 5 bit quotient . Overflow occurs when quotient requires more than 5 bits.

**20.T:** same as textbook except X is 5 bits,Y is 3 bits and Z is 3 bits. Use 9 bit accumulator.

**20.U:** same as 20T except X is 4 bits, y is 4 bits and Z is 3  bits. Use 9 bit accumulator.

**20.V:** same as 20T except X is 3 bits, y is 5 bits and Z is 5 bits. Use 9 bit accumulator.

**20.W:** same as 20T except X is 4bit, y is 3 bits and Z is 3 bits. Use 8 bit accumulator.

**Please note the changes in the problems.**

**Detailed Instructions**

0) **Read and understand** the section 18.2 of your text book. The corresponding VHDL code for Figure 18-7, Page 599 is in Figure 20-2, Page 652.

1) **Read** your assigned problem from the text book.

2) Draw a block diagram of the system showing registers, adders, MUXes, and other components. Define the necessary signals. specify the size of registers, adders, etc. (**Deliverable #1**). Provide an active-high asynchronous *RESET* for your design.

3) Draw a state graph for the CONTROL CIRCUIT. (**Deliverable #2**)

4) Based on the results of steps 1 and 2, write a VHDL description of the system using control signals and TWO PROCESSES as in figure 20-7 ( page 657 of text book).

* First process     : generates the next-state an control signals
* Second process : updates the states and other registers (This is your clock dependent process)

5) Compile your code. Those who want to use Modelsim, you already know what to do from previous labs.

6) Refer to Test Data in Appendix A at the end of this manual. Remember, the check out test data will be different. Make sure you test your circuit exhaustively for wide range of input combinations.

7) Simulate your code using the Test sequences you obtained in Step 6.

* Note : Clock period is 200ns and input changes 25ns after the clock
* Remark : If you submit a waveform and cannot interpret the output, your TA may deduce points.

8) Go over the waveform and make sure it corresponds to the assigned problem.

9) Submit your VHDL code (**Deliverable #3**) and a snapshot of the waveform (**Deliverable #4**)

10) Open Vivado software and load your code on to a new project.

11) Synthesize your code in Vivado.

12) Examine the synthesize report.

13) If you see any latch under HDL Synthesis Report, you must modify your code to eliminate the latch

14) If no latch was identified, then print the HDL Synthesis Report page (Only 1 page, **Deliverable #5**)

16) Implement your code and load it on the board. Use the tutorial for programming the board. Follow the port mapping given in the tables on the next page to write the XDC file (**Deliverable #6**). This time only guidelines are given and you are expected to write the file. So you start with a blank XDC file, completely commented out and you add your port mappings. **If you cannot understand what to do, reopen your Lab 3 and see what you did.**

**You will need a clock for the design. To use the internal clock on the board, use the following section on the Basys XDC file. The example clock port name is “Clk”. You have to use the name that you use for your clock. Do not map your clock signal to any switch as it will cause routing issues and your circuit implementation will not be correct.**

*## Clock signal*

*set\_property PACKAGE\_PIN W5 [get\_ports Clk]*

*set\_property IOSTANDARD LVCMOS33 [get\_ports Clk]*

*create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports Clk]*

***Remember the board clock is too fast (100MHz frequency clock). In case anything fails debugging in this clock domain is painful. So if you need to debug a failing design on the board, you may use a slower clock just like it was done for Lab 5 and 6. You can use the clock divider from there and instantiate your design along with the clock divider (just as was provided to you for your Lab 5 and 6) to create a top level wrapper. DO NOT FORGET TO MAP THE SLOW CLOCK TO ONE OF THE AVAILABLE LEDs SO THAT YOU CAN SUPPORT THE DEBUG. ALSO ACCORDINGLY MODIFY THE XDC FILE BECAUSE NOW YOUR TOP LEVEL IS THE WRAPPER AND NOT THE DESIGN.***

**Rest of the port mapping to be used is given below:**

**For Problem 20.A**

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| Dividend [6:0] | Switches 6 to 0 with switch 6 mapped to MSB and switch 0 mapped to LSB |
| Divisor [2:0] | Switches 10 to 8 with switch 10 mapped to MSB and switch 8 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Quotient [3:0] | LEDs 3 to 0 with LED 3 mapped to MSB and LED 0 mapped to LSB |
| Remainder [2:0] | LEDs 7 to 5 with LED 7 mapped to MSB and LED 5 mapped to LSB |
| Done | LED 15 |
| Overflow | LED 14 |

**For Problem 20.B**

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| Dividend [5:0] | Switches 5 to 0 with switch 5 mapped to MSB and switch 0 mapped to LSB |
| Divisor [2:0] | Switches 9 to 7 with switch 9 mapped to MSB and switch 7 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Quotient [2:0] | LEDs 2 to 0 with LED 2 mapped to MSB and LED 0 mapped to LSB |
| Remainder [2:0] | LEDs 6 to 4 with LED 6 mapped to MSB and LED 4 mapped to LSB |
| Done | LED 15 |
| Overflow | LED 14 |

**For Problem 20.C**

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| Dividend [7:0] | Switches 7 to 0 with switch 7 mapped to MSB and switch 0 mapped to LSB |
| Divisor [3:0] | Switches 12 to 9 with switch 12 mapped to MSB and switch 9 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Quotient [3:0] | LEDs 3 to 0 with LED 3 mapped to MSB and LED 0 mapped to LSB |
| Remainder [3:0] | LEDs 8 to 5 with LED 8 mapped to MSB and LED 5 mapped to LSB |
| Done | LED 15 |
| Overflow | LED 14 |

**For Problem 20.D**

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| Dividend [7:0] | Switches 7 to 0 with switch 7 mapped to MSB and switch 0 mapped to LSB |
| Divisor [2:0] | Switches 11 to 9 with switch 11 mapped to MSB and switch 9 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Quotient [5:0] | LEDs 5 to 0 with LED 5 mapped to MSB and LED 0 mapped to LSB |
| Remainder [2:0] | LEDs 9 to 7 with LED 9 mapped to MSB and LED 7 mapped to LSB |
| Done | LED 15 |
| Overflow | LED 14 |

***For Problem 20.T***

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| X [4:0] | Switches 4 to 0 with switch 4 mapped to MSB and switch 0 mapped to LSB |
| Y [2:0] | Switches 8 to 6 with switch 8 mapped to MSB and switch 6 mapped to LSB |
| Z [2:0] | Switches 12 to 10 with switch 12 mapped to MSB and switch 10 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Result [7:0] | LEDs 7 to 0 with LED 7 mapped to MSB and LED 0 mapped to LSB |
| Done | LED 15 |

***For Problem 20.U***

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| X [3:0] | Switches 3 to 0 with switch 3 mapped to MSB and switch 0 mapped to LSB |
| Y [3:0] | Switches 8 to 5 with switch 8 mapped to MSB and switch 5 mapped to LSB |
| Z [2:0] | Switches 12 to 10 with switch 12 mapped to MSB and switch 10 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Result [7:0] | LEDs 7 to 0 with LED 7 mapped to MSB and LED 0 mapped to LSB |
| Done | LED 15 |

***For Problem 20.V***

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| X [2:0] | Switches 2 to 0 with switch 2 mapped to MSB and switch 0 mapped to LSB |
| Y [4:0] | Switches 8 to 4 with switch 8 mapped to MSB and switch 4 mapped to LSB |
| Z [4:0] | Switches 13 to 9 with switch 13 mapped to MSB and switch 9 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Result [7:0] | LEDs 7 to 0 with LED 7 mapped to MSB and LED 0 mapped to LSB |
| Done | LED 15 |

***For Problem 20.W***

|  |  |
| --- | --- |
| **Ports** | **Component on the FPGA Board** |
| X [3:0] | Switches 3 to 0 with switch 3 mapped to MSB and switch 0 mapped to LSB |
| Y [2:0] | Switches 7 to 5 with switch 7 mapped to MSB and switch 5 mapped to LSB |
| Z [2:0] | Switches 11 to 9 with switch 11 mapped to MSB and switch 9 mapped to LSB |
| Clock | Internal board clock |
| Reset | Switch 15 |
| Start | Switch 14 |
| Result [6:0] | LEDs 6 to 0 with LED 6 mapped to MSB and LED 0 mapped to LSB |
| Done | LED 15 |

17) Print your CoverSheet.

18) Turn in your work

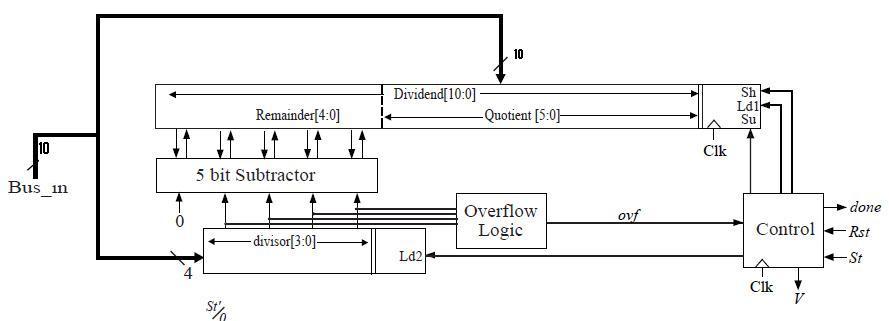
19) Demo your work to a TA

Good luck!

**Demo Example**

Design a divider for unsigned binary numbers that divides a 10-bit dividend by a 4-bit divisor to give a 6-bit quotient.

**Block Diagram:**



**entity** divider **is**

**port**(dividend: in std\_logic\_vector(7 downto 0);

divisor: in std\_logic\_vector(4 downto 0)

St, Clk, Reset: **in** std\_logic;

           Done,v:  **out** std\_logic;

           Quotient: **out** std\_logic\_vector (5 downto 0);

           Remainder: **out** std\_logic\_vector(4 downto 0));

**end** divider;

**architecture** behav **of** divider **is**

-- Define your signals in here !

-- Define your signals in here !

-- Define your signals in here !

-- Define your signals in here !

**begin**

**process**(St, \_\_\_ , \_\_\_ )

**begin**

    --Initialize your signals here

**case** State **is**

**when** 0 =>

**---** Remaining part of the code

**end case**;

**end process**;

**process**( \_\_ , \_\_ )

**begin**

**if** Rst = '1' **then** State <= 0;

**elsif** Clk'event **and** Clk = '1' **then**

State <= NextState;

**----------------------------**

**end if;**

**end process**;

**end** behav;

**Input text file(test.txt)**

force clk 0 0,1 100 -repeat 200

force Rst 0 0,1 50, 0 75

force dividend <>

force divisor

force st 0 0, 1 50

**Extra Requirements**

For problems 20.A, 20.B, 20.C, and 20.D, you will need to add a “**done**” state to your state graph.  The inputs except the clock will come from the switches on the FPGA board.

**FAQ**

**Q:** What should I study for this Lab?

**A:** Read Ch. 18 and Ch.20 of your text book.

**Q:** What is my assignment?

**A:** Refer to the table at the beginning of this presentation.

**Q:** What the maximum extra credit on this lab?

**A:**signal Bonus: integer range 0 to 20;

    if ((ur\_assigned\_problem == 20.A) and (u\_work\_out == 20.T))

    then Bonus=20;

**Q. When simulating my behavioral code in DirectVHDL, I see intermittent X's for my signal outputs. Also, the code does not synthesize in Xilinx Vivado?**

You probably have not followed the conventional VHDL programming methodology. An intermittent 'X' during simulation may be due to multiple drivers. Your code has multiple drivers if you are trying to assign value to a VHDL signal in two or more different processes and/or concurrent statements. Now, if the assigned values to the same VHDL signal are different, then a wire contention occurs and the simulation reports the value as X. Such multiple drivers will frequently be reported as synthesis errors in Xilinx Vivado.

In the example below, signal “a” is being assigned at 3 places, which execute concurrently. The output of a would be 'X' in simulation.

architecture SM\_a of SM is

    signal a: std\_logic;

begin

**a <= '0';**

    process(in\_a , in\_b)

    begin

**a <= in\_a or in\_b;**

    end process;

    process(in\_c, in\_d)

    begin

**a <= in\_c and in\_d;**

    end process;

end SM\_a;

Appendix A (Test Data)

You need to use the following test inputs for each assigned problem.

20.A.

|  |  |  |
| --- | --- | --- |
| Test | Dividend | Divisor |
| 1 | 1100111 | 111 |
| 2 | 1010000 | 110 |
| 3 | 1111000 | 101 |

20.B.

|  |  |  |
| --- | --- | --- |
| Test | Dividend | Divisor |
| 1 | 100111 | 110 |
| 2 | 011010 | 100 |
| 3 | 110100 | 011 |

20.C.

|  |  |  |
| --- | --- | --- |
| Test | Dividend | Divisor |
| 1 | 10011011 | 1101 |
| 2 | 01011010 | 1000 |
| 3 | 11100110 | 1010 |

20.D.

|  |  |  |
| --- | --- | --- |
| Test | Dividend | Divisor |
| 1 | 01010011 | 110 |
| 2 | 10110101 | 110 |
| 3 | 11011011 | 100 |

20.T.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | X | Y | Z |
| 1 | 11001 | 101 | 100 |
| 2 | 01101 | 100 | 110 |
| 3 | 00011 | 111 | 010 |

20.U.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | X | Y | Z |
| 1 | 1101 | 1010 | 110 |
| 2 | 1001 | 0111 | 010 |
| 3 | 1100 | 1110 | 011 |

20.V.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | X | Y | Z |
| 1 | 111 | 11100 | 01111 |
| 2 | 010 | 10010 | 11101 |
| 3 | 100 | 10101 | 10001 |

20.W.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | X | Y | Z |
| 1 | 1110 | 011 | 101 |
| 2 | 1001 | 111 | 100 |
| 3 | 1101 | 110 | 011 |

**Please note that this is representative data only. During a demo, the TA may ask you to show for a different input combination and if your circuit fails to work, you lose credit. It is therefore advisable that you test your circuit enough and not just these test input combinations**