ELE306 FALL2004 LAB2

Hierarchical VHDL Designs

Instead of using the block diagram/schematic drawing tool, we will use VHDL to create macro(s) to be used in a hierarchical design. Note that this method of design will be used for the rest of the labs. To assist with the writing of the VHDL file, syntax highlighting and templates will be utilized.

Exercise #1 (Creating a 7-Segment display entity in VHDL)

- Start the Quartus II software and create a project named lab2 in your user directory as was done in Lab 1 using "*File -> New Project Wizard"*.
- Once the project is created, click "*File -> New"*. Choose the "Device Design Files" tab, select "*VHDL File"*, and click *OK*.
- If you like to see larger font, go to "*Tools -> Options…*" and select Fonts for Text Editor (near the end of the list on the left).
- There are three basic sections that the VHDL example being created: the include, the entity declaration, and the architecture declaration section. In the text editor, type the following:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
```

- The preceding text encompasses the include section of which specifies what functionality to include and from which libraries are they declared.
- Next, the entity declaration section will be constructed. Type the following after the include section.

```
ENTITY sevenseg IS
        PORT(       ain : IN std_logic_vector(3 DOWNTO 0);
                    aout : OUT std_logic_vector(6 DOWNTO 0)
                    );
END sevenseg;
```

  - The keywords (shown in capital here and colored blue in the editor) are case insensitive. Also note the second declaration does not end with a semicolon; the line just before the right parenthesis does not need a semicolon.

- The entity declaration section defines the ports on the module that you are creating. Ports such as ain and aout in this design are a vector or an array of bits declared with the DOWNTO keyword. When you define the limits of the ports, be sure to include the value 0. In this case, ain is a 4 bit input and aout is a 7 bit output. Since VHDL is a description language, you can describe inputs and outputs as busses which are basically an array of the corresponding type. An example of a one bit output is "cout: OUT STD_LOGIC;". To declare a 4-bit output bus, a

vector is declared in the following manner: “mybus: OUT STD_LOGIC_VECTOR (3 DOWNTO 0);”.

- Now, the final section will be constructed – the architecture declaration section. At the end of the VHDL file, insert the following text.

```
ARCHITECTURE decoder OF sevenseg IS
BEGIN
                    --abcdefg
        aout <=     "0000001" WHEN ain="0000" ELSE --0
                    "1001111" WHEN ain="0001" ELSE --1
                    "0010010" WHEN ain="0010" ELSE --2
                    "0000110" WHEN ain="0011" ELSE --3
                    "1001100" WHEN ain="0100" ELSE --4
                    "0100100" WHEN ain="0101" ELSE --5
                    "0100000" WHEN ain="0110" ELSE --6
                    "0001111" WHEN ain="0111" ELSE --7
                    "0000000" WHEN ain="1000" ELSE --8
                    "0001100" WHEN ain="1001" ELSE --9
                    "0001000" WHEN ain="1010" ELSE --A
                    "1100000" WHEN ain="1011" ELSE --B
                    "0110001" WHEN ain="1100" ELSE --C
                    "1000010" WHEN ain="1101" ELSE --D
                    "0110000" WHEN ain="1110" ELSE --E
                    "0111000" WHEN ain="1111" ELSE --F
                    "1010101"; -- default
END ARCHITECTURE decoder;
```

  - “--“ denotes the beginning of comments. The comments will be displayed in green in the text editor. The multi-digit numbers must be placed inside double quotation marks; whereas single digit numbers are in single quotation marks. The numbers are shown in magenta color.

- The architecture section defines the functionality of the design. In this case, a strict definition of the output is defined based upon the input provided. An assignment of a value to an output is done with the “<=” characters. The WHEN-ELSE statements specify the output to set when a particular condition is met. In the case of the 7-segment decoder, the value of *ain* will be between 0 to 15. The last value specified after the last ELSE keyword is the default value used. (Remember: with std_logic, four inputs should have 9^4=6561 possible combinations not just 16!) The output values of *aout* were determined by looking at the pin of the 7segment display that should be turned on then the pins need to be assigned to each element. Be careful, the board uses active low logic.
- Save the file by “*File -> Save”*. IMPORTANT!!! When asked for a filename, specify “sevenseg.vhd”. The file name has to be consistent with your entity name!
- Click “*Project->Set as Top-Level Entity”*. This will make the current entity the project’s top-level design entity. This step is necessary because the entity name is different from the project name. (You will notice the top-level entity name on the left panel change from lab2 to sevenseg)

Functional Simulation

- Click "*Processing->Generate Functional Simulation Netlist"*. This will enable the VHDL codes be processed so all information for a functional simulation are generated. Note that, the VHDL is yet to be translated into hardware in this step.
- Click "*Assignments -> Settings"*. In the Category section, click on *Simulator Settings* category and select *Functional* for "Simulation mode".
- Click *File->New* and choose the Other Files tab. Select the Vector Waveform File and press OK.
- Since the hardware is yet to be created, you will have to type in the node names: ain and aout. Double click on the "Name" field and a "Insert Node or Bus" window appear. Type in ain for name, select "INPUT" as the type and change "Bus width" to 4. Perform the same steps for aout except for "OUTPUT" and 7-bit bus.
- Setup a simulation to test all the possible states of the input for the module in a vector waveform file, as was done in lab 1 for the simulation. An easier way is to right click on ain and select "Value → Count Value…" Go to "Timing" and select count every 50ns. This will give you all combinations plus a few more. The values are displayed in Hexadecimal.
- Click *Processing->Start Simulation* after saving the waveform.
- Verify the correct output is obtained according to the VHDL code. For the bus node a "+" sign button and be click to see individual nets. Note that the outputs exhibit no delay time at all, because the simulation is performed only at functional level with no real hardware yet (and thus no real hardware delay time).

Compilation

- If there are any discrepancies, fix the problems with the VHDL code and re-simulate. Once the design appears correct in functional simulation, compile the design by clicking "*Processing->Start Compilation*".
- From the Compilation Reports under the "Timing Analyzer" section, determine how fast the circuit can run (Timing Analyzer Summary).
- Assign the following pin names to test the design by typing them in under *Assignments->Assignment Editor* for the correct pin value. Click the Pin button (second button from the left) in the upper right hand corner of the window. The first row of the assignment table should be "To". The rest is the same as in Lab 1.

Pin Name	**Pin Value**
ain[0]	38
ain[1]	39
ain[2]	40
ain[3]	41
aout[0]	24
aout[1]	23
aout[2]	21
aout[3]	20
aout[4]	19
aout[5]	18

aout[6]	17

- Run compilation again to incorporate the pin assignment into the design.

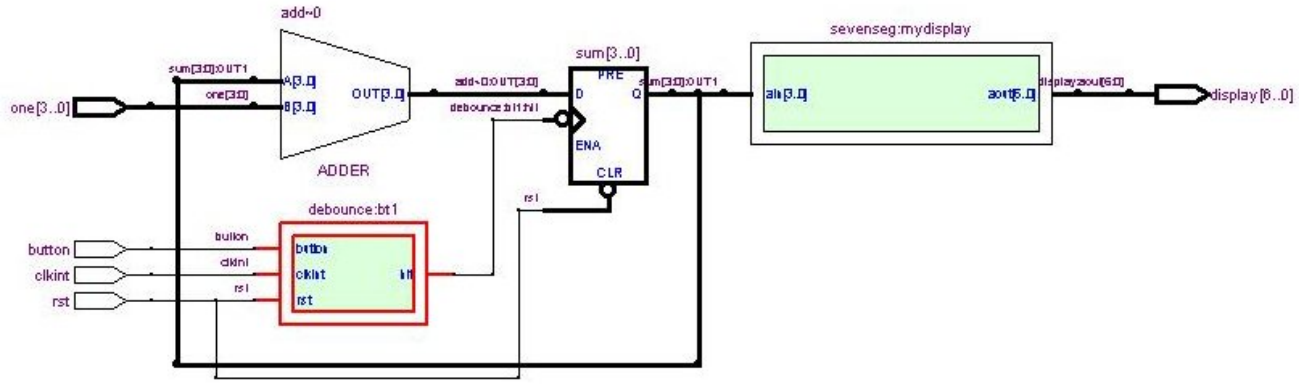
Timing Simulation

- Now we will do a timing simulation. Click "*Assignments -> Settings*" and change *Simulation Mode* to *Timing*.
- Click *Processing->Start Simulation*. Verify the correct output is still given.
- You will notice the effects of actual hardware delay time. After the inputs have changed the output will start changing, giving wrong answers, and finally settle on the correct answer. This dramatically shows that not all outputs are generated with the same amount of delay time.

Programming

- Program the device as was done in Step 5 from Lab1.

Exercise #2 (Creating a hierarchical design: a 4-bit adder)



In this exercise, from a hierarchical design point of view, a new top-level module will be created that uses the sevenseg module from exercise #1 as a component. The RTL (register-transfer level) schematic above shows the entire design to be unveil next. The new module being created will be called lab2. The design will provide the addition of two 4-bit numbers and display the result. The way this design utilizes sevenseg is by converting the sum of a 4-bit addition to a value that can be displayed on a 7-segment display.

- Start by creating a new VHDL file.
- Type the following in the editor.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY lab2 IS
        PORT( clkint: IN std_logic;
              button: IN std_logic;
              rst: IN std_logic;
              one: IN std_logic_vector(3 DOWNTO 0);
```

```
                display: OUT std_logic_vector(6 DOWNTO 0)
                );
END lab2;

ARCHITECTURE myadder OF lab2 IS
        signal clk: std_logic;
        signal sum: std_logic_vector(3 downto 0);

        --define the seven segment decoder component
        COMPONENT sevenseg
                PORT( ain: in std_logic_vector(3 downto 0);
                        aout: out std_logic_vector(6 downto 0)
                        );
        END COMPONENT;

        --define the debounce circuit for the pushbutton
        COMPONENT debounce
                PORT( clkint: IN std_logic;
                        rst: IN std_logic;
                        button: IN std_logic;
                        hit: OUT std_logic
                        );
        END COMPONENT;

BEGIN
        --Set the port map for our seven segment display
        mydisplay : sevenseg PORT MAP (ain => sum, aout => display);
        button1 : debounce PORT MAP (clkint, rst, button, clk);

        PROCESS(clk, rst)
        BEGIN
                IF(rst='0') THEN
                        sum <= "0000";
                ELSIF(clk'event AND clk='0') THEN
                        sum <= one + sum;
                END IF;
        END PROCESS;

END ARCHITECTURE myadder;
```

- The preceding design contains a few new elements. First off, the line “USE IEEE.std_logic_unsigned.ALL;” is included to allow for the addition of two values. The next new element is the use of a signal. A signal is like a variable that is only visible within the current architecture and can be assigned a value and read from. A type is assigned to each signal; and this design uses a signal called sum to hold the value of the addition of the two input vectors.

- Directly following the signal declaration, a COMPONENT is defined. As you might guess, this defines which external modules you will be using. In this module, we use the 7segment module developed in the previous exercise to display the output of the addition performed. The COMPONENT is declared very similar to the entity section of any module by defining the inputs and outputs used.
- After defining the COMPONENT, an instance of the module is created by the statement "mydisplay : sevenseg PORT MAP (ain => sum, aout => display);". The statement defines first the name of the instance *mydisplay* (the module name is optional), and then the ports that each of the inputs and outputs map. The same is done for our debounce circuit which allows for the debounced push-button required for one-pulse signals when the button is pushed. In the case of debounce circuit, we use *positional association* meaning that you mention only the variable names on the lab2 side and they will be map to the variables on the debounce side according to the sequence in which you put them. The method used in mapping sevenseg is known as *named association*.
- The next new section is the PROCESS declaration. A PROCESS creates a sequential block of statements to be executed. The parameters specified after PROCESS, clk and rst, are the inputs that trigger the following block of code held within the BEGIN and END PROCESS statements to be executed.
- Click *File->Save* and specify lab2.vhd as the filename.
- Click *Project->Set as Top-Level Entity*. This will make the current entity the project's top-level design entity. Normally, the top-level entity's name is default to the project name, which in this case is lab2.
- Since we defined the component debounce, we need to define the debounce circuit. Create a new VHDL file called debounce.vhd and place the following text into that file.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY debounce IS
        PORT( clkint: IN std_logic;
                        rst: IN std_logic;
                        button: IN std_logic;
                        hit: OUT std_logic
                );
END debounce;

ARCHITECTURE mydebounce OF debounce IS
        SIGNAL count: integer range 0 to 250000;
        SIGNAL buttonclk: std_logic;
        SIGNAL state: std_logic_vector(3 downto 0);
BEGIN
        PROCESS(clkint, rst, count)
        BEGIN
                IF(rst='0') THEN
                        count<=0;
```

```
                buttonclk <= '0';
            ELSIF(count=250000) THEN
                count<=0;
            ELSIF(clkint'event AND clkint='1') THEN
                if(count=0) THEN
                    buttonclk <= not buttonclk;
                END IF;
                count<=count+1;
            END IF;
        END PROCESS;

        PROCESS(button, buttonclk, rst)
        BEGIN
            IF(rst='0') THEN
                state<="1111";
            ELSIF(buttonclk'event AND buttonclk='1') THEN
                state(3 downto 1) <= state(2 downto 0);
                state(0) <= button;
                IF( state = "0000") THEN
                    hit <= '1';
                ELSE
                    hit <= '0';
                END IF;
            END IF;
        END PROCESS;
    END mydebounce;
```

- The debounce circuit takes in a 25MHz clock and steps it down to a 100Hz clock in the first PROCESS statement. The second PROCESS statement takes this slower clock and uses it to check over a sampling of 4 clock cycles if the input is maintained and thus eliminating a false reaction to the bouncy switch contact.

Functional Simulation

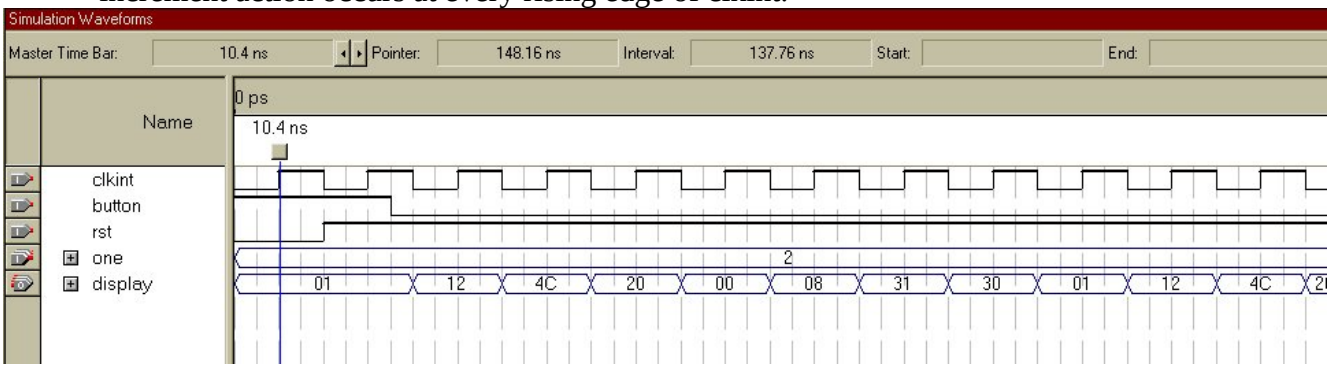
- The “debounce” circuit is necessary for the design to work. However, we should ignore it for now and treat it as a “black box.”

- Commented out the component instantiation of debounce and reconnect the wire as described below:

```
-- button1 : debounce PORT MAP (clkint, rst, button, clk);
clk <= clkint and not button;
```

  The above two line effectively remove the debounce circuit and re-wired the lab2. Now you may perform the functional simulation with ease.

- Since there is no circuit delay to worry about, you may set “clkint” as fast as you can. The simulation waveform should looks like the one shown below. The circuit lab2 is an accumulator with “one” as its input. For the simulation, “one” input is set to 2 and therefore, the accumulator’s output goes from 0→2→4→6→8→A→C→E→0. In reality, the increment should occur after each button is push. However, since we hardwired the button with clkint, the increment action occurs at every rising edge of clkint.



- As was done in the previous exercise, add the following pin assignments for the design through the Assignment Editor. You will have to delete or change the original assignments made in the previous exercise since they are no longer used.

Pin Name	**Pin Value**
one[0]	38
one[1]	39
one[2]	40
one[3]	41
display[0]	24
display[1]	23
display[2]	21
display[3]	20
display[4]	19
display[5]	18
display[6]	17
button	28
rst	29
clkint	91

- Complete the compilation, timing simulation, and programming procedures for this design as was done in the previous exercise to verify and realize the circuit.
- **HINT#1**: You do not need to simulate debounce circuit individually. Instead you should perform functional simulation (and subsequent timing simulation) on lab2.
- **HINT#2**: If you try to assign pin numbers before performing compilation, you will have to type in the node names as they will not be available. Also, remember, you must compile AFTER pin assignment.

Assignments

- Always create new directory for the assignment so not to override your exercise files above. Use the "Project → Archive Project…" and "Project → Restore Archived Project…" to transport your project between lab partners. The archive has an extension ".qar". DO NOT tamper with the project directory in any way and NEVER use ZIP or any system level archival program (such as TAR, RAR, ACE, etc.) to transport your project.

- **Assignment #1**: Design an accumulator that takes in a value of 0 to 15 (e.g. 4-bit binary) and holds a sum up to 31 (e.g. 5-bit binary). The carry output will be seen as the first and only bit of the second hexadecimal digit. Thus, the sum ranges from 00 to 1F. Use both seven segment displays (obviously) on the development board for this.

- **Assignment #2**: Same as #1 except display the sum in ***decimal***.

- **Assignment #3**: Modify the design so you can control whether the accumulator ***adds or subtracts*** the value set on the dip switches. Sum may be displayed in hexadecimal or decimal.

Lab 2 Report:

- **The exercise parts:** function and timing simulation waveforms, respectively from both exercises.
- **The assignment part:** VHDL code listings and function and timing simulation waveforms for all assignments. Duplicated parts need not be presented.
- **Remember to demonstrate your implementations to TA or the instructor.**