EDA322 Digital Design Lab

LAB6

Designed by: Anurag Negi, AngelosArelakis, Ioannis Sourdis

List of requirements for lab6:

1. Have completed lab5.

In this lab you will synthesize your code using Xilinx ISE and rectify any problems related to synthesis. Then you will generate a *bit file* which will be used to program a Spartan 6 FPGA. The file will be downloaded onto the Nexys 3 board over a USB link using a software tool called Digilent Adept.

Before you start, please be reminded that this Lab, as well as Lab 7, are optional. Completing both of these Labs will award you 10% in the final exam. Completing only one of them will have no effect.

You will use several supplementary VHDL files in this lab. These files will be available for download on Ping Pong or on a USB storage device (ask the instructors). Your module will be instantiated in the module called proc_top.

Below is the list of VHDL modules that will be provided to you in this lab:

- proc_top.vhd : top level module for synthesis
- pin.ucf: defines interconnections between top level ports and resources on the Nexys3 board
- pulse_on_edge.vhd: generates a 1clock-wide pulse on detecting an edge on the input. It is used to generate master_load_enable pulses based on switch toggles
- async_reset_deassert_sync.vhd: synchronizes deassertion of asynchronous reset to the clock domain
- synch_1bit.vhd: synchronizes a 1bit signal to the processor clock domain. This is used to synchronize inputs from on-board components like switches.
- debugmux.vhd: used to implement functionality that selects signals to be displayed on the seven segment displays
- sseg_driver.vhd: drives the seven segment display units with the required refresh rate
- sseg_decode.vhd: converts hexadecimal digits to seven segment display enables

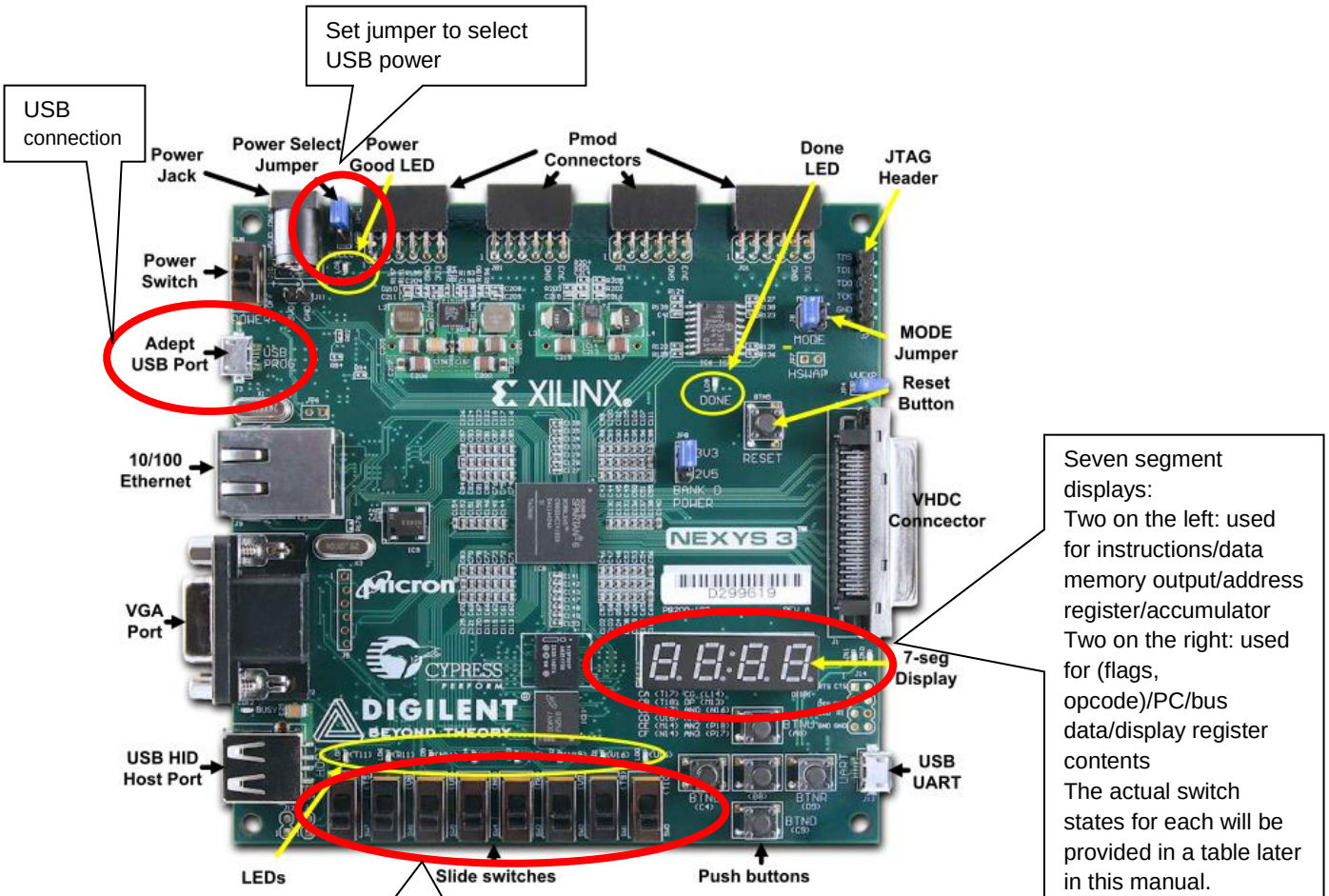
Make sure that your top level design entity description looks exactly like the one shown below:

```
entity EDA322_processor is
    Port (   externalIn : in  STD_LOGIC_VECTOR (7 downto 0); -- "extIn" in Figure 1
             CLK : in STD_LOGIC;
             master_load_enable: in STD_LOGIC;
             ARESETN : in STD_LOGIC;
       pc2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- PC
       instr2seg : out  STD_LOGIC_VECTOR (11 downto 0); -- Instruction register
       Addr2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Address register
       dMemOut2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Data memory output
       aluOut2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- ALU output
       acc2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Accumulator
       flag2seg : out  STD_LOGIC_VECTOR (3 downto 0); -- Flags
       busOut2seg : out  STD_LOGIC_VECTOR (7 downto 0); -- Value on the bus
             disp2seg: out STD_LOGIC_VECTOR(7 downto 0); --Display register
             errSig2seg : out STD_LOGIC; -- Bus Error signal
             ovf : out STD_LOGIC; -- Overflow
             zero : out STD_LOGIC); -- Zero
end EDA322_processor;
```

Ensure that all the necessary signals for observation (in this lab and for later download on the board) have been brought out to the top level module. The comments in the code snippet above indicate the values which need to be brought out as output ports in the top level module.

This manual will now show how to use resources available on the board to control your design and observe values at pertinent top-level ports.

The image below shows how the Nexys 3 board looks like. Please take special note of components highlighted using red circles, as these will be used to control the FPGA once you have programmed the board with your design.



The VHDL modules provided to you for use in this lab contain code which allows you to display values on certain top level ports onto one of two pairs of seven segment displays available on the board. The choice is made using switches SW2 through SW5. The table below shows how the state of these switches affects outputs on the seven segment displays.

Switch state (SW3, SW2)	**Display (left pair)**
00	Instruction (7 downto 0)
01	Data memory output
10	Contents of address register
11	Accumulator contents
Switch state (SW5, SW4)	**Display (right pair)**
00	Flags, Opcode [i.e. instruction(11 downto 8)]
01	PC
10	Bus data
11	Contents of the display register

Those who are interested in investigating how these connections are setupcan take a look at the UCF (User constraints file) file that has been provided for use during the mapping stage in the design flow. It specifies how top level ports (in this lab we will use the file called proc_top.vhd as the top level entity) connect to various resources available on the board. Look at the Nexys3 reference manual to identify pin names to be used in the UCF file. The UCF file contains several statements, each specifying how a top-level input/output port is connected to a pin available on the board.

Task Description

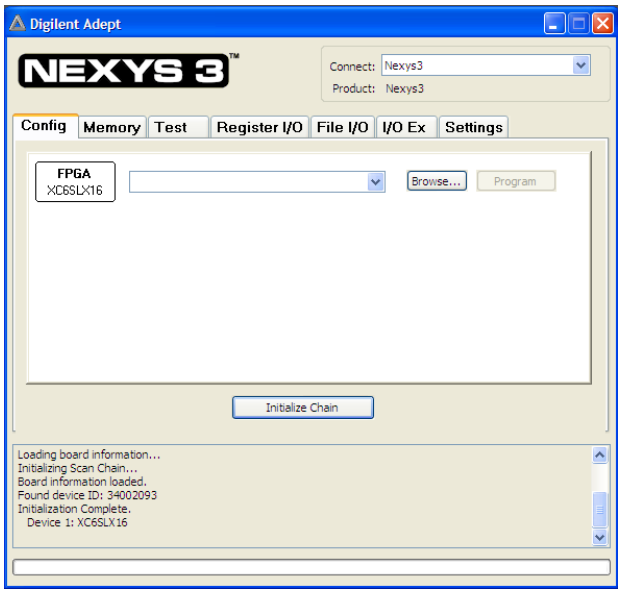
You must complete the following tasks:

Design Synthesis–Task 1

Add the files provided to you to a Xilinx ISE project containing your code for the processor. Choose proc_top as the top level module and synthesize your design. Rectify any problems during synthesis. Demonstrate correct synthesis to the instructor.

FPGA programming and operation–Task 2

After successful synthesis double click on “Generate Programming File” to create a bit file for programming the FPGA device. Remember that the tool may report errors which need to fixed before a bit-file can be successfully generated. Plug in the Nexys3 board into the USB port of your computer. Switch it on (make sure that the power jumper is set to USB). Next, start the Digilent Adept tool. You should see a window similar to the one shown below:



If everything has been plugged in correctly, the tool will automatically detect and report the existence of Nexys 3 board with the appropriate FPGA device name. Click on “Browse…”. This opens up a dialogue box that lets you browse and select a bit file. Select “proc_top.bit” and then click “Program”. This should now program the FPGA device. Once programming is complete you can test your design by providing reset/clock toggles using the switches. Use switches 2 through 5 to observe values of various signals in the design. Once you are convinced that the device works correctly, please demonstrate a successful run of the Fibonacci sequence generator to the instructor. This code is automatically loaded into the instruction memory when the FPGA is programmed. (Note that the code may rely upon data loaded into the data memory at program time. This can change during the course of execution and is not reset when using the reset switch. Thus, runs after reset may produce spurious values. So, reprogram and run to check for correctness.)

Lab report:

In your final lab report, include a section about Lab 6. In particular, try to focus on the following points:

- Mention what kind of problems you encountered during synthesis and implementation and explain how you solved them. How is synthesis/implementation different from simulation?
- Describe how you verified the correctness of your FPGA implementation. Note that the code that is executed on the implementation is the same code used for testing in Lab 5. You should compare sequences of values on various signals observed on the seven-segment displays to values seen in Modelsim simulation of the design. Explain which

signals you chose to observe and which values you expected to see. Please include in the report the sequence of program counter (PC) and display register values you observed during a successful execution on the FPGA.

Hints and Tips:

There are several issues that might cause a design don't work on the FPGA board although simulation results are correct. We list here some of the things you can check to find the source of the malfunction.

1- *Master_load_enable* is an extra signal which is used in this lab to control the processor and make it possible to observe the outputs on the board one clock at a time. Itis very important that you incorporate *master_load_enable* with every **state transition** and well as **outputs of controller that go to the registers.** A simple *if clause* or *and gates* can be used for this matter.

2- Make sure that your design is latch free. The tool will issue a **warning**whenever a latch is detected. Latches will cause timing problems when running a design on an actual board.

3- If the LEDs on the board are dim, it might be because of missing UFC or pin files. Include them in your project in ISE.

4- Try to perform firm movements when flipping switches on the board. Flipping a switch slowly and having it "float" between the two states for some time might result in "bouncing" – that means the perceived value of the switch changing more than once during the flip. Furthermore, these "extra" value changes might last very short time and, as such, cause your circuit to exercise random behavior. Some boards are more sensitive than others to bouncing, so if your implemented design behaves strangely, you can try with a different board before looking for other problems.

5- Always remember that there is no way to reinitialize the processor memories without reprogramming the FPGA. Thus, between different runs of the program, make sure to reprogram the device, since values in the data memory might have changed.