CG3207 COMPUTER ARCHITECTURE

LAB 1: FAMILIARIZATION WITH HDL/FPGA AND ASSEMBLY LANGUAGE

WEEK 4,

SLIDES COURTESY: GU JING &
ANIMESH GUPTA

CG3207 Computer Architecture

WEEK	CG3207		
	DESCRIPTION	MARKS	REMARKS
4	Lab 1 (Familiarization with HDL/FPGA and Assembly Language)	—	Individual exercise
5	Lab 1 (Demo) + Lab 2 (Basic CPU Design)	10	
6	Lab 2 (Basic CPU Design)	—	Team of 2 or 3 students
Recess	No Lab	—	—
7	Lab 2 (Demo) + Lab 3 (ALU Design)	30	Team of 2 or 3 students
8	Lab 3 (ALU Design)	—	
9	Lab 3 (Demo) + Lab 4 (Advanced CPU Design)	20 + 5**	
10	Lab 4 (Advanced CPU Design)	—	
11	Lab 4 (Advanced CPU Design)	—	
12	Lab 4 (Demo)	15 + 10**	
	Totally 4 Labs :	90	= 45% of the module marks
	Quizzes+Examination :		= 55% of the module marks

*** Additional marks for implementing enhancements.***

^^ Self-study at home due to Public Holiday.

Module Guideline

* Objective: Building an ARM-like processor using FPGA
* Hardware: Digilent Nexys 4 board based on Xilinx Artix 7 FPGA
* Software: download from https://wiki.nus.edu.sg/display/CG3207/Downloads
  - Xilinx Vivado 2019.1 WebPACK (for FPGA Programming, other versions also ok)
  - Keil MDK 4 (for Assembly Programming, or newer versions with the legacy pack)
* Resources: **Wiki.nus** > CG3207 Computer Architecture
  - Main reading material to understand all labs
  - Check Wiki comments for common doubts and queries

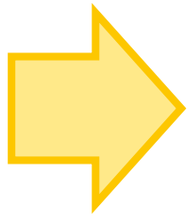
Module Policies

* **Demonstration**
  - Demonstrate on the stipulated date during your scheduled time slot
  - Intra-team mark difference may happen due to contribution difference
  - Heavy discount for late submission without valid reason

* **Plagiarism**
  - Discussions are encouraged, but it may not be a valid excuse if programs are similar
  - Warning from the NUS Code of Student Conduct: http://www.nus.edu.sg/registrar/adminpolicy/acceptance.html
  - Your teammates might be better off with no contribution at all from you than to receive plagiarized code

Workflow of CG3207 Lab 2 to 4

Assembly Program



ARMv3 Processor

* Design your assembly program accordingly, so that you could “prove” that the processor you designed can process certain assembly instructions.
* Useful Outputs: LEDs / 7-Segments of Nexys 4
* Useful Inputs: Pushbuttons, Switches of Nexys 4

Lab1	Understand the decoded assembly instructions (Lecture 3) Display the decoded hexadecimal representations (EE2026)
Lab2	Basic ARMv3 processor which supports: * LDR, STR with +ve immediate offset * AND, OR, ADD, SUB where Src2 is register or immediate without shifts * Branch B
	Improvements: * CMP and CMN * LDR, STR to support -ve immediate offset * DP instructions with Src2 is immediate shifts (LSL, LSR, ASR, ROR)
Lab3	* Incorporate division into MCycle unit (for both signed and unsigned) * Incorporate MCycle unit into your ARMv3 processor to support MUL and DIV
	Improvement(s) for signed multiplier
Lab4	Expand your processor to support all 16 DP instructions
	Improvements can be done in following areas: * Additional instructions * Implement pipelining with or without hazard hardware * Implement some kind of exception handling / support for interrupts

Refer to Wiki.nus for details

Lab 1 Objectives

* Refresh your memory of ARM programming, HDL simulation and FPGA implementation.
* Study the given sample assembly program to understand how instruction and data memory work.
* Get familiar with corresponding 32-bit/hex representation of instructions and be able to interpret them.
* *Not working on processor design yet*

Lab 1 Workflow

Sample Assembly Program

* *Understand and complete the assembly program to meet the specifications in Task(1)*
* *Note: read and understand each line of the example assembly program*

KEIL MDK 4

** ARM simulator that tests assembly program without hardware*

.hex file

** Contains the decoded 32-bit instructions and data*

Hex2ROM

** Instruction and data initialization*

Template Porject: Lab_1_Verilog

* *Paste the clipboard to corresponding .v file and complete the project to achieve the following:*
  - *Display 32-bit instructions & data on 7-Segments and LEDs (16 bits each time) consecutively with speed 1instruction/data per second.*
  - *When BTNU is pressed, the display rate should increase to ~4 times.*
  - *When BTNC is pressed, the display should pause.*

Task 1: Software simulation of an ARMv3/RISC-V based system

More Details on wiki > Lab1 > Tasks > 1) Software

Objective:

1. *Download the sample assembly program from wiki > Downloads page.*
2. *Understand the assembly program line by line.*

wiki > ARM Programming > Programming Instructions

wiki > ARM Programming > ARM Memory MAP

3. *Create a new project in Keil MDK 4 and apply correct options to your project.*

wiki > ARM Programming > Creating a Project and Basic Settings

4. *Build the project.*
5. *Start a debugging session.*

wiki > ARM Programming > Debugging Instructions

- *Play with Memory Map to check if DIPs value goes to LEDs value.*
- *Play around with DELAY_VALUE*
- *Understand HEX representation of assembly instructions.*

6. *Check if .hex file has been generated.*
7. *Use Hex2ROM to create .v or .vhd file for Task 2.*

wiki > ARM Programming > Converting .hex to ROM Initialization Code

Task 2: Display the Instruction (INSTR_MEM) and Data (DATA_CONST_MEM) ROM contents on the physical LEDs (LEDs on the FPGA board)

More Details on wiki > Lab1 > Tasks > 2) Hardware

Objective:

1. *Dump the Binary representation of your INSTR and DATA ROMs on LEDs of FPGA.*

Note: We are not executing the same thing as we simulated in Task 1. This will be done in further labs.

2. *As each location contains 32 bits though we have only 16 LEDs, display them in consecutive clock cycles, with the most significant half-word first.*

3. *The rate of the display should be approximately (doesn't need to be exact) 1 instruction/data per second.*

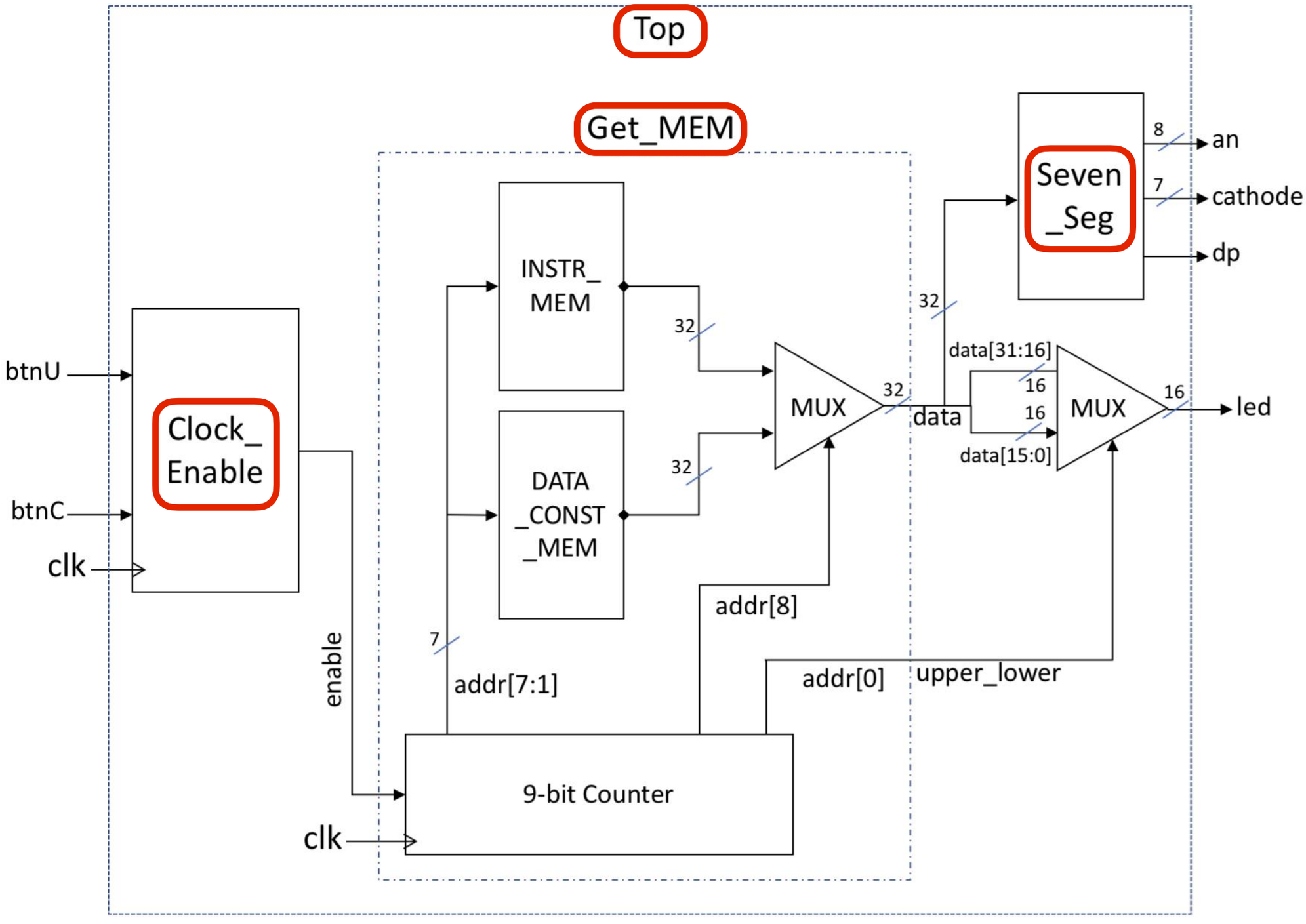
4. *When the instruction ROM display has been completed, display the contents of the data ROM. Do this in a cyclical manner (infinite loop).*

5. *When the pushbutton BTNU is pressed, the display rate should increase to approximately 4 instructions per second.*

6. *When the pushbutton BTNC is pressed, the display should pause.*

Simulate the HDL code and implement it on FPGA board.

Lab 1 Template



Task 2: Display the Instruction (INSTR_MEM) and Data (DATA_CONST_MEM) ROM contents on the physical LEDs (LEDs on the FPGA board)

More Details on wiki > Lab1 > Tasks > 2) Hardware

Hints:

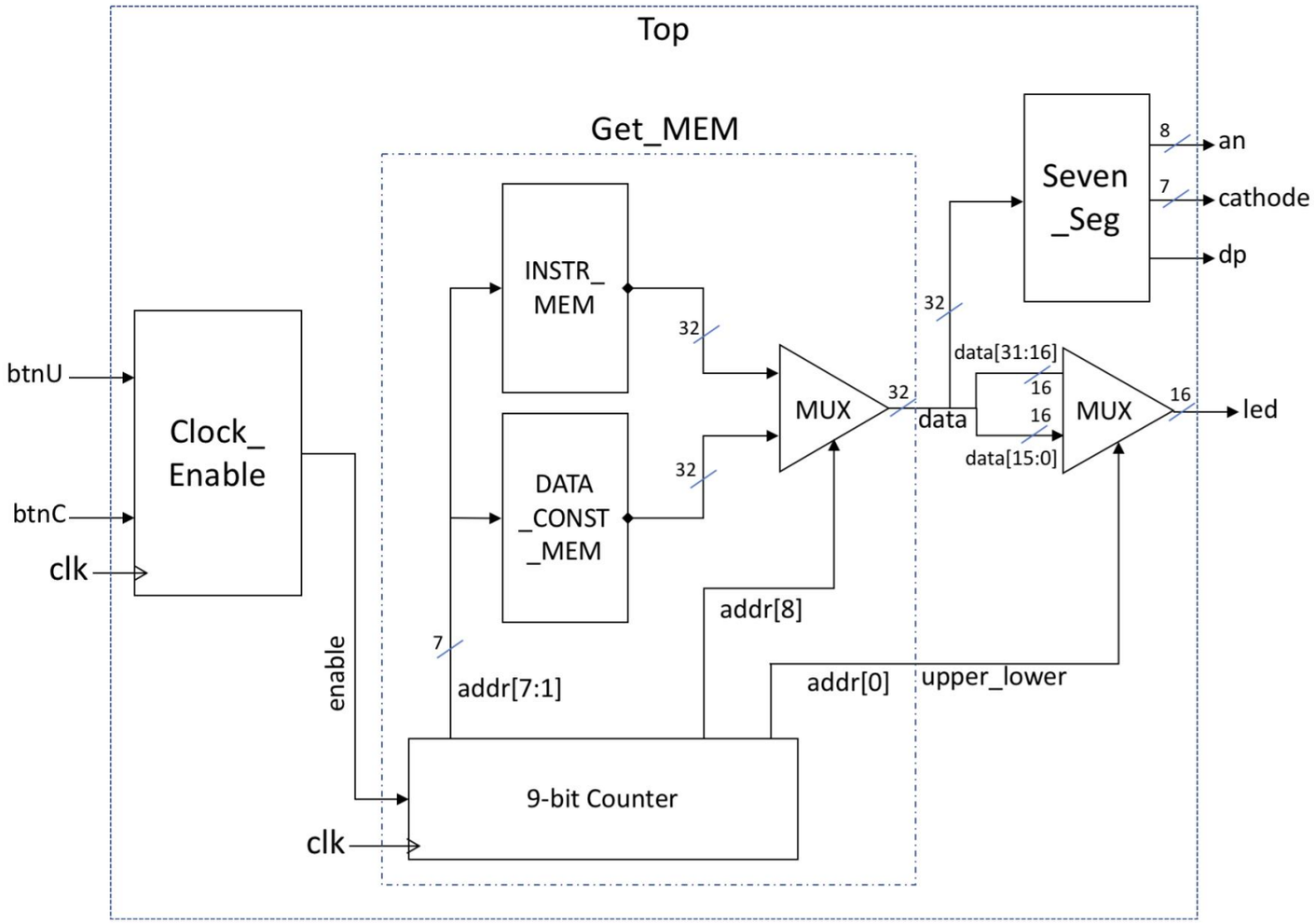
1. *Download Templates for Verilog and VHDL for all the boards from wiki > Downloads.*
   Its okay to not use the template files. You can write on your own as well.

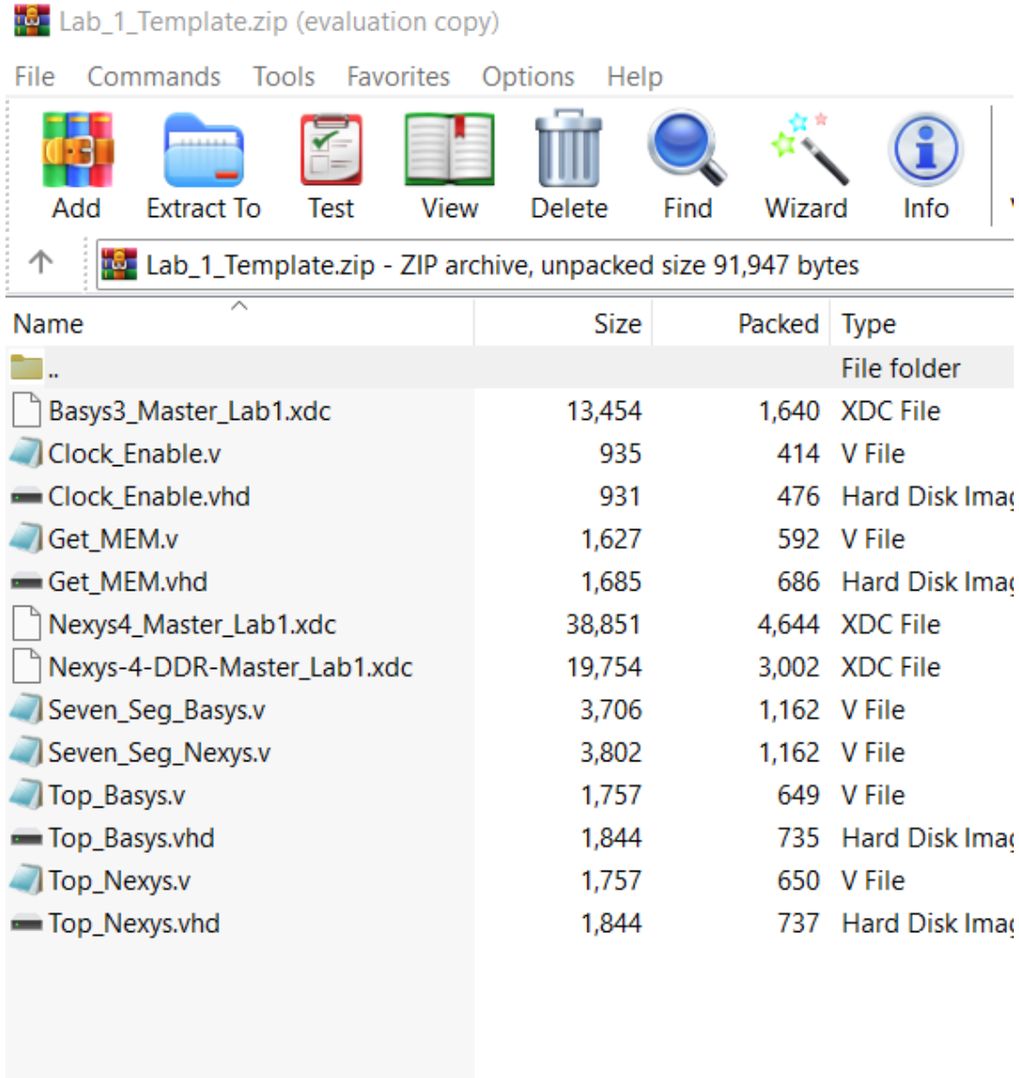
2. *Read notes carefully on wiki > Lab1 > Tasks > 2) Hardware > Notes.*

3. *Choose correct Part number while creating the new project in FPGA.*

Board	Part number	Settings
Nexys 4 / Nexys 4 DDR	XC7A100T-1CSG324C Note : Nexys 4 / Nexys 4 DDR both use the same FPGA chip, but pin mappings and hence the .xdc file are **different**. If you use the wrong .xdc file, you get **no warnings at all**, (as Vivado cares only about the chip you are using, not the board) but the hardware will be non-functional!	Reset All Filters Category: General Purpose Package: csg324 Temperature: All Remaining Family: Artix-7 Speed: -1 Static power: All Remaining Search: Part | I/O Pin Count | Available IOBs | LUT Elements | FlipFlops | Block RAMs | Ultra RAMs | DSPs xc7a15tcsg324-1 | 324 | 210 | 10400 | 20800 | 25 | 0 | 45 xc7a35tcsg324-1 | 324 | 210 | 20800 | 41600 | 50 | 0 | 90 xc7a50tcsg324-1 | 324 | 210 | 32600 | 65200 | 75 | 0 | 120 xc7a75tcsg324-1 | 324 | 210 | 47200 | 94400 | 105 | 0 | 180 xc7a100tcsg324-1 | 324 | 210 | 63400 | 126800 | 135 | 0 | 240

Task 2: Display the Instruction (INSTR_MEM) and Data (DATA_CONST_MEM) ROM contents on the physical LEDs (LEDs on the FPGA board)





Task 2: Display the Instruction (INSTR_MEM) and Data (DATA_CONST_MEM) ROM contents on the physical LEDs (LEDs on the FPGA board)

```
module_name    instance_name (
                    .port_name ( local_name),
                    … do for all ports
);
```

name of the module you want to instantiate

user defined name.
can be anything,
DO NOT start with a number, like 1inst

```
module_name    instance_name (
                .port_name ( local_signal_name),
                … do for all ports
);
```

name of wire/reg in parent module that you want to connect to the specified port

one of the names from input/output list of module that you want to instantiate

child module

```
module Clock_Enable(
      input clk,              // fundamental clock 1MHz
      input btnU,             // button BTNU for 4Hz speed
      input btnC,             // button BTNC for pause
      output reg enable);     // output signal used to enable the reading of next memory data
```

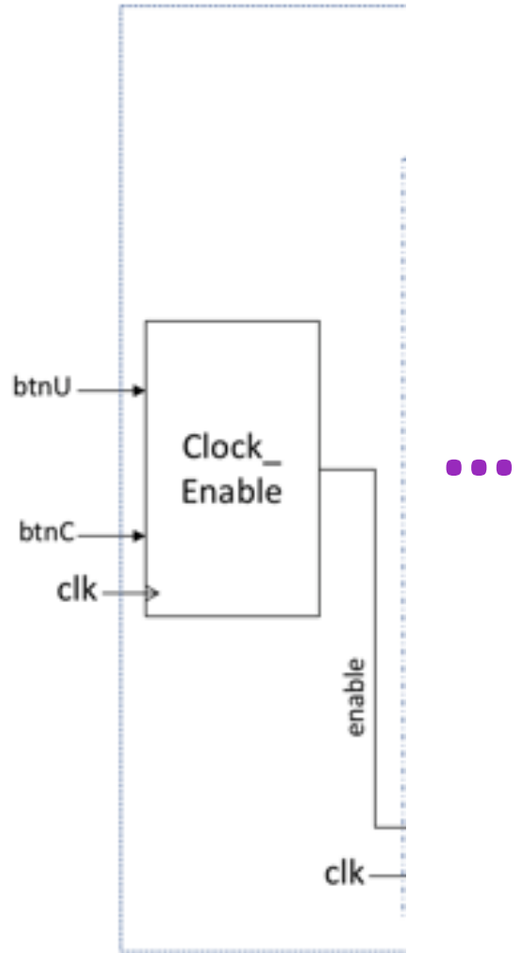
parent module

```
module Top(
    input clk,                // fundamental clock 1MHz
    input btnU,               // button BTNU for 4Hz speed
    input btnC,               // button BTNC for pause
    output [15:0] led,        // 16 LEDs to display upper or lower 16 bits of memory data
    output dp,                // dot point of 7-segments, can be deleted if 7-segments are not
    output [7:0]anode,        // anodes of 7-segments, can be deleted if 7-segments are not im
    output [6:0]cathode       // cathodes of 7-segments, can be deleted if 7-segments are not
    );

wire enable_out;              // enable signal to read the next memory content
wire upper_lower;       // 1-bit signal used between modules to indicate either upper or lower
                                  // upper_lower = 1 to display upper half of the Memory dat
wire [31:0] data;   // entire 32-bit contents displaying on LEDs and 7-segments, can be delete
```

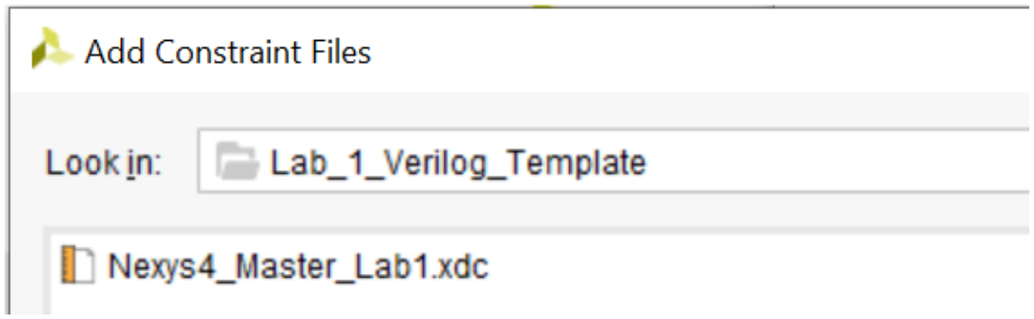
instantiation

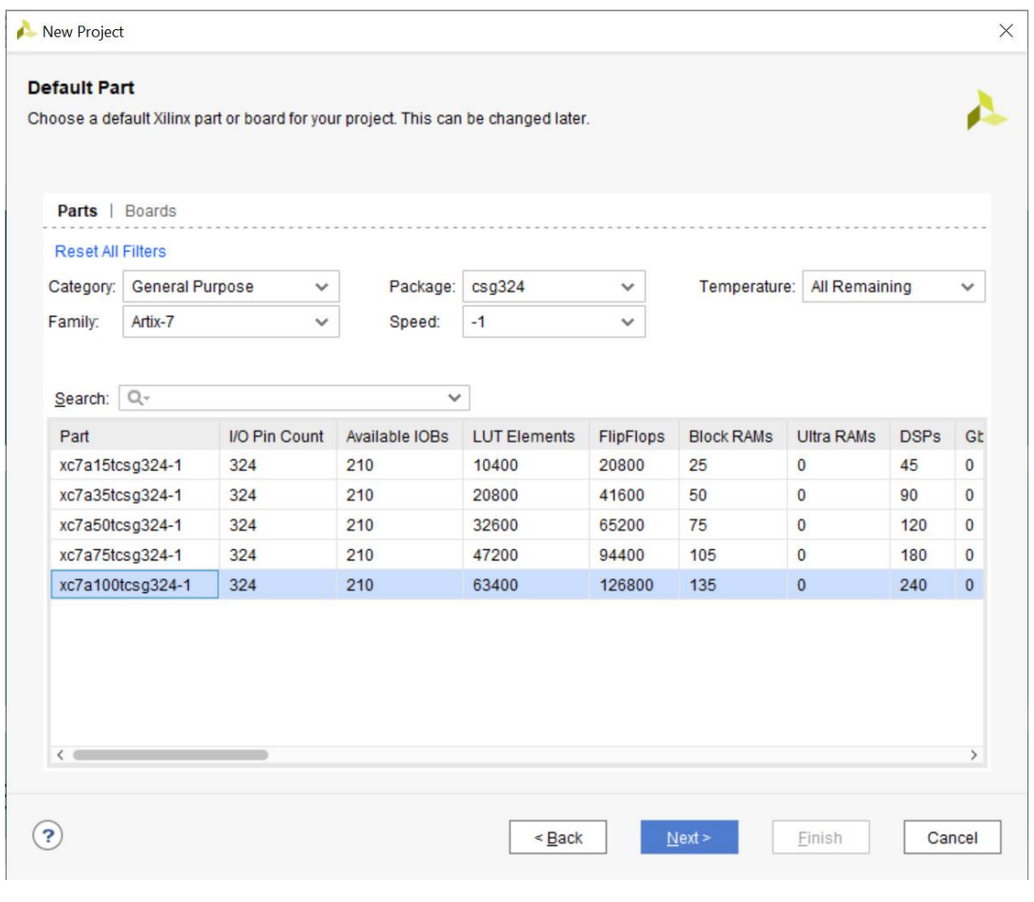
```
// Choose 1hz or 4hz display frequency based on BTNU and BTNC readings, using given module Clo
Clock_Enable my_clock_enable (
            .clk(clk),
            .btnU(btnU),
            .btnC(btnC),
            .enable(enable_out)
);
```



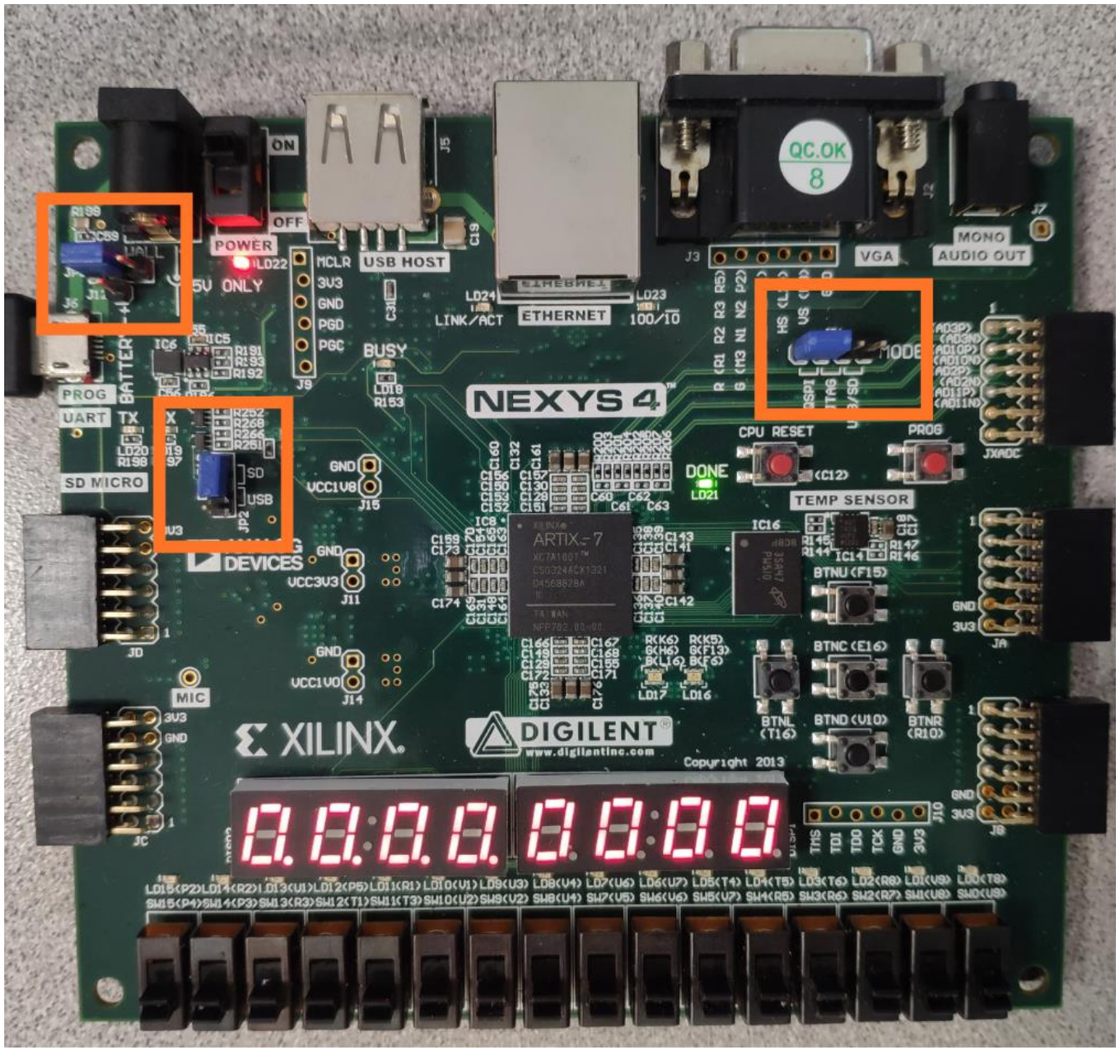
Lab 1 Hardware and Demonstration (Nexys4)

Constraint file and board selection for Nexys4:





Pin Configuration:



THE END