LAB 4: LEDCPU with ROM & RAM

This simple CPU is one that has a simple mechanism that we will describe below.

The instructions are 16 bits. These are kept in a ROM containing up to 256 instructions. The CPU has 2 possible instructions:

- Output
- Jump

These operations are divided within the instructions as follows:

BIT #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Output	OUTPUT PATTERN								DELAY ≠ 0							
Jump	JUMP ADDRESS								DELAY = 0							

Table I. CPU Instruction Set

So, we have the following logic:

1) DELAY = 0 => Do Jump operation

   BIT[15:8] is the address we jump to.

2) DELAY ≠0 => Do Output operation

   BIT[15:8] is the pattern shown in the LEDs.

   BIT[7:0] is the amount of time this particular pattern should be shown in the LEDs.

   Then it moves to the next address.

The CPU simply starts from address 0 and executes the following instructions.

Example

Let's say we wish to make the rotating dot example with this CPU and consider the original version where the dot rotated from left to right.

Let's divide the rotating dot to LED outputs, so we should see:

BIT #	7	6	5	4	3	2	1	0
STAGE0	1	0	0	0	0	0	0	0
STAGE1	0	1	0	0	0	0	0	0
STAGE2	0	0	1	0	0	0	0	0
STAGE3	0	0	0	1	0	0	0	0
STAGE4	0	0	0	0	1	0	0	0
STAGE5	0	0	0	0	0	1	0	0
STAGE6	0	0	0	0	0	0	1	0
STAGE7	0	0	0	0	0	0	0	1

Table II. LED Output for Rotating Dot

So the correspoding LedCPU's ROM, i.e. its memory, should be:

BIT # / Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	1	1	0	0	0	0
1	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	0	0	1	1	0	0	0	0
2	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	0	0	1	1	0	0	0	0
3	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	0	0	1	1	0	0	0	0
4	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	0	0	1	1	0	0	0	0
5	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	0	0	1	1	0	0	0	0
6	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	0	0	1	1	0	0	0	0
7	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	0	0	1	1	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

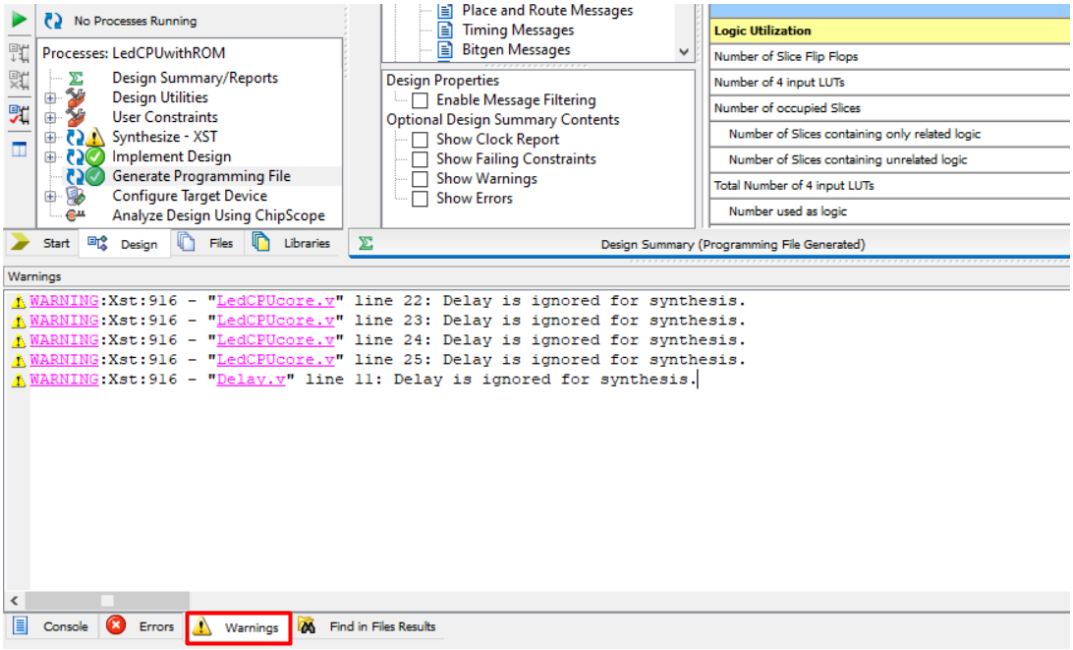
Table III. Complete Memory Contents for Rotating Dot

So, between addresses 0 and 7, we give new LED patterns that are in the instruction's BIT[15:8] and ask the delay to be 8'b00110000 in BIT[7:0]. At the last address (address 8), the CPU sees that the delay part (BIT[7:0]) is 0, and then jumps back to the instruction at the address 0 since BIT[15:8] is 0.

ASSIGNMENTS

Part 1:

1. Download lab4 files.
2. Create new project in Xilinx ISE Design Suite, name it ledCPUWithROM and add the design files using “add copy of source”. The files are LedCPUWithROM.v, LedCPUWithROM.ucf, Delay.v, ROM.v and LedCPUcore.v.
3. Open LedCPUCore.v file and modify as required.
4. Do “Generate Programming File”, you should only see the following warnings in the Warnings window.



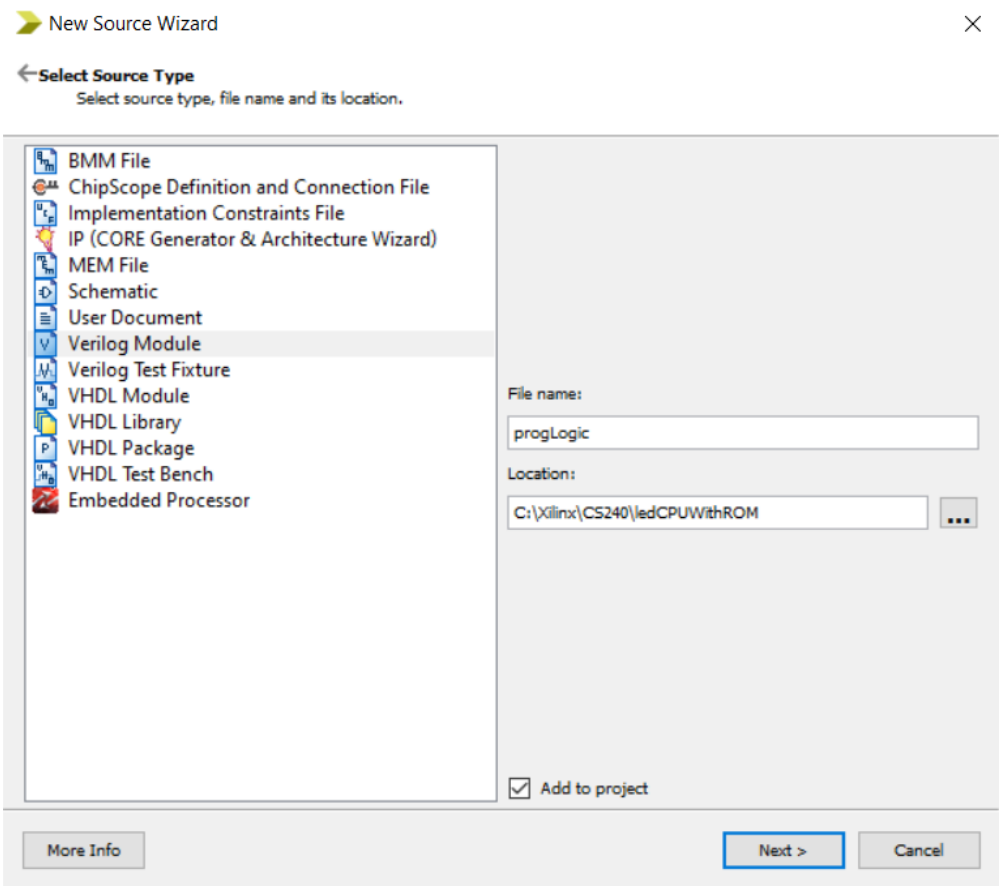
5. When the design is complete, program the Basys2 FPGA board. You should see a rotating dot running from left to right on the FPGA.
6. Show your work to the T.A.s.

Part 2:

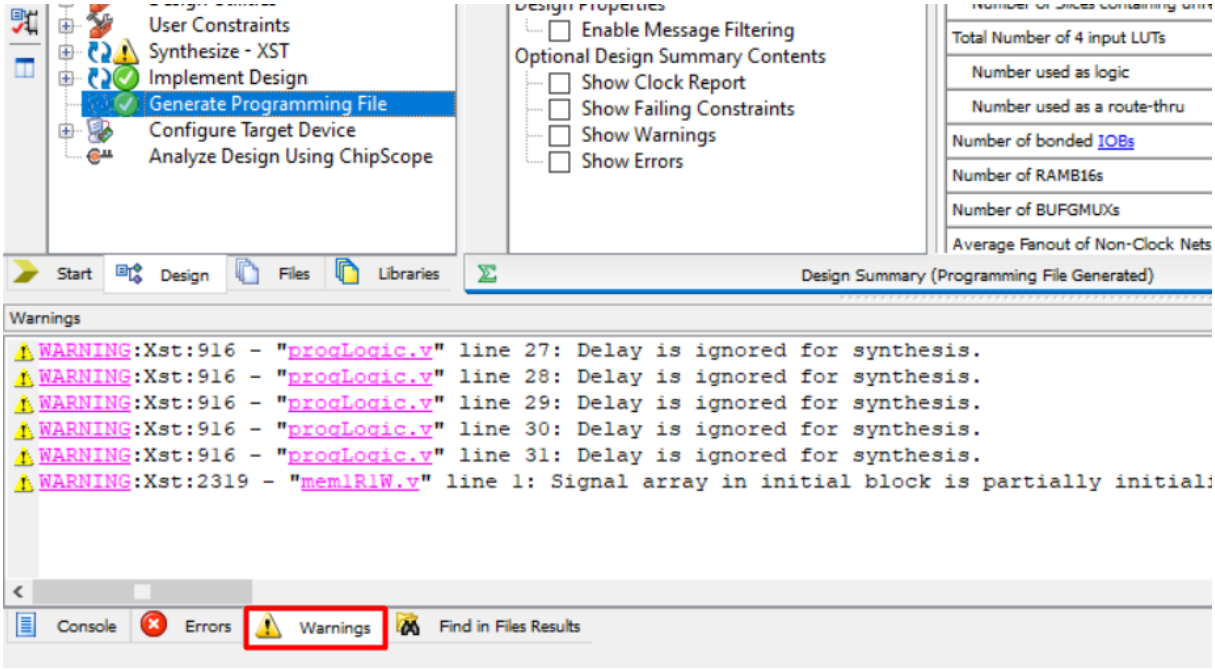
1. Open ROM.v file.
2. Modify ROM code for right to left rotation with 1.5 seconds. (You already have the bits ready from your prelab, just modify ROM.v file).
3. Do “Generate Programming File” and program the FPGA. You should see the LED shift every 1.5 seconds.
4. Show your work to the T.A.s.

Part 3:

1. Create new project in Xilinx ISE Design Suite, name it ledCPUWithRAM and add the design files using “add copy of source”. The files are LedCPUWithRAM.v, LedCPUWithRAM.ucf, Delay.v, mem1R1W.v, init.v, debounce.v.
2. Copy your LedCPUCORE.v from Part1 (LedCPUWithROM) to this project using “add copy of source”.
3. Right click on “xc3s100e-4cp132” and choose “New Source”. Choose “Verilog Module” and write “progLogic.v” as the File name.



4. Click on “Next”, “Next” and then “Finish”.
5. Modify the progLogic.v file as required so that it can take the “switch” input when the “enter” button is pressed and save it to memory, first saving the switches to bits [15:8] in address 0, then to bits [7:0] in address 0, then to bits [15:8] in address 1 and so on.
6. Do “Generate Programming File” and make sure you get the same warnings as in the figure below.



7. Program the FPGA and test it out.
8. When it works, show your work to the T.A.s.