Eric Chuong

Rebecca Grob

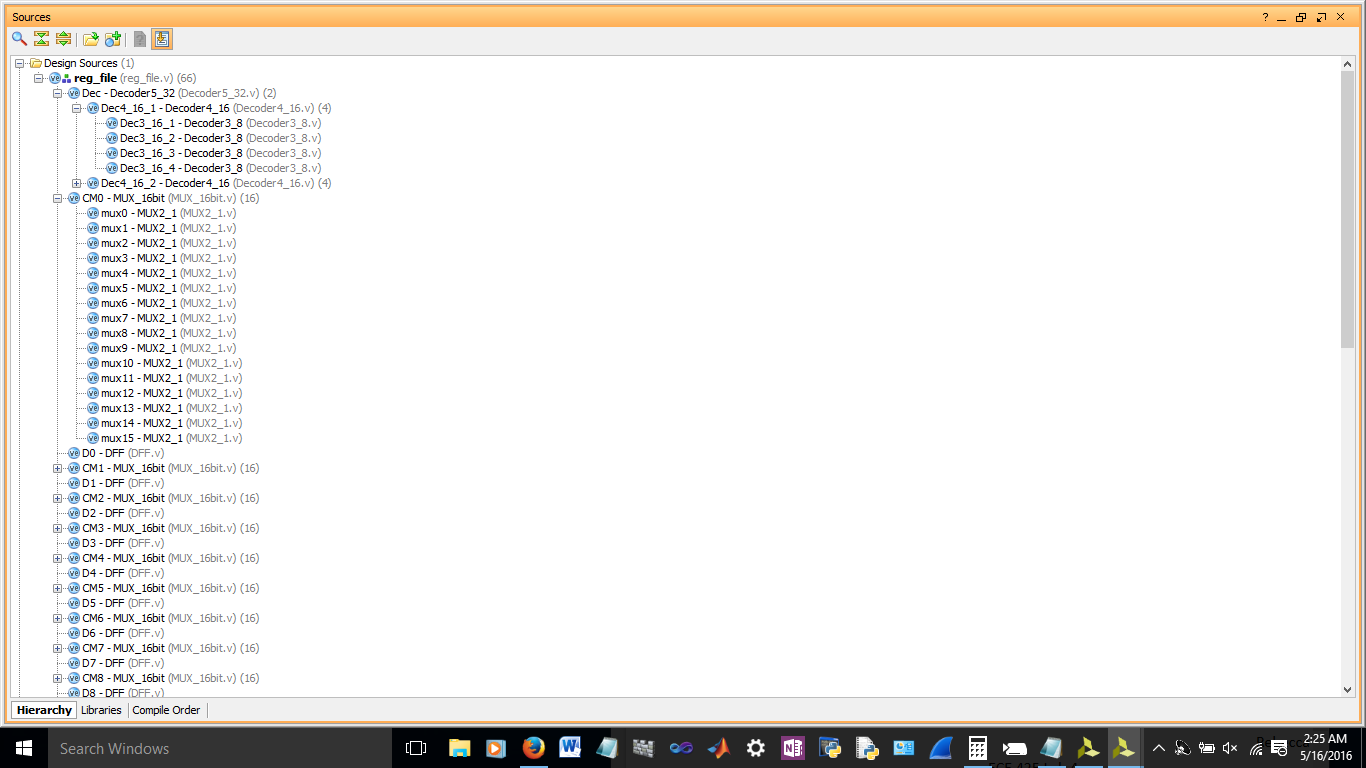
ECE 425 Lab 4

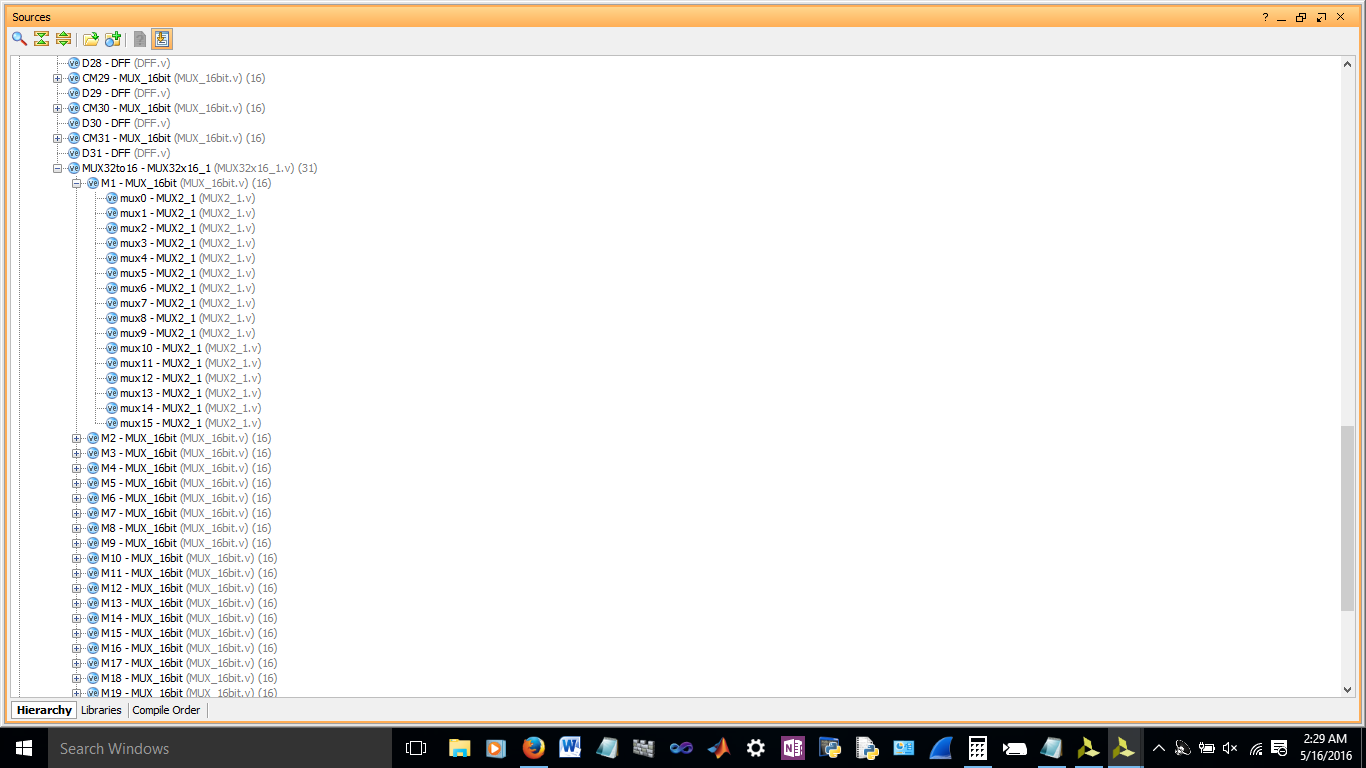
Introduction and Objective

The objective of this lab is to create three parts: the data memory, the instruction memory, and the control unit.

**Part One**

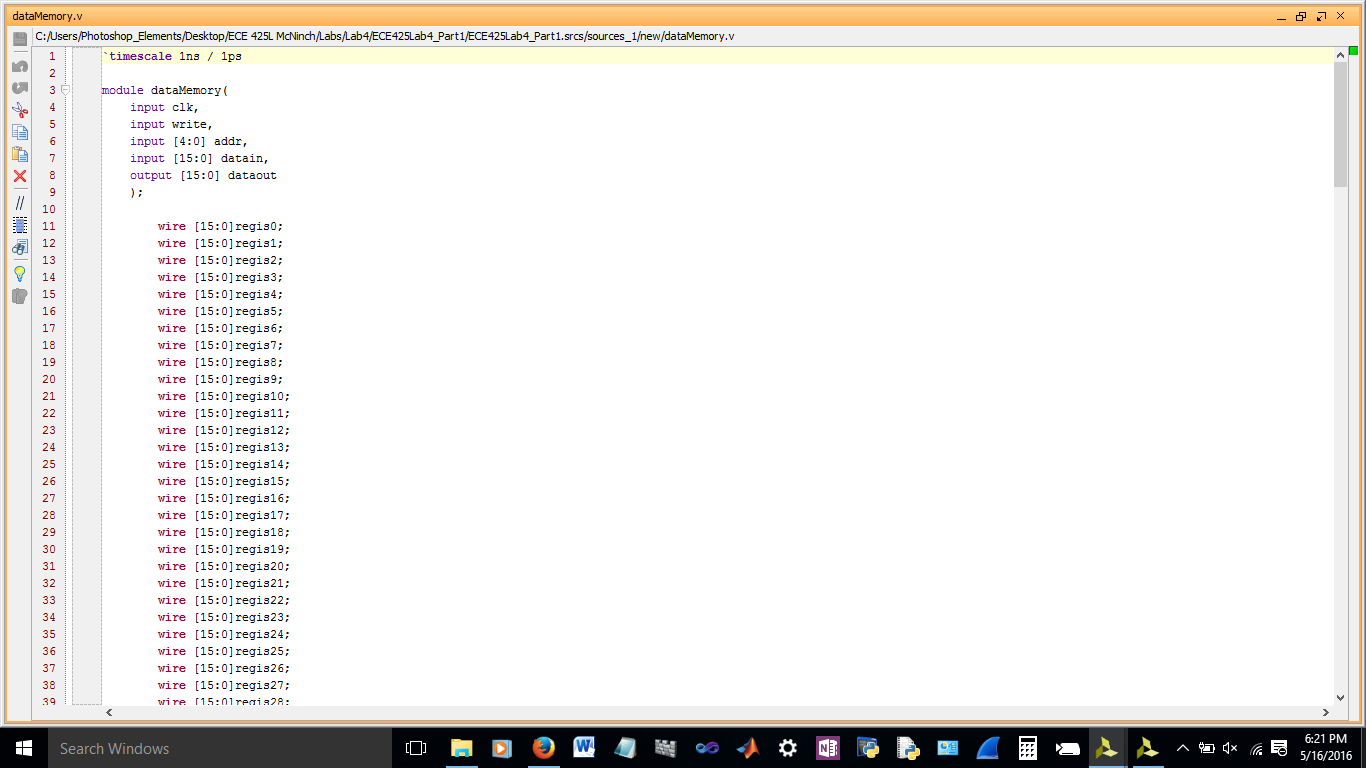
Hierarchy

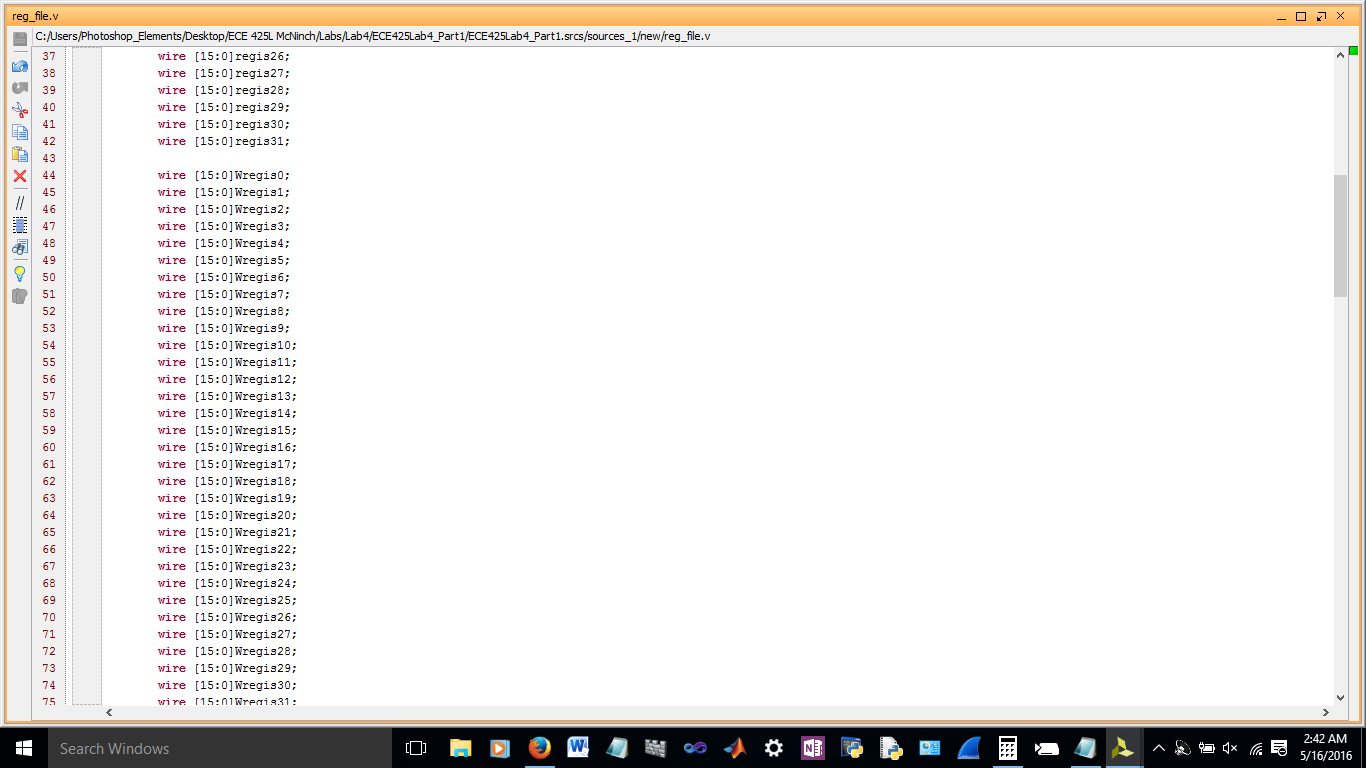


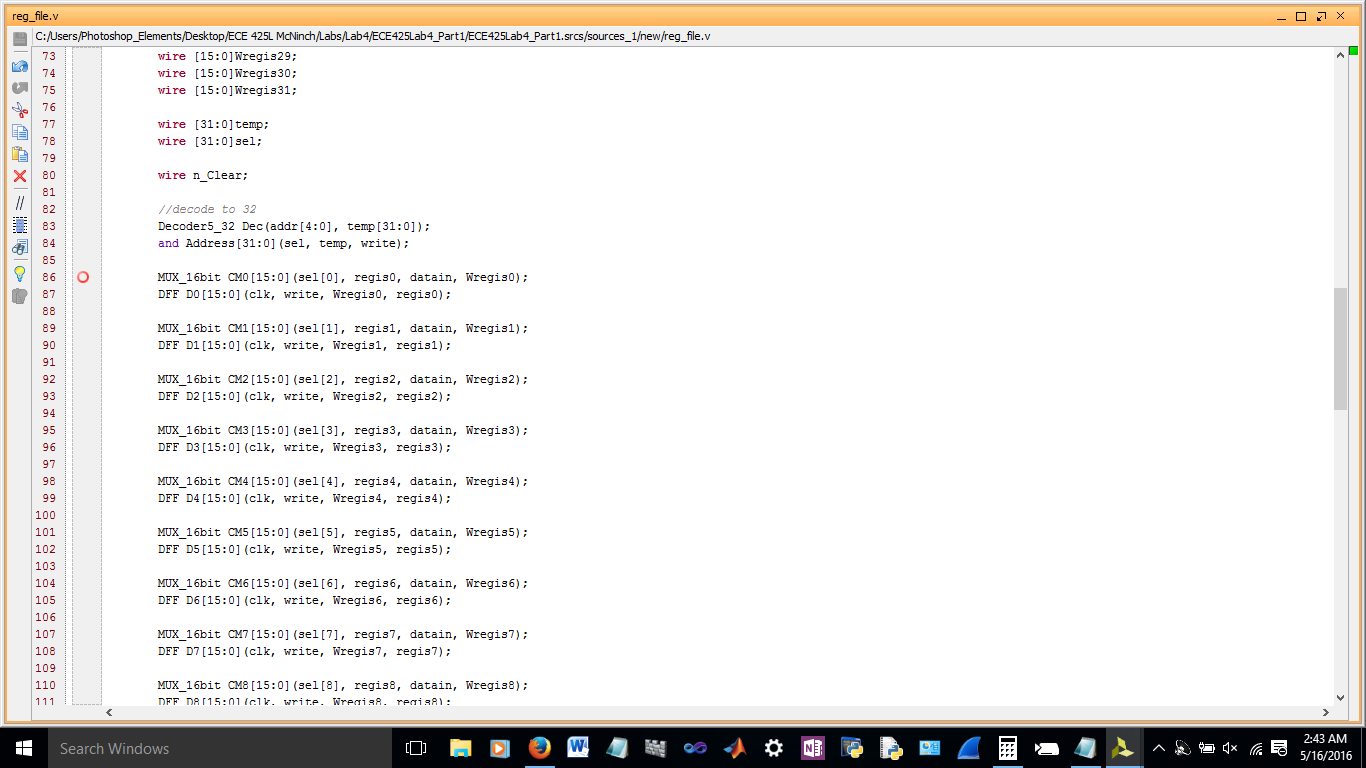


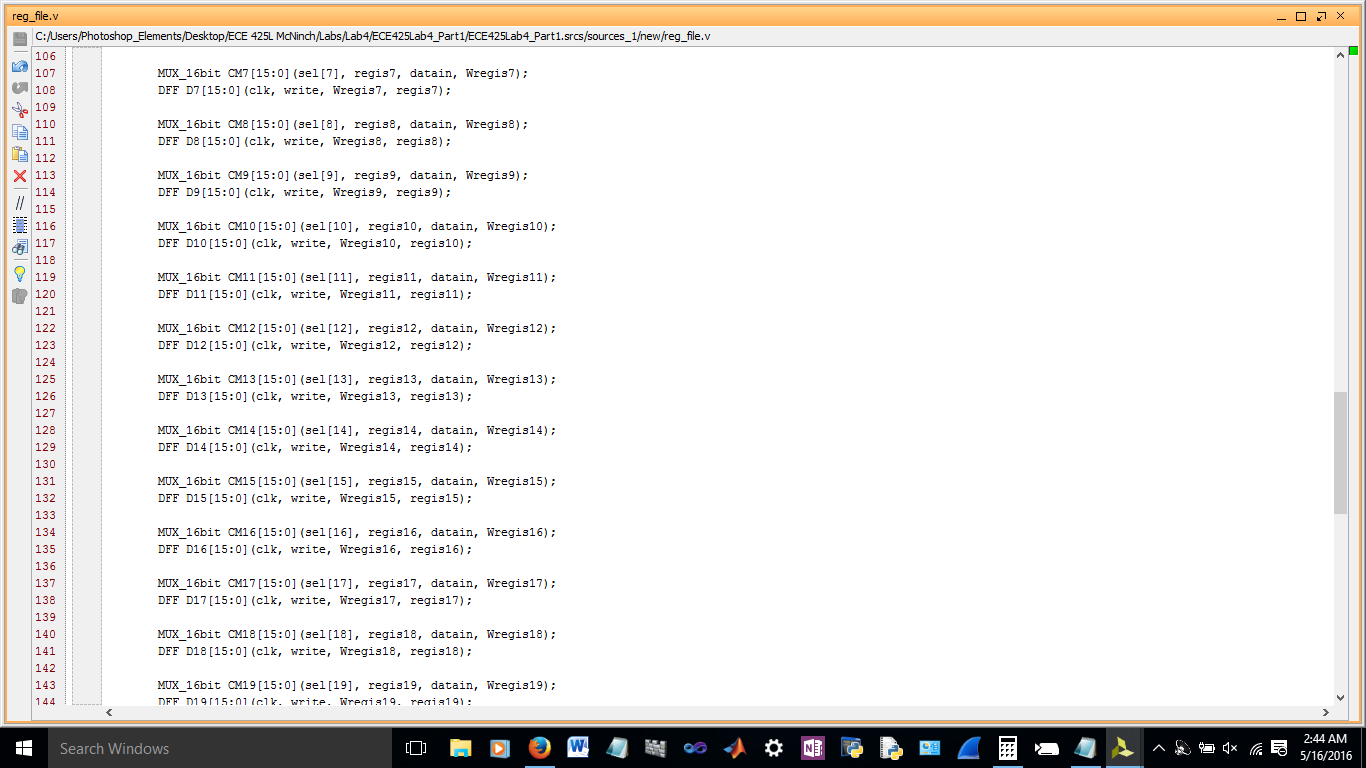
*dataMemory*

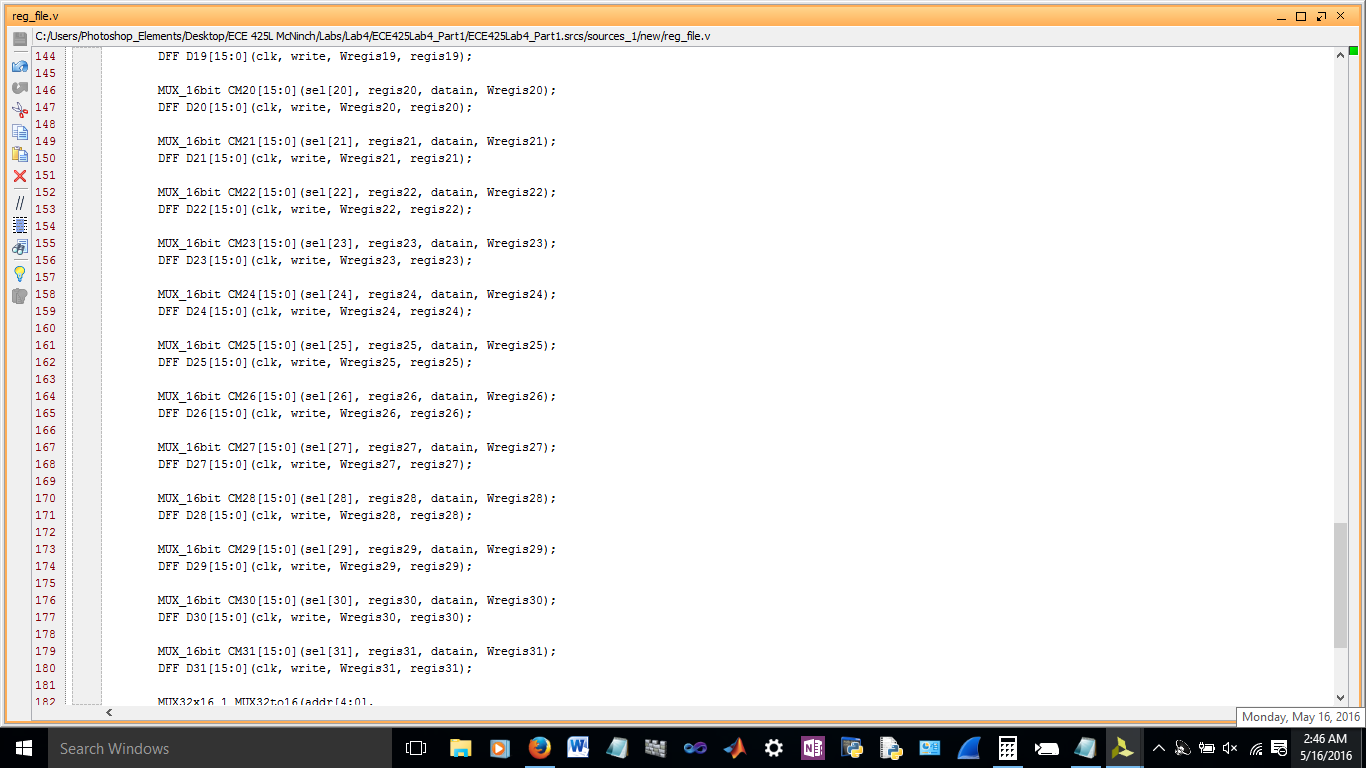
Code

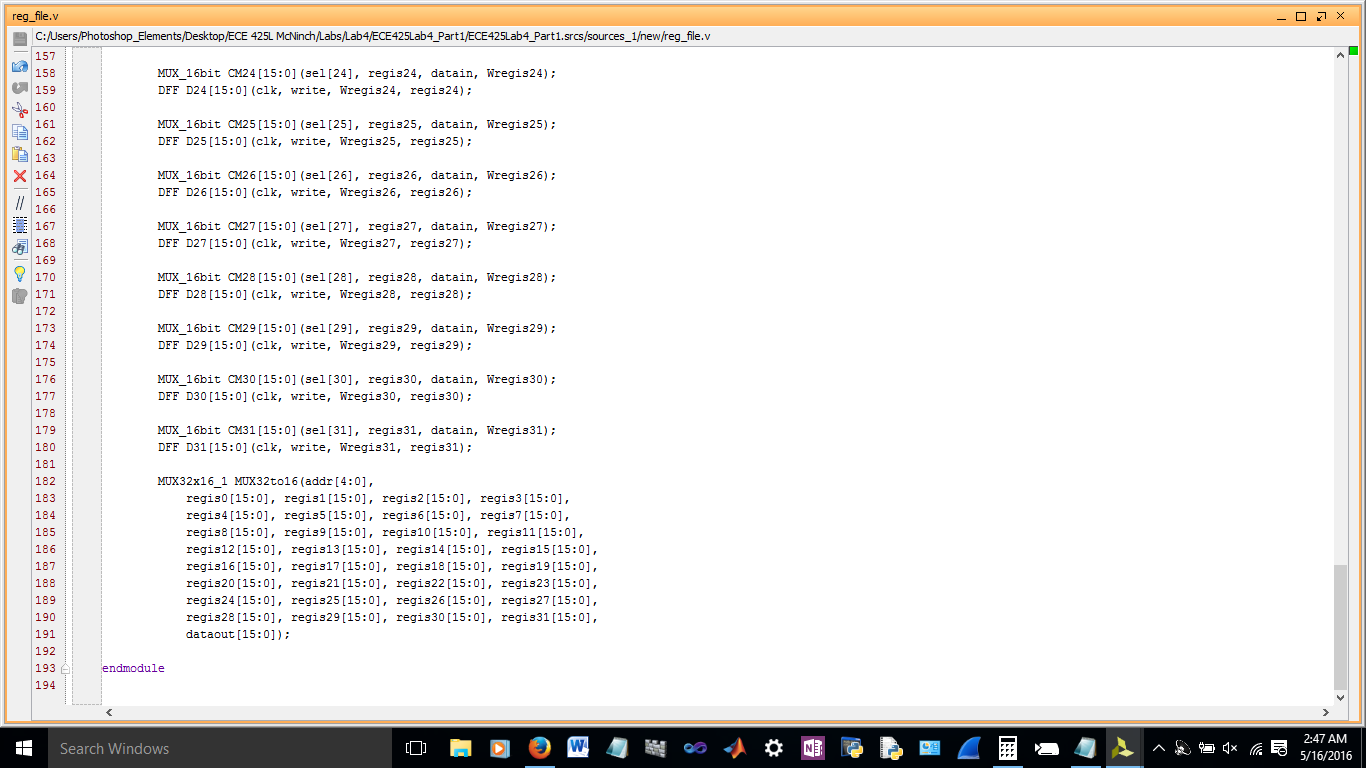




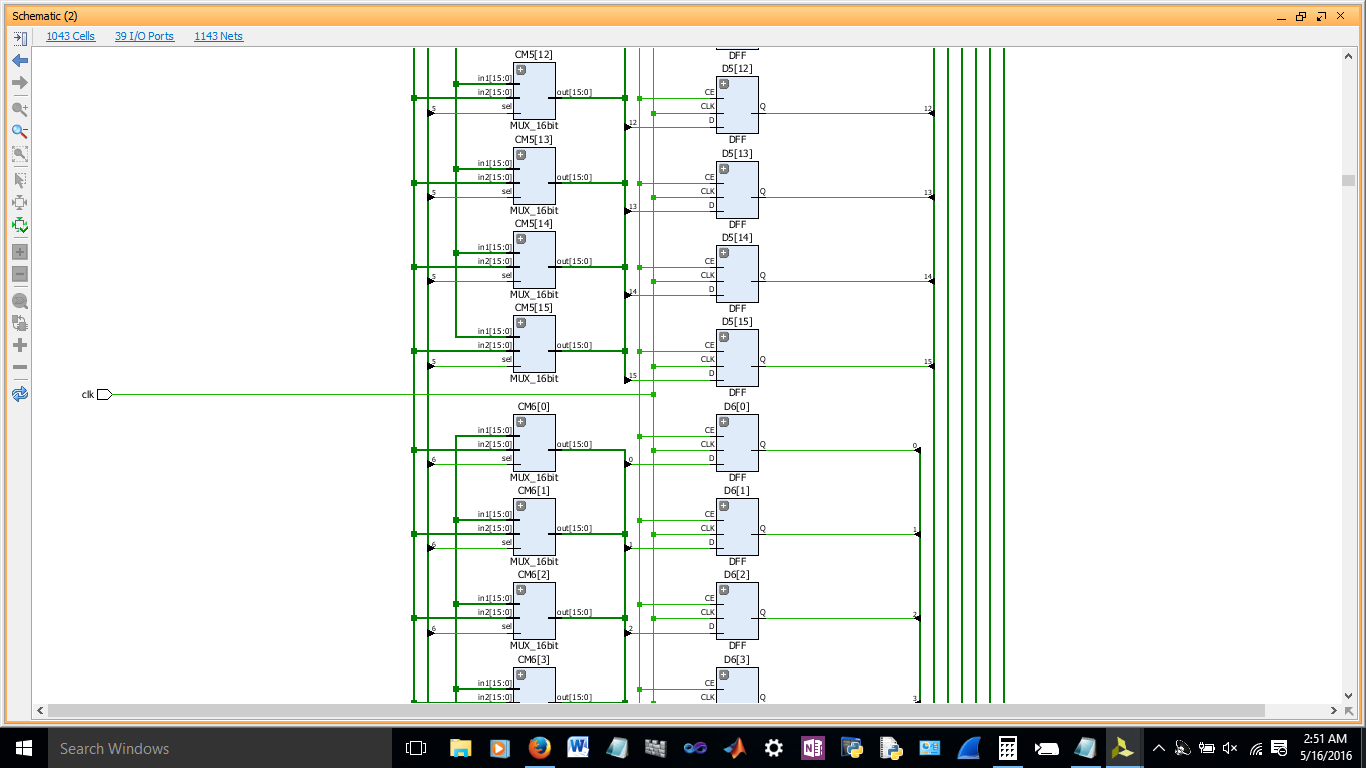


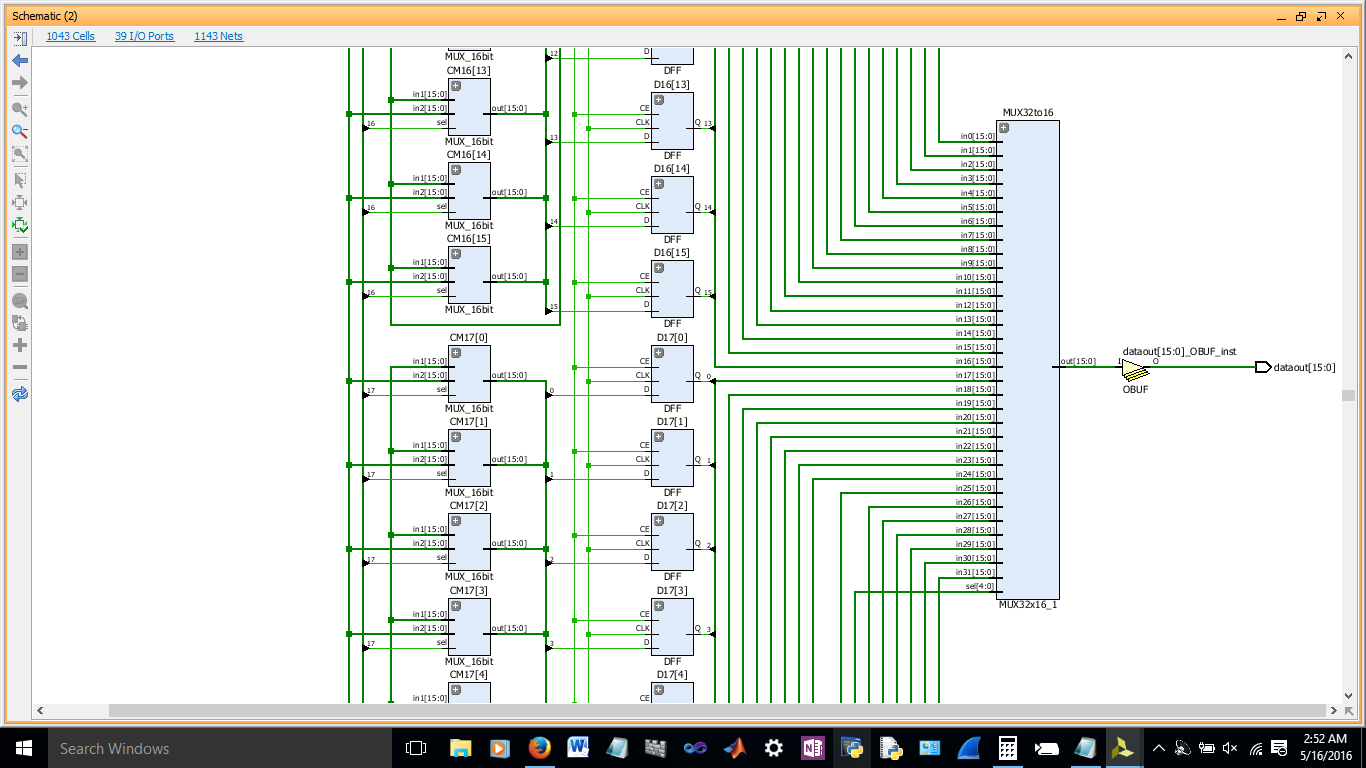


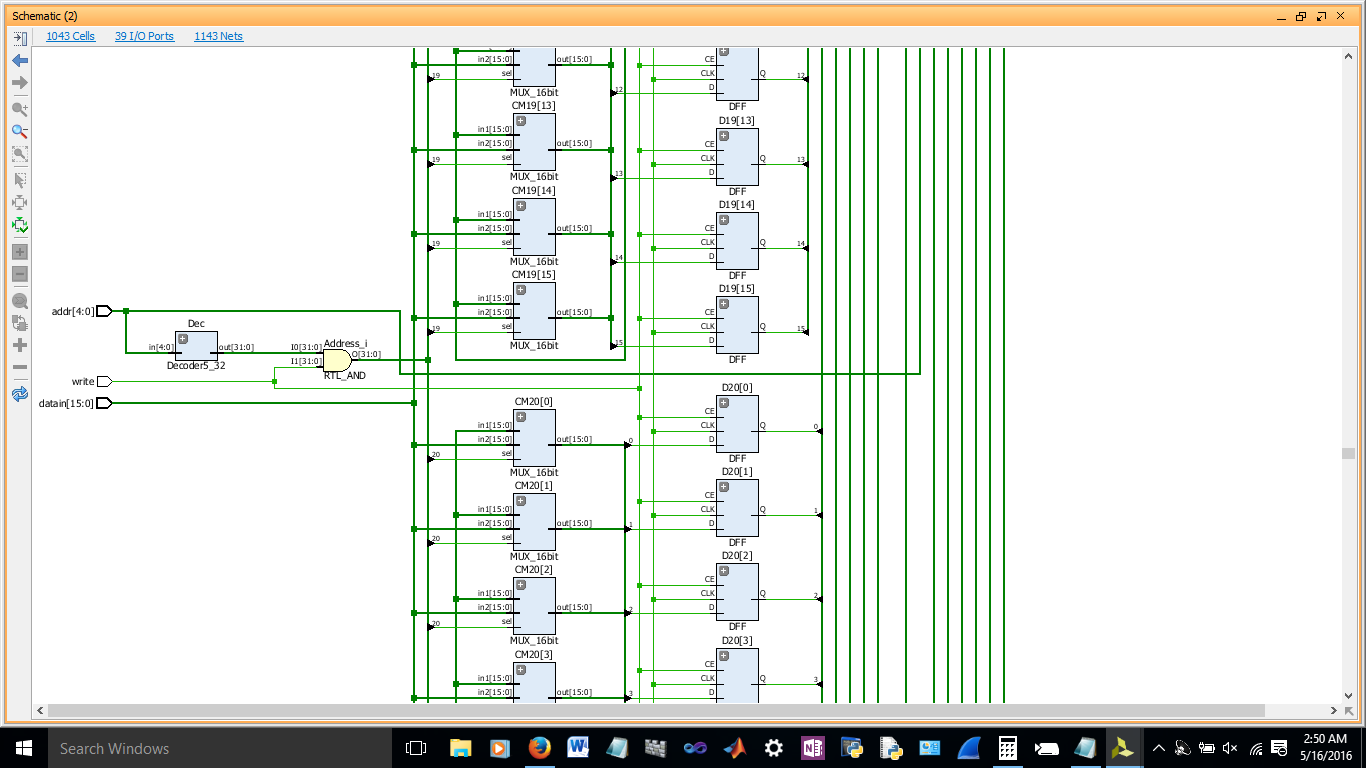




Schematic

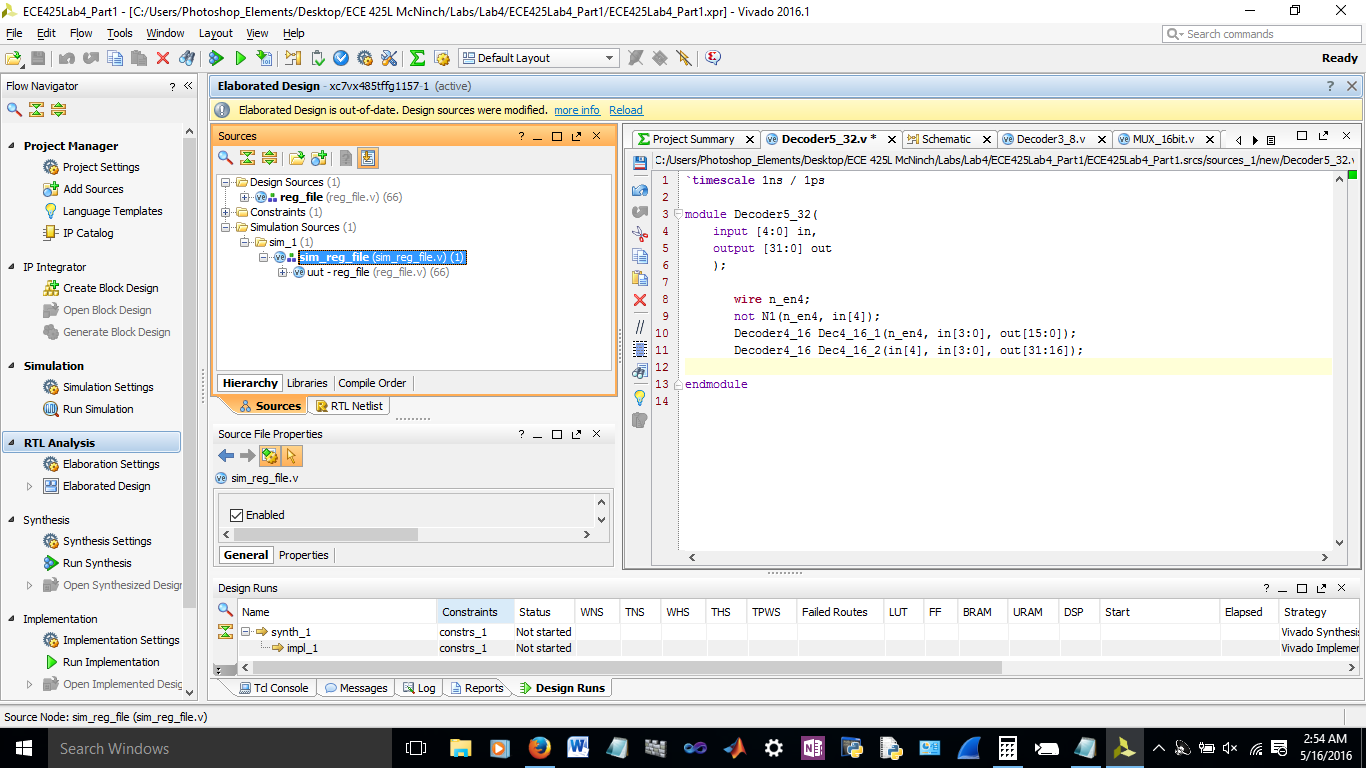






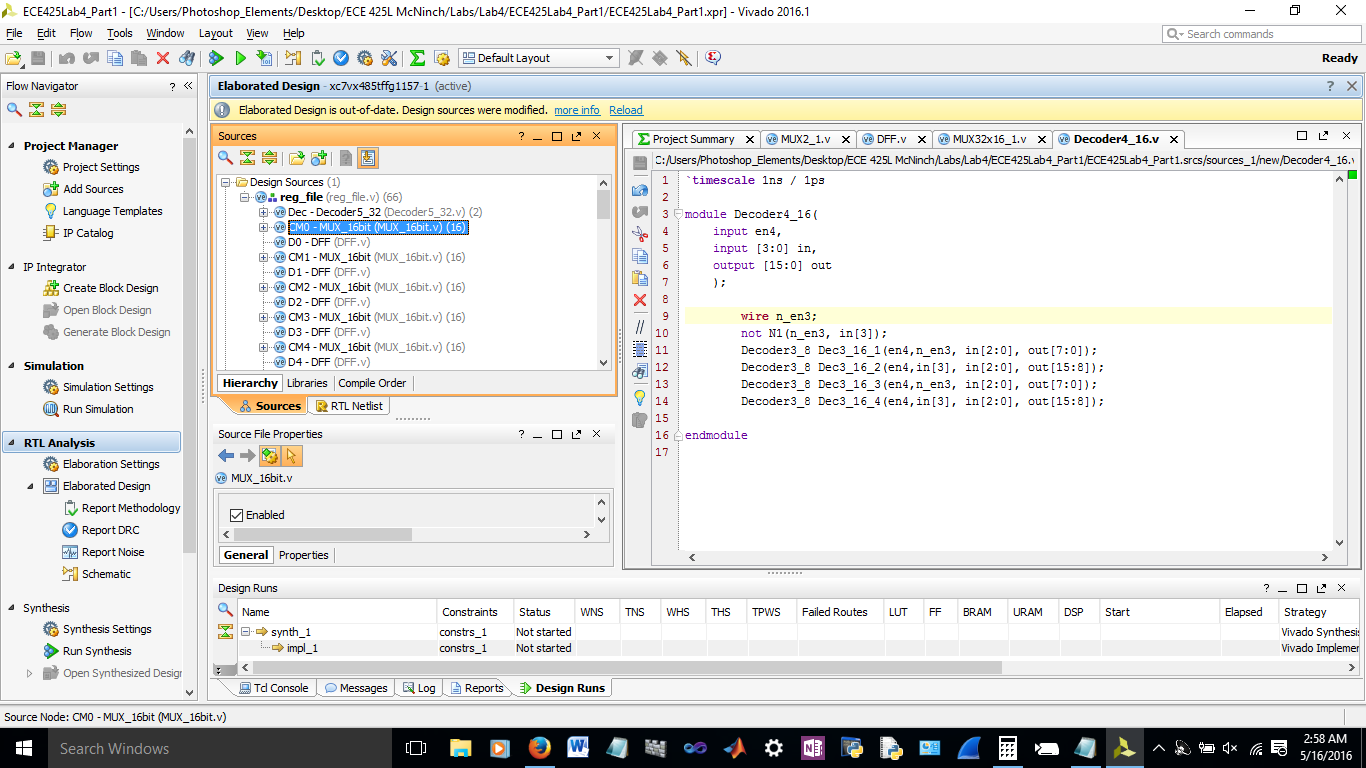
*Decoder5\_32*

Code



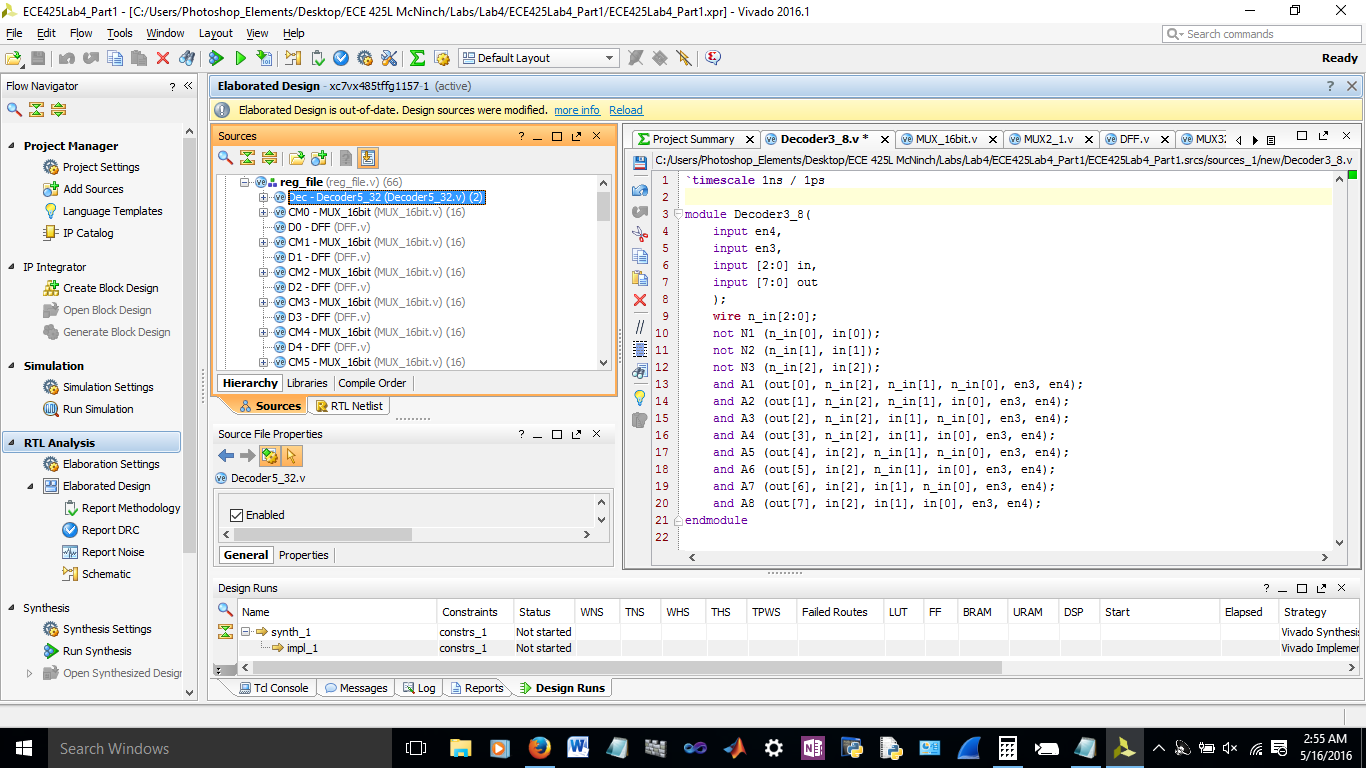
*Decoder4\_16*

Code



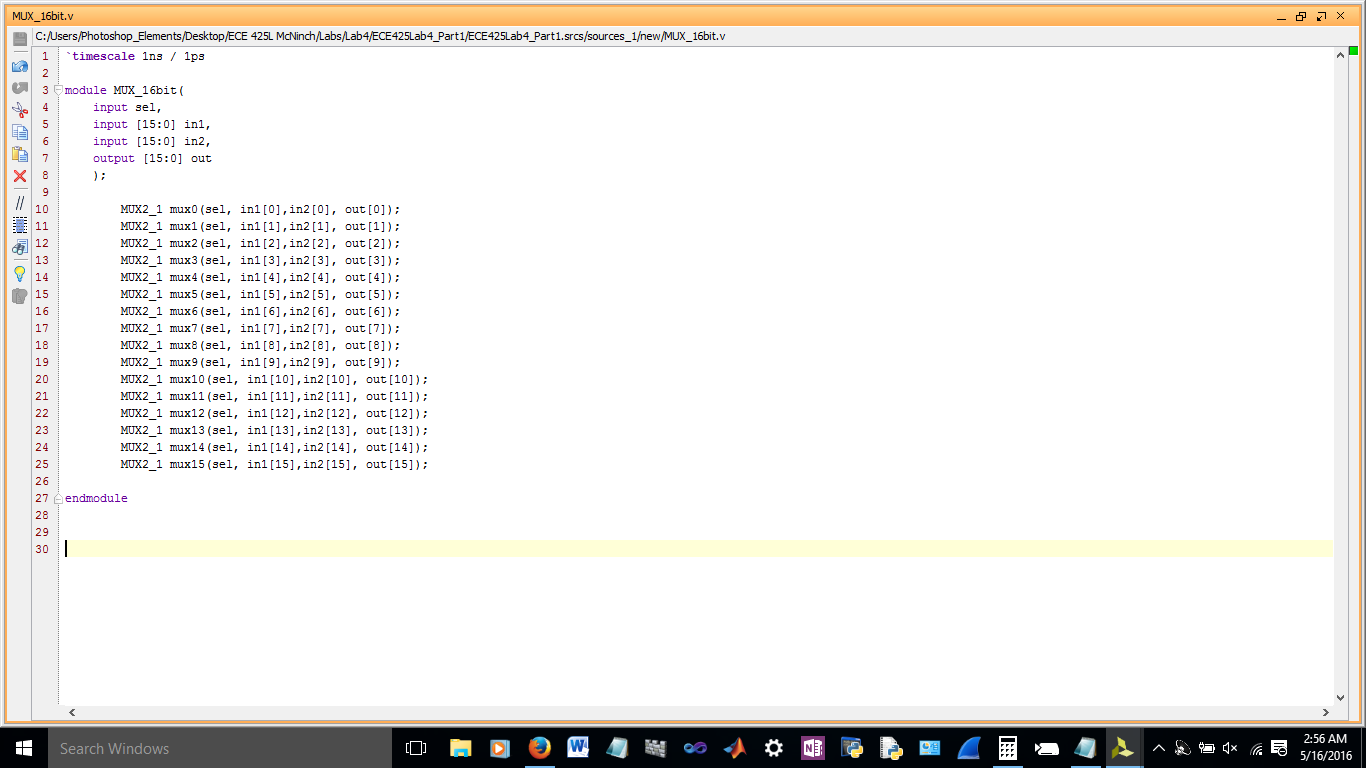
*Decoder3\_8*

Code



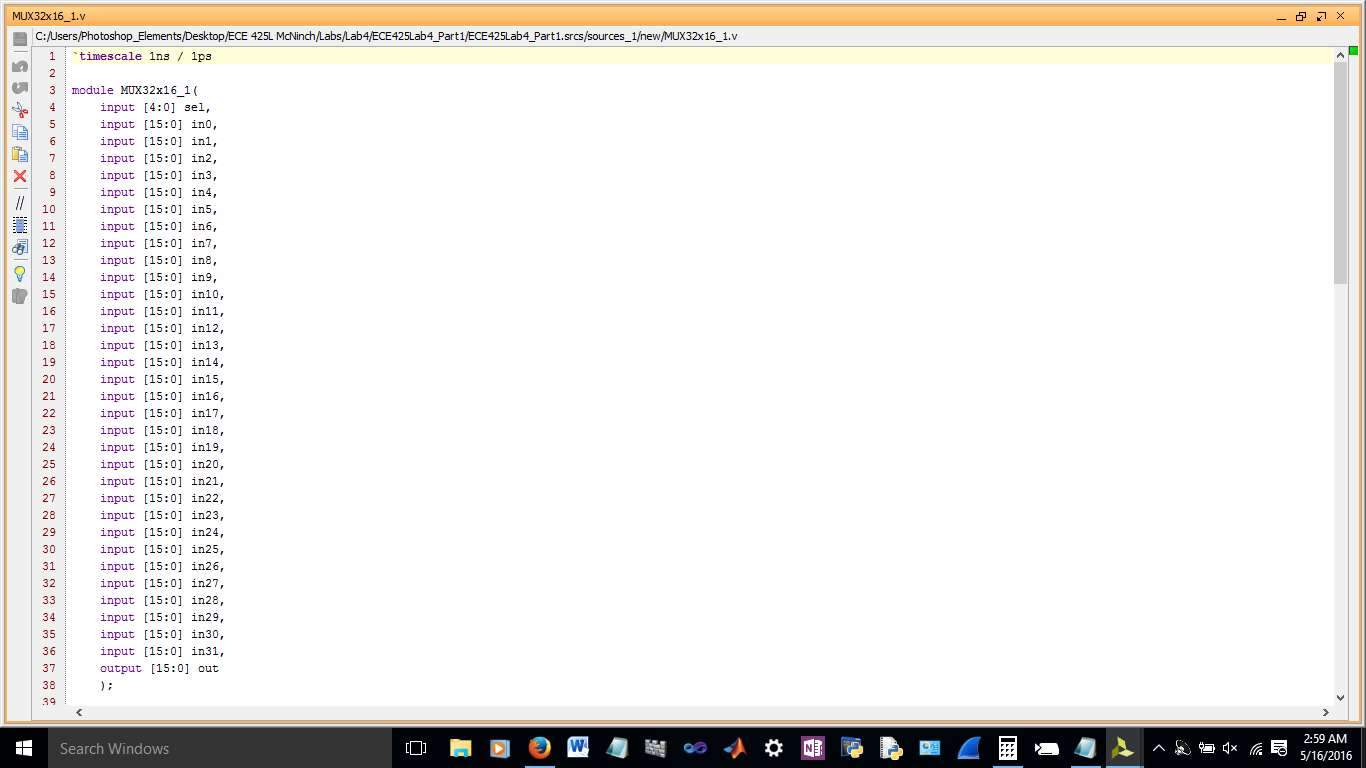
*MUX\_16bit*

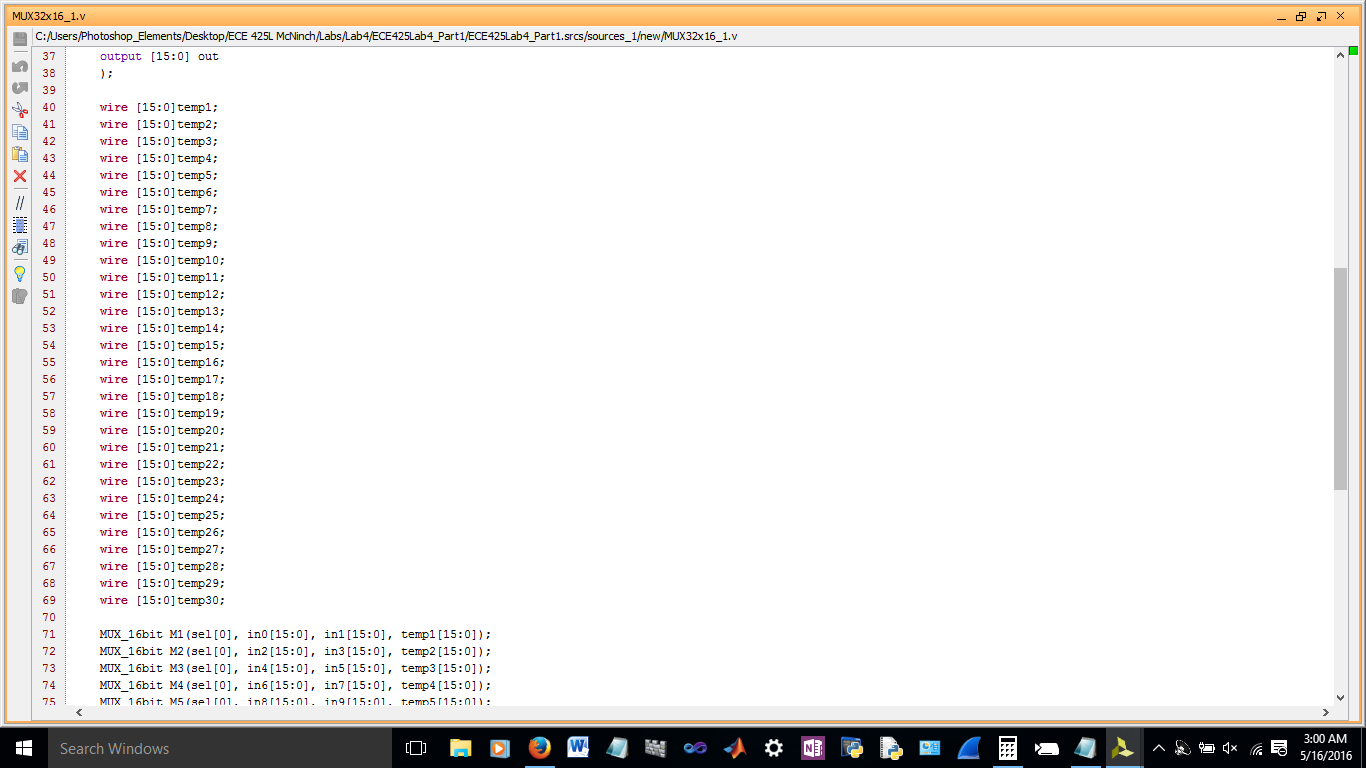
Code

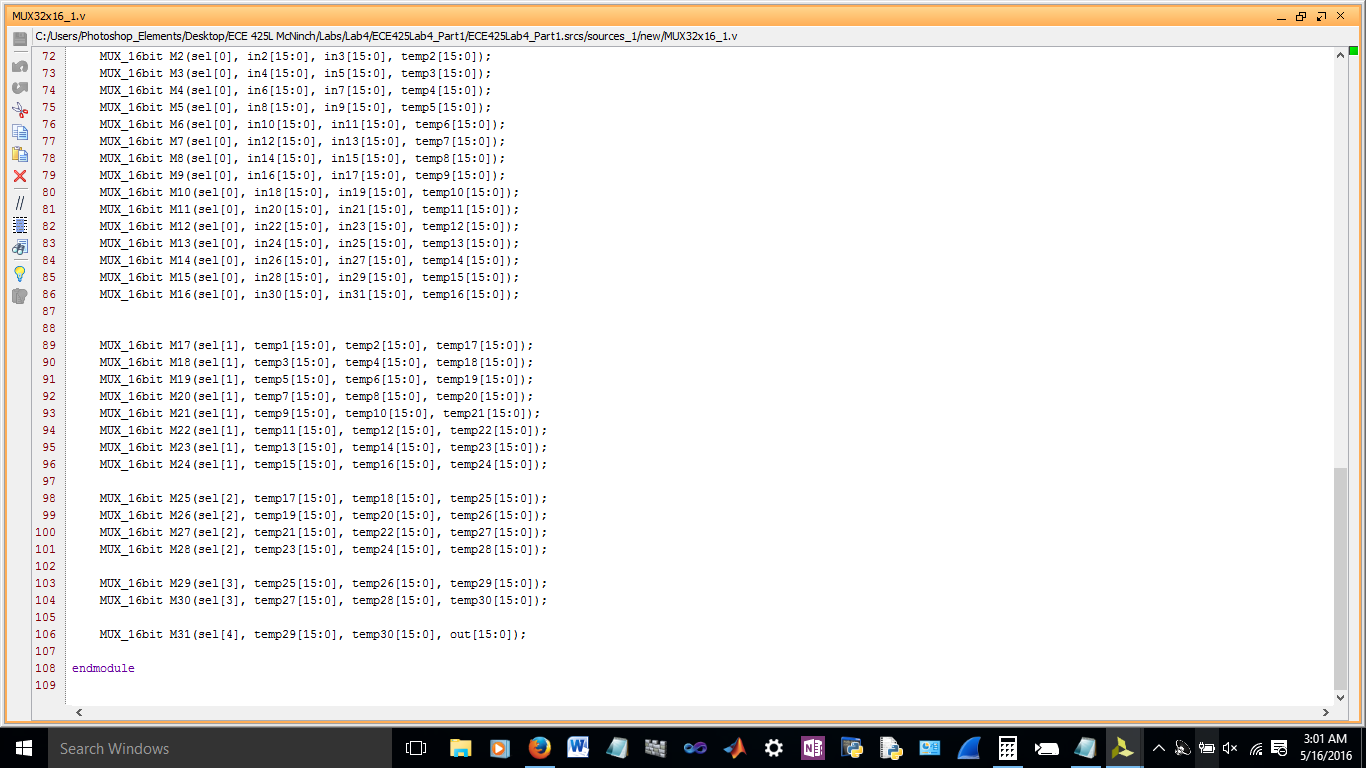


*MUX32x16\_1*

Code

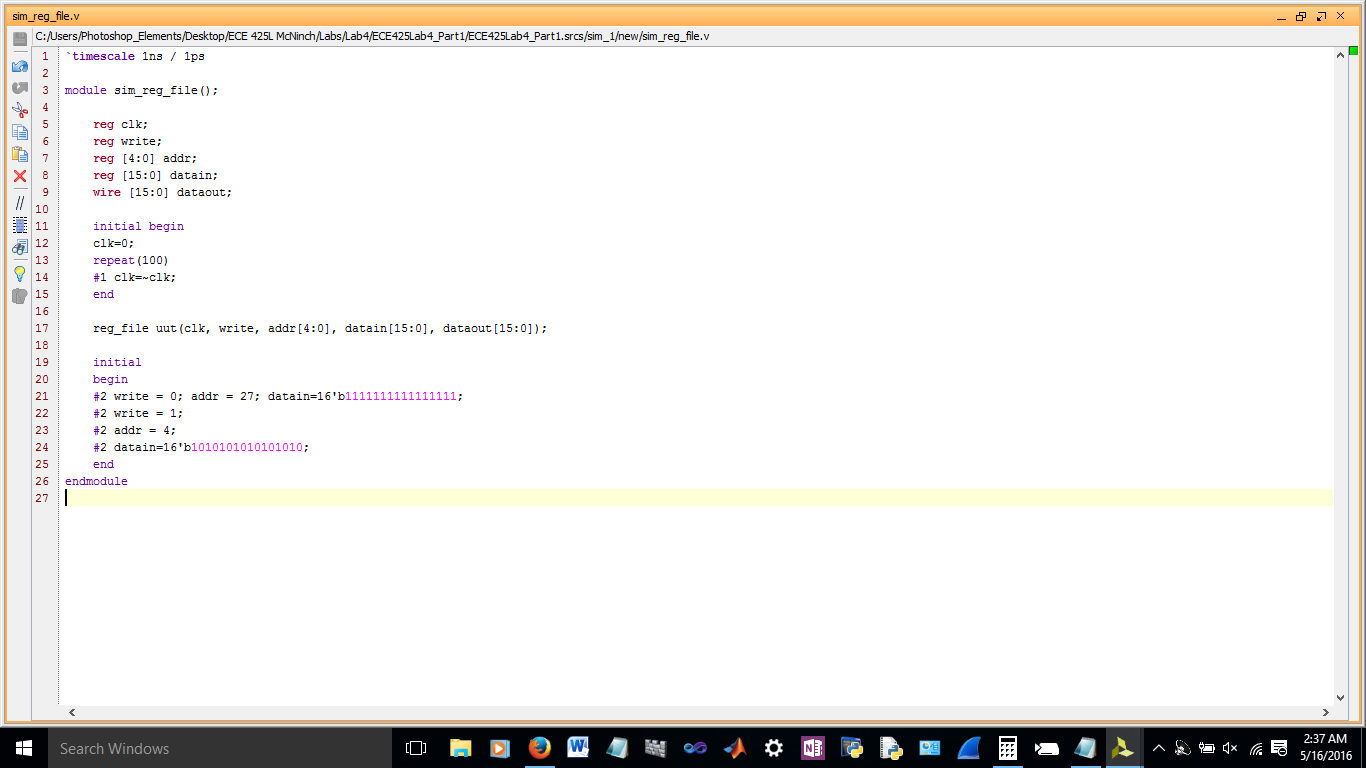




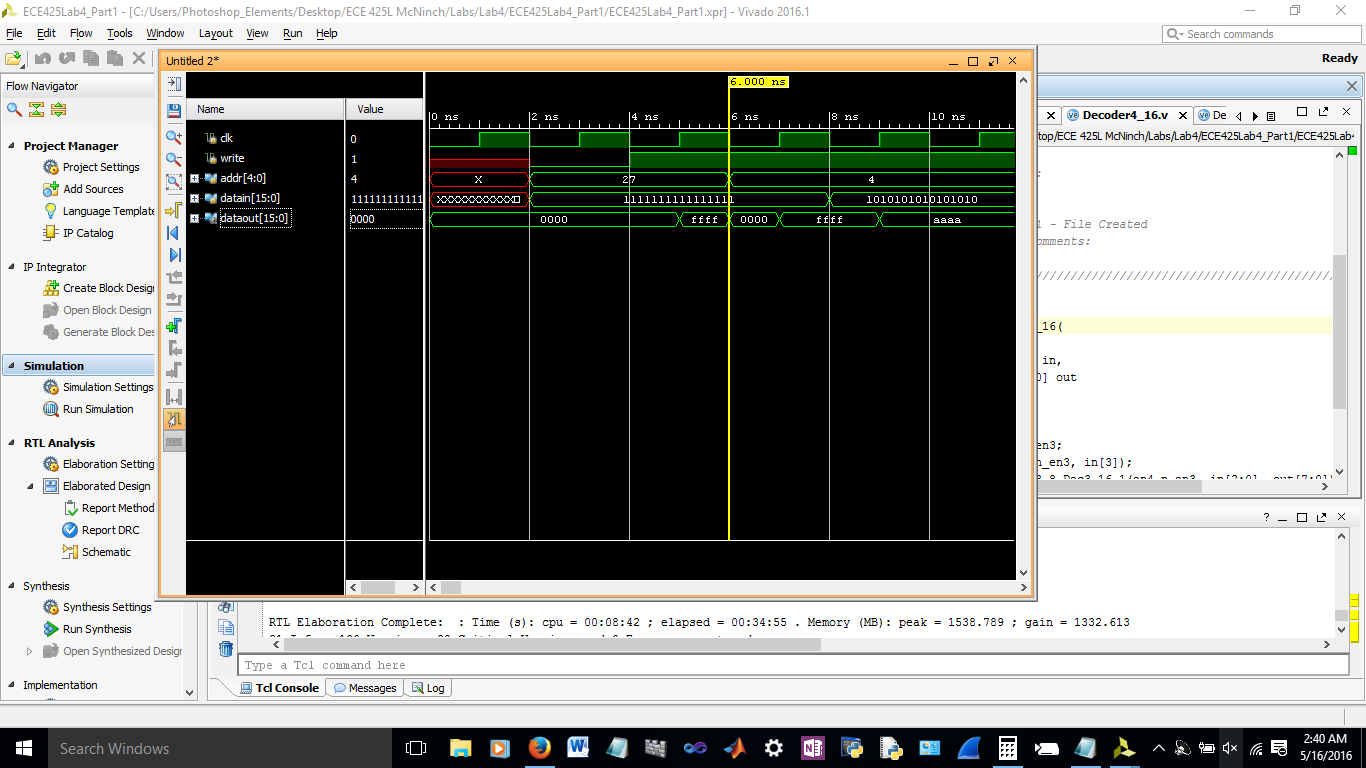


*Simulation*

Code



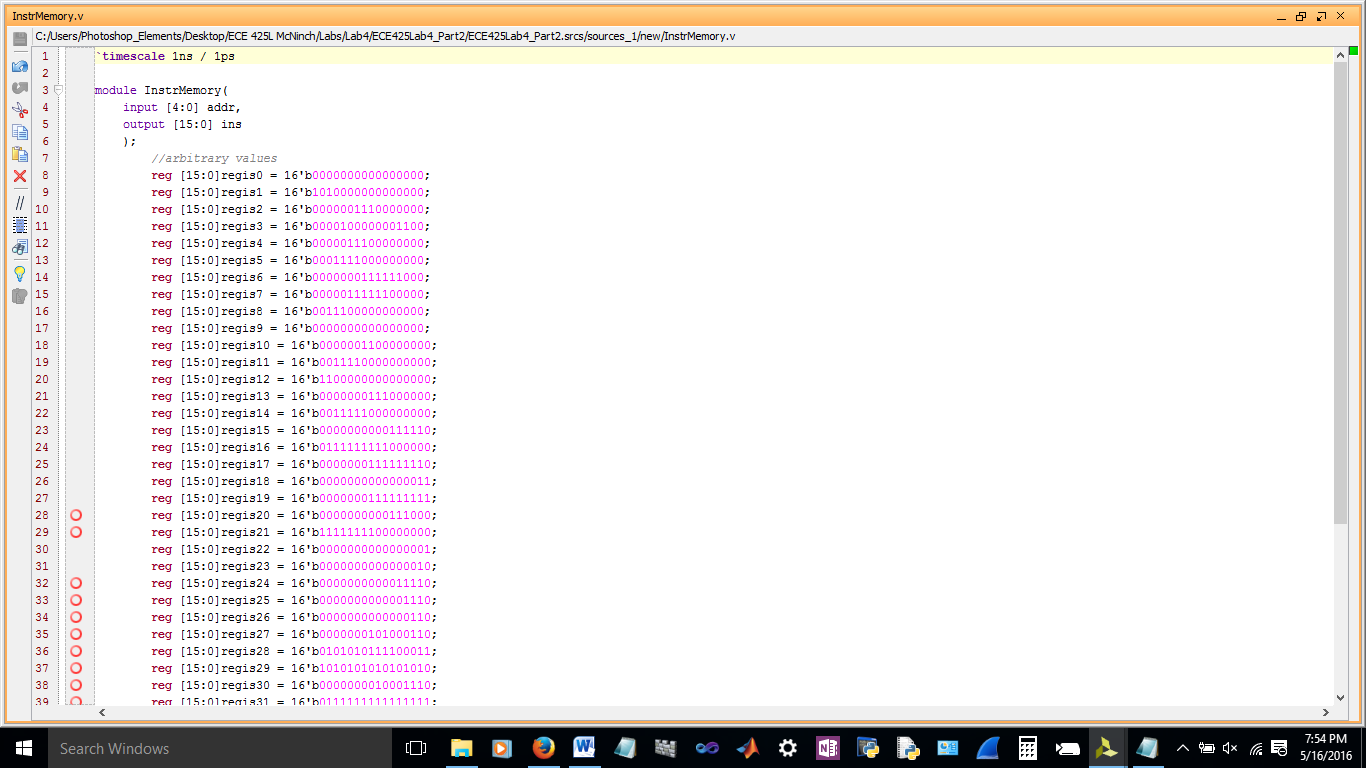
Output

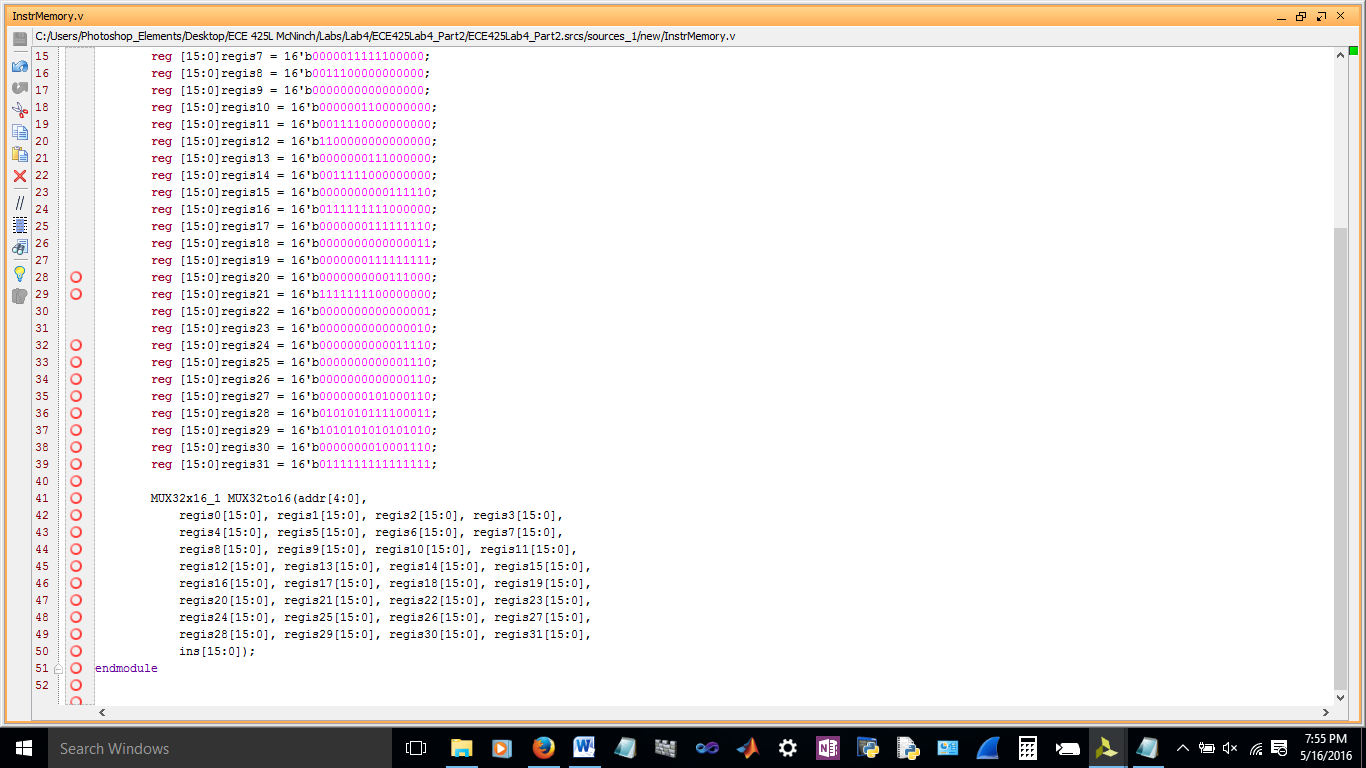


**Part Two**

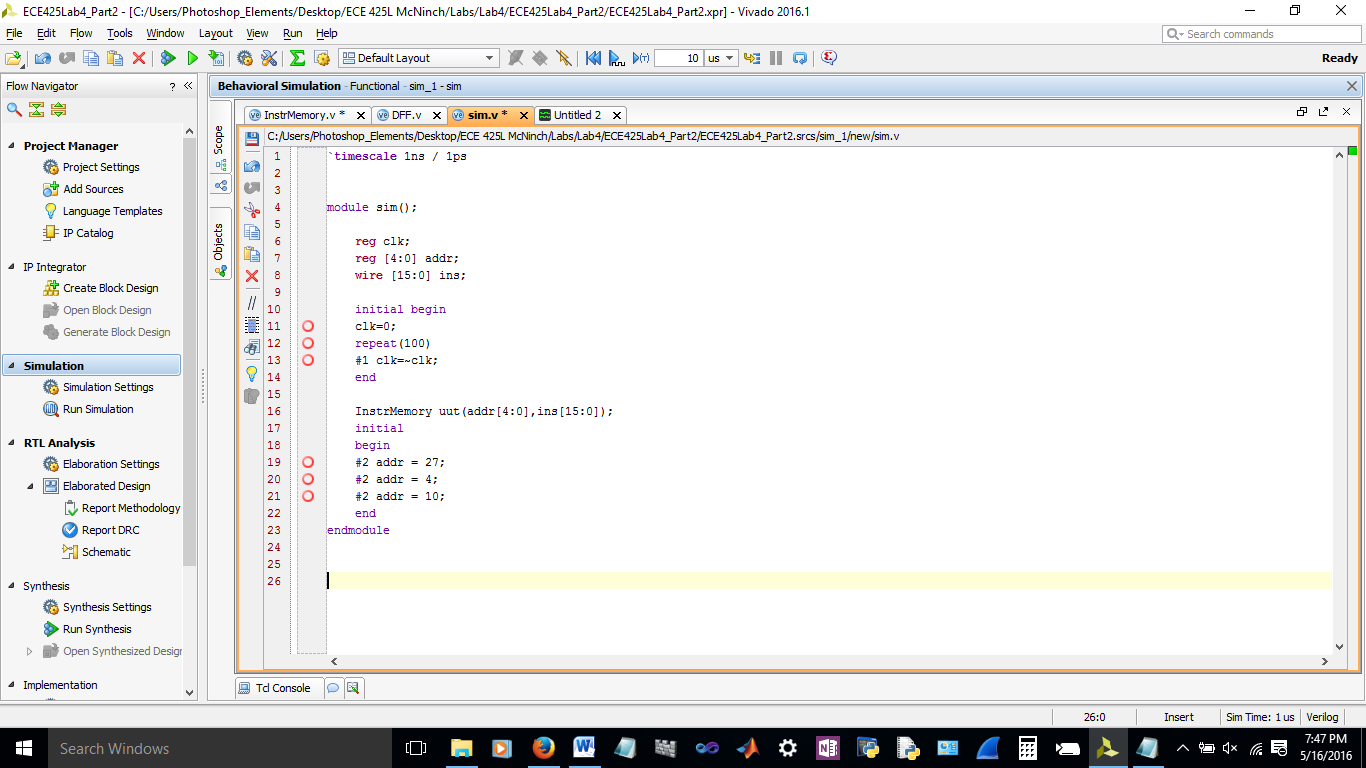
*InstrMemory*

Code

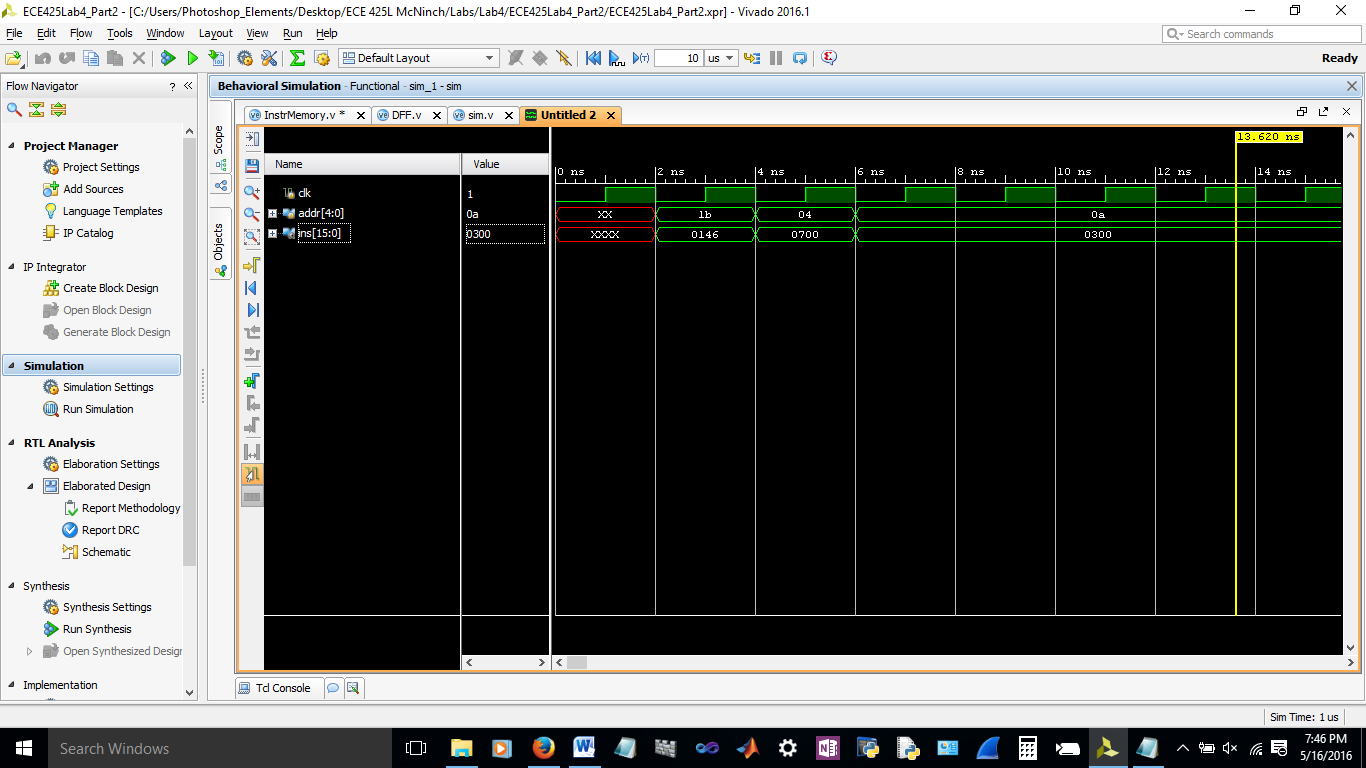




Simulation (clock added for convenience)



Output



**Control Unit**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **opcod** | **instrn** | **RegDest** | **ALUSrc** | **MemToReg** | **Reg**  **Write** | **MemWrite** | **Branch** | **ALUop2** | **ALUop1** | **ALUop0** | **Jump** |
| **0** | **AND** | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | x | 0 |
| **1** | **OR** | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | x | 0 |
| **2** | **ADD** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **6** | **SUB** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| **7** | **SLT** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | x | 0 |
| **8** | **LW** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **A** | **SW** | x | 1 | x | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **E** | **BNE** | x | 0 | x | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| **F** | **JMP** | x | x | x | 0 | 0 | x | x | x | x | 1 |

When opcod is 0-7, we are using R type instructions and so the control will send the same signals. R type instructions need the RegDest to be 1 and they need to write into the register so RegWrite is 1 also. The ALUSrc picks the data out of the register. It does not touch the memory and the ALUop that we created in the previous labs requires that R type is 10. Lastly, R types do not branch or jump.

When we use the instruction load word, the RegDest is 0 and the ALUSrc will choose the data out of the register to put into the ALU. The important part is that MemRead is 1 and MemToReg is 1 because we want to take what is in the data memory and put it into the register.

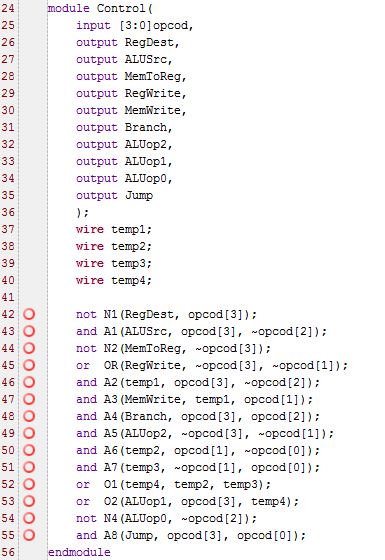
The important part for store word instruction is that we choose the address using ALUSrc as 1 and we do not write into the registers. Instead we are writing into the memory so MemWrite is 1.

For Branch instruction, it is similar to R type except we are not writing into the register, branch is 1, and ALUop0 is 1 instead of ALUop1.

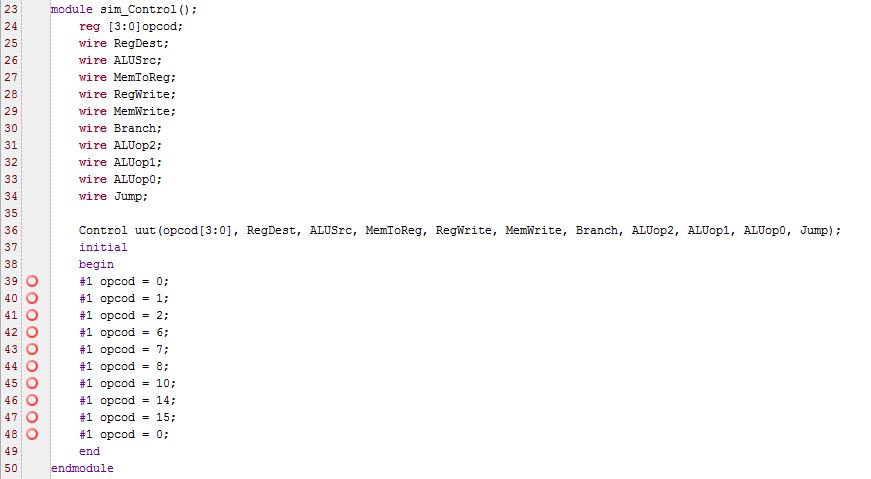
For the Jump instruction, the only thing that matters is that we are not writing into the registers or the memory and the jump signal is 1.

Using this table, it is possible to create Kmaps for each control signal. After doing so, we were able to generate this code.

Code



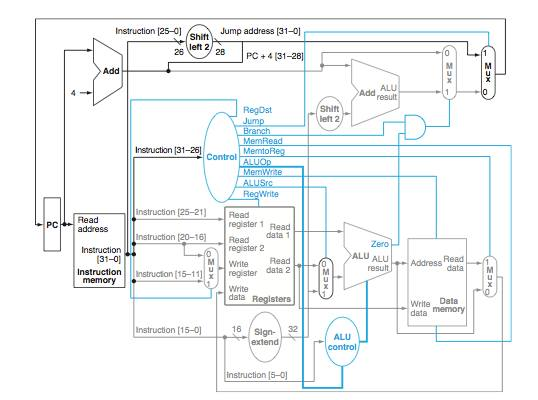
Simulation code



Simulation



Schematic



Discussion of Results

Overall this lab was pretty simple. The only difficult part was making sure the truth tables were correct and then drawing each K-map to determine the control unit. We are confident that our results are correct and there should be no errors.