Lab 2

Ishan Dane idane
Ismail Memon ishmemon

Design Procedure

In this lab we were tasked with working with and implementing memory designs. We are asked to create memory from Quartus's library modules, from our own System Verilog code and a library memory with its own read and write operations We are provided with conceptual RAM and RAM register diagrams to implement into design and code through System Verilog. Depending on the task, the DE1_SoC board is used to simulate RAM input and output with input: switches SW3-SW1 specifying DataIn, switches SW8-SW4 to specify address, SW0 as the Write signal and KEY0 as the Clock input. Input information is also displayed on the board by the HEX display HEX5-HEX4 for the address and HEX1 for DataIn. Output DataOut on the board is shown by the HEX display on HEX0.

Task #1

For task 1, we are asked to create the memory unit specified by the lab manual shown on Figure 1 using the prebuilt modules provided to us by Quartus libraries. In this RAM model, input data ("Address", "DataIn", "Write") is processed through registers before it is inputted into the RAM memory unit.

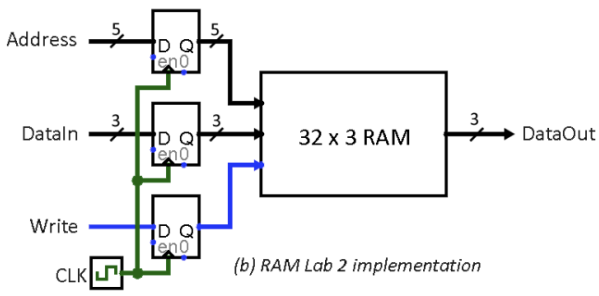


***Figure 1:** 32 x 3 RAM module for Lab 2.*

To implement this design, we used 1 bit flip flops to process each bit of Address (5 bits), DataIn (3 bits), and Write (1 bit) in our code and assigned new variables that receive the output data Q storing each inputs' bits. These variables are then inputted into the prebuilt RAM modules provided and DataOut[2:0] is outputted from the RAM memory unit.

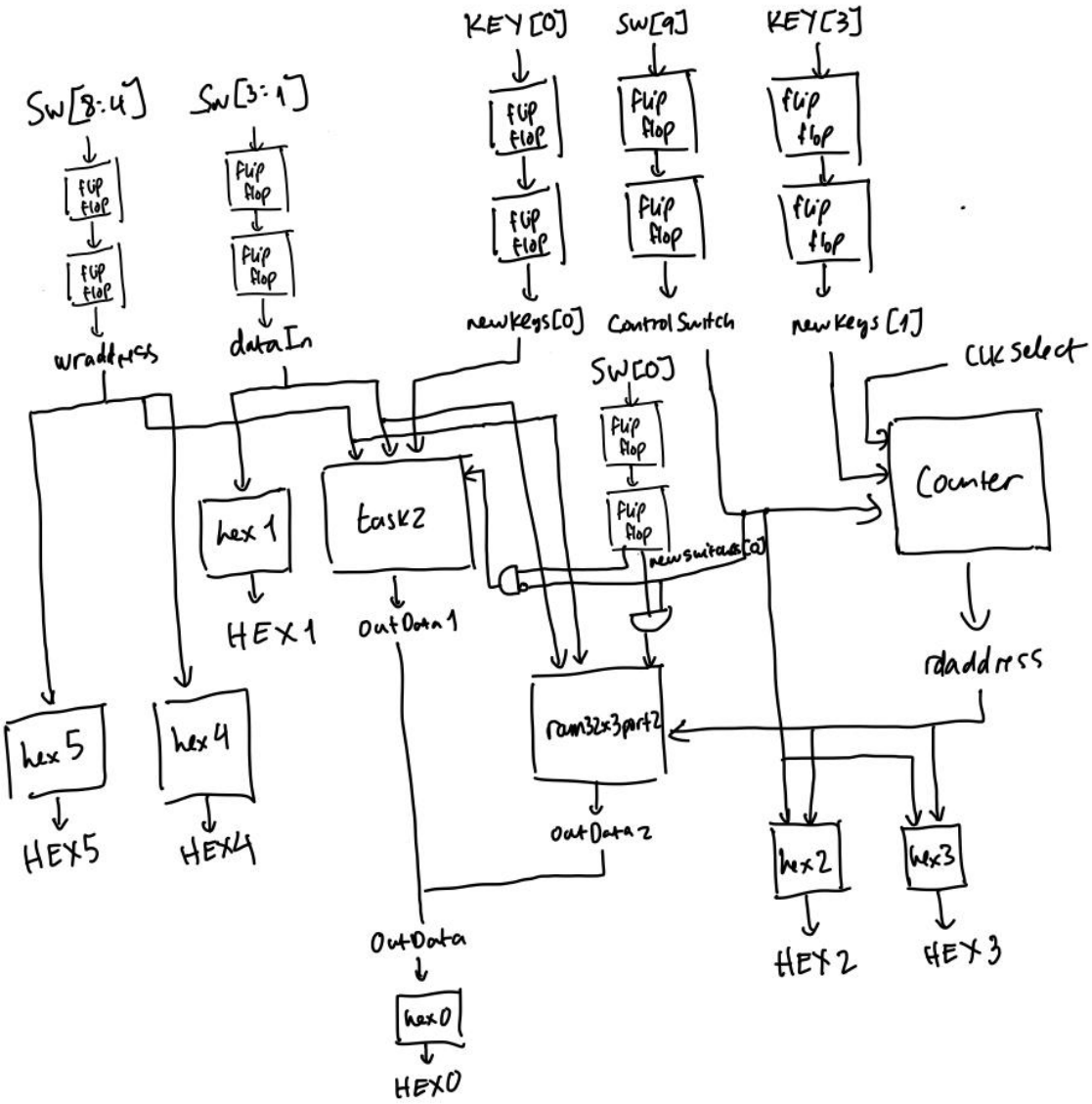
Task #2

For task 2, we are asked to create the RAM memory module without using the prebuilt modules provided by Quartus libraries. We would implement the memory unit through SystemVerilog as a 32 x 3 multidimensional array structure with size: 32 words x 3 bits per word. To implement this design, we would have to create the memory module ourselves. Therefore we created task2mem.sv that would act as the RAM memory unit called into our main task2.sv module. Our main module is coded and behaves identical to task 1. In our task 2 memory module, we created the RAM structure through “logic [2:0] memory_array [31:0]”. We used an if statement such that if write is enabled, we write the input “data” to the given address input. For both cases when write is enabled or disabled, we read the data from the address input and assign it to output q. Everything is connected to the clockedge by an always_ff statement and thus everything occurs on the positive clockedge.

We then created a DE1_SoC module which acted as the top-level module that would port in the task2 RAM module. The switches mentioned in “Design Procedure” are used to simulate dataIn, address and write inputs and KEY0 acts as the clock input. We used the seg7.sv file to display the HEXs accordingly on the board. (The DE1_SoC KEY signals are active-low so it is pressed when given a value of 0.

DE1_SoC Overall System

DE1_SoC is the top level module used in task 3. The module first instantiates a clock divider module (the code for that is from EE 271) and allows us to toggle between two different clocks (one for ModelSim and one for the board). Then we pass the KEY inputs and the Switch inputs through double flip-flops to deal with metastability. We also create a counter inside the module that increments every clock edge. With these refined inputs we can now instantiate the task 2 and 2 port RAM modules with those values. After that, our task is to display the appropriate values on the HEX displays. We created a new module seg7_reset that had a reset logic that turned the HEX off, for HEX2 and HEX3 (since they had to be toggled off) and used the given seg7 module to display the other HEX values.


KEY[0]
SW[9]
KEY[3]
SW[8:4]
SW[3:1]
flip flop
wraddress
dataIn
newkeys[0]
Control Switch
newkeys[1]
SW[0]
CLK select
Counter
hex1
task2
HEX1
outData1
rdaddress
hex5
hex4
ram32x3part2
HEX5
HEX4
outData2
hex2
hex3
HEX2
HEX3
OutData
hex0
HEX0

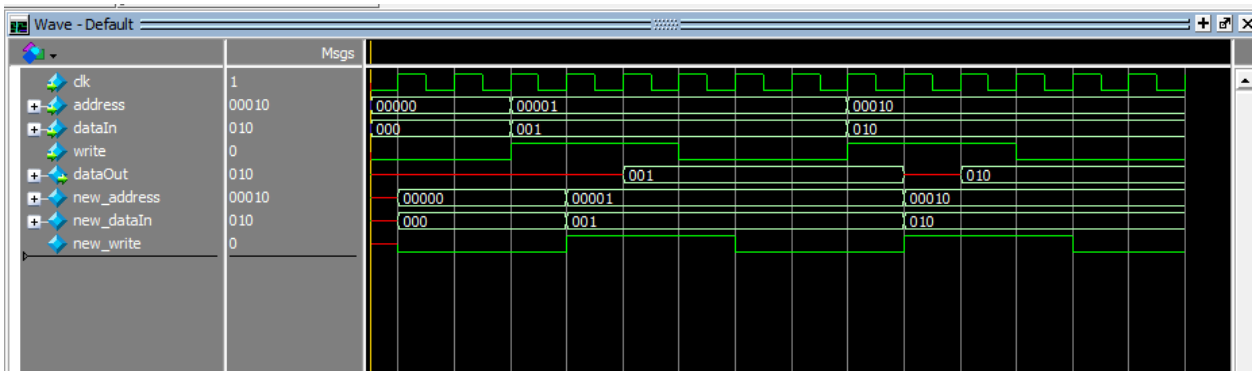
Results

The ModelSim waveforms for each major module are presented and explained here.

Task 1 Model Sim

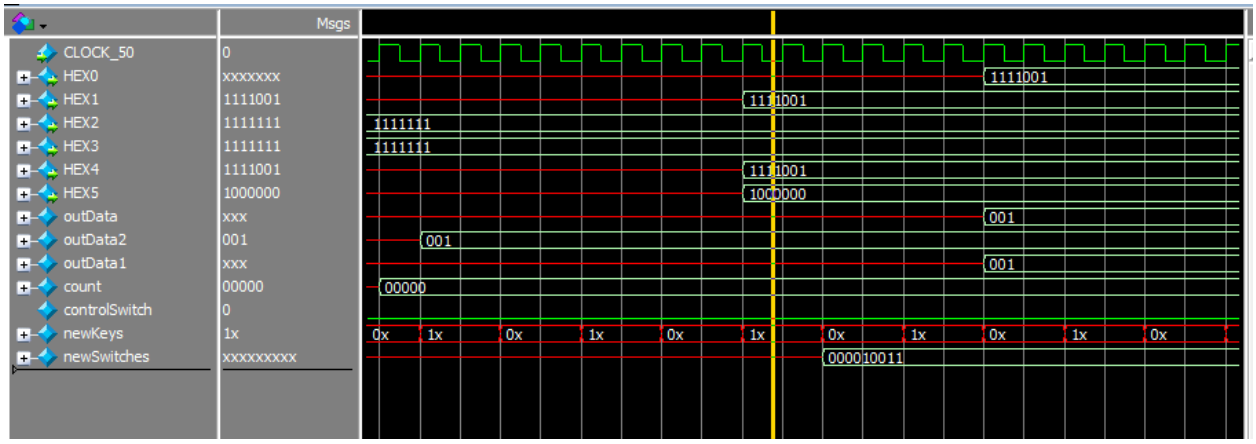
Our simulation of Task 1 is designed to show the output result “dataOut” of the RAM memory unit provided the inputs “new_address”, “new_dataIn” and “new_write” (the “new” part indicates that they have passed through the register flip flops). From the simulation, we can see that when write is enabled, the memory at the address specified and dataOut will match dataIn and the memory continues to stay at the same value until it is rewritten. When write is disabled, dataOut represents the current data at the address and is not affected by dataIn.

Task 2 Model Sim

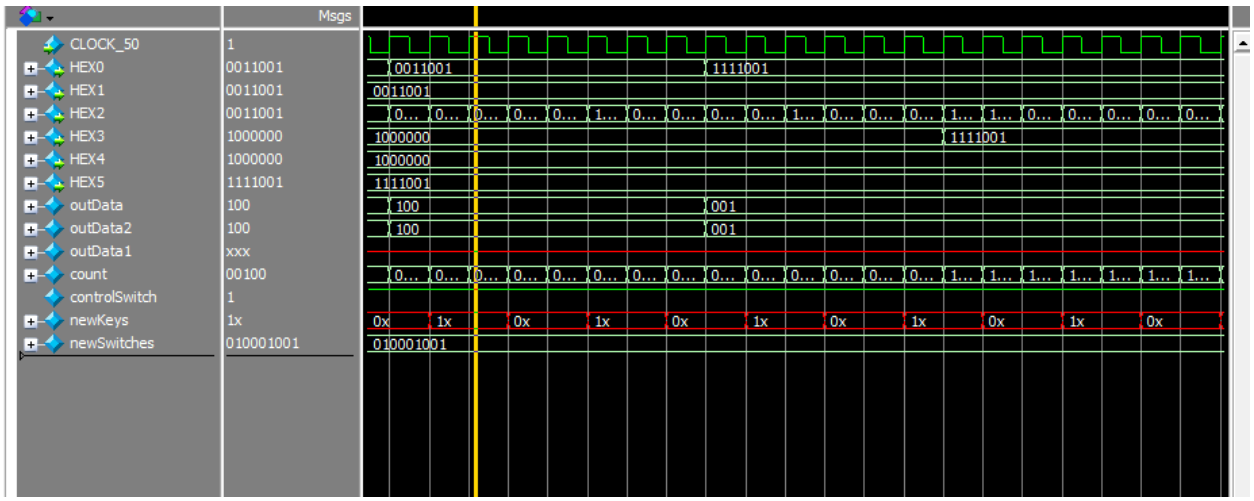


Our simulation of Task 2 is identical to task 1as we test our inputs to check for the right output. As we can see, we receive the same results as task 1 which is evidence that the SystemVerilog RAM module works accordingly.

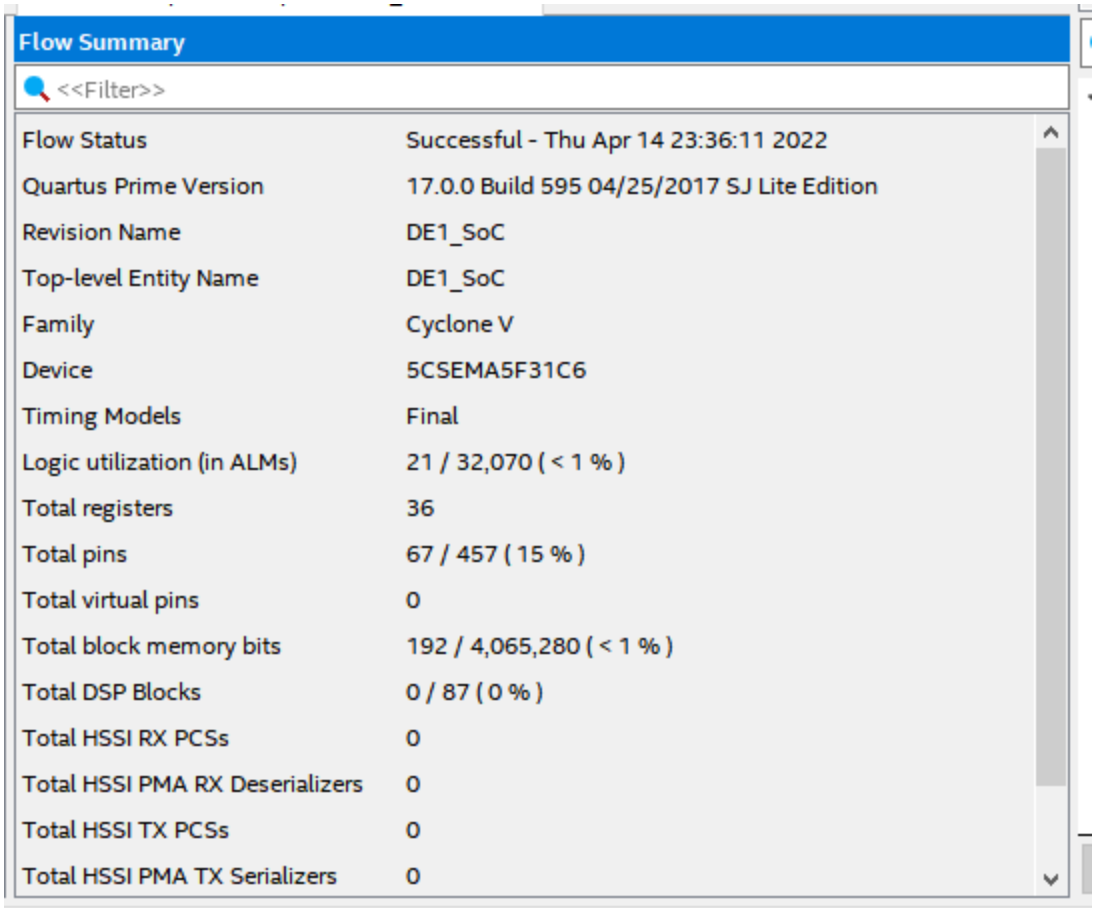
DE1_SOC Model Sim



The DE1_SoC testbench tested reading and writing to various adresses first with SW[9] off and then with SW[9] on. The figure above shows the task 2 part of the DE1_SoC with the SW[9] being off. We can see that the HEX's update according to the adresses and data. The following figure shows the task 3 part with the two port memory, where we have the SW[9] being on. From that we see similar behavior, however, the read adress is now a counter and HEX2 and HEX3 are now displaying data.



Flow Summary



Flow Summary

<<Filter>>

Flow Status	Successful - Thu Apr 14 23:36:11 2022
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	21 / 32,070 (< 1 %)
Total registers	36
Total pins	67 / 457 (15 %)
Total virtual pins	0
Total block memory bits	192 / 4,065,280 (< 1 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0

Experience Report

We found this lab to be more straightforward than the previous one. The specification did a good job in guiding us through the tasks and providing the relevant information required to complete them. It took us a while to figure out how to make sure that memories from Task 2 and Task 3 were independent from each other. We ended up using cases to resolve this. What took the longest time on task 3 was creating the counter and implementing KEY3 as the reset signal. We initially tried to reuse a counter form EE 271, however this did not work as the counter was too specific to the 271 lab. We tried changing parts of it here and there but ultimately it did not work and we had to create our own. Implementing KEY3 to reset also took a while as we were unsure of what to reset to since it was not specified. Therefore we eventually came to the conclusion that the reset button was for the counter and adjusted the code accordingly. The testbenches for the tasks also proved to be a little confusing to understand. What

we then did was write out all our inputs and outputs in a sequential manner so that we could clearly see what should be happening at each clock edge.

The Lab took us a total of 15 hours to complete:

- Reading: 30 minutes
- Planning: 30 minutes
- Design: 1.5 hours
- Coding: 5 hours
- Testing: 2.5 hours
- Debugging: 5 hours