EE 371 Lab4 Report: Binary Searcher and Bit Counter

Ziyi Huang, Shu Xu

ECE, Embedded System

University of Washington

May 7, 2019

Introduction

In this lab, we will use ASM chart to help implement algorithms in logic circuits. There are two tasks in this lab.

In task 1, we will use the given ASM chart to design a bit-counting circuit, which can count the number of 1s being input up to 8 bit.

In task 2, we will implement binary search algorithm in Verilog. The circuit is expected to search through an 32x8 RAM and locate an 8-bit value being input. Here we assign the memory with sorted values, and implement the comparison between the middle element and the given value to locate faster.

The rest of the report will describe the procedures, results, and the problems faced/feedbacks in sections, repsectively.

Procedure

Task 1

In this task, we implemented a bit counter that counted the number of 1s in the binary format of the given number. Figure 1 demonstrates its ASM chart. For each registered value besides the "done" in the square block, we created a temporary register and used it in a combinational logic to hold its next-state value. This implementation allowed us to synchronously update those values at a clock edge. On the contrary, the "done" update was delayed by 1 cycle due to the lack of a next-state register.

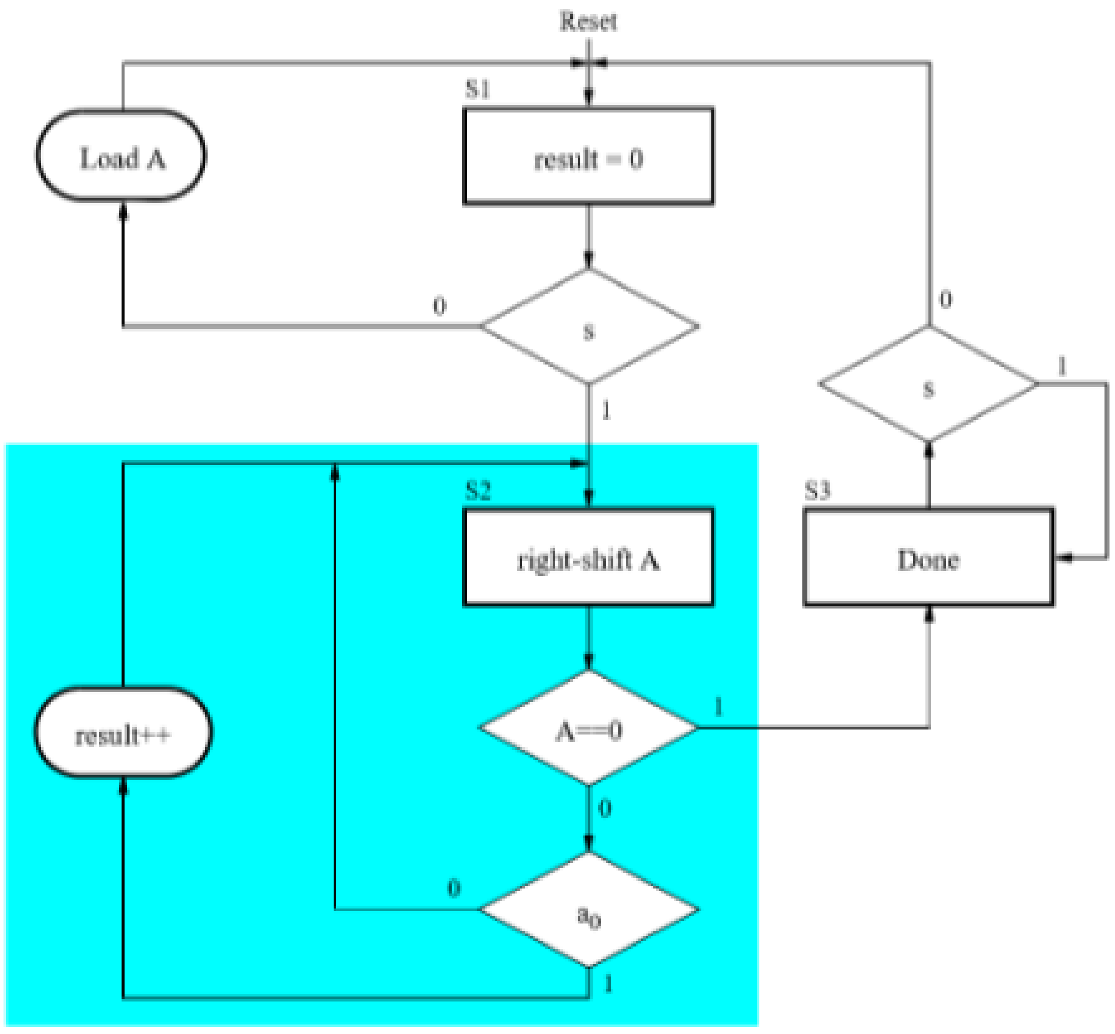


Figure 1: the ASM chart of the bit counter

Task 2

In this task, we implemented the binary search algorithm in VHDL. To do that, we designed a ASMD diagram, used a block diagram to break down the modules, and then implemented the system accordingly. The ASMD chart is shown in Figure 2, and the control & datapath block diagram is shown in Figure 3.

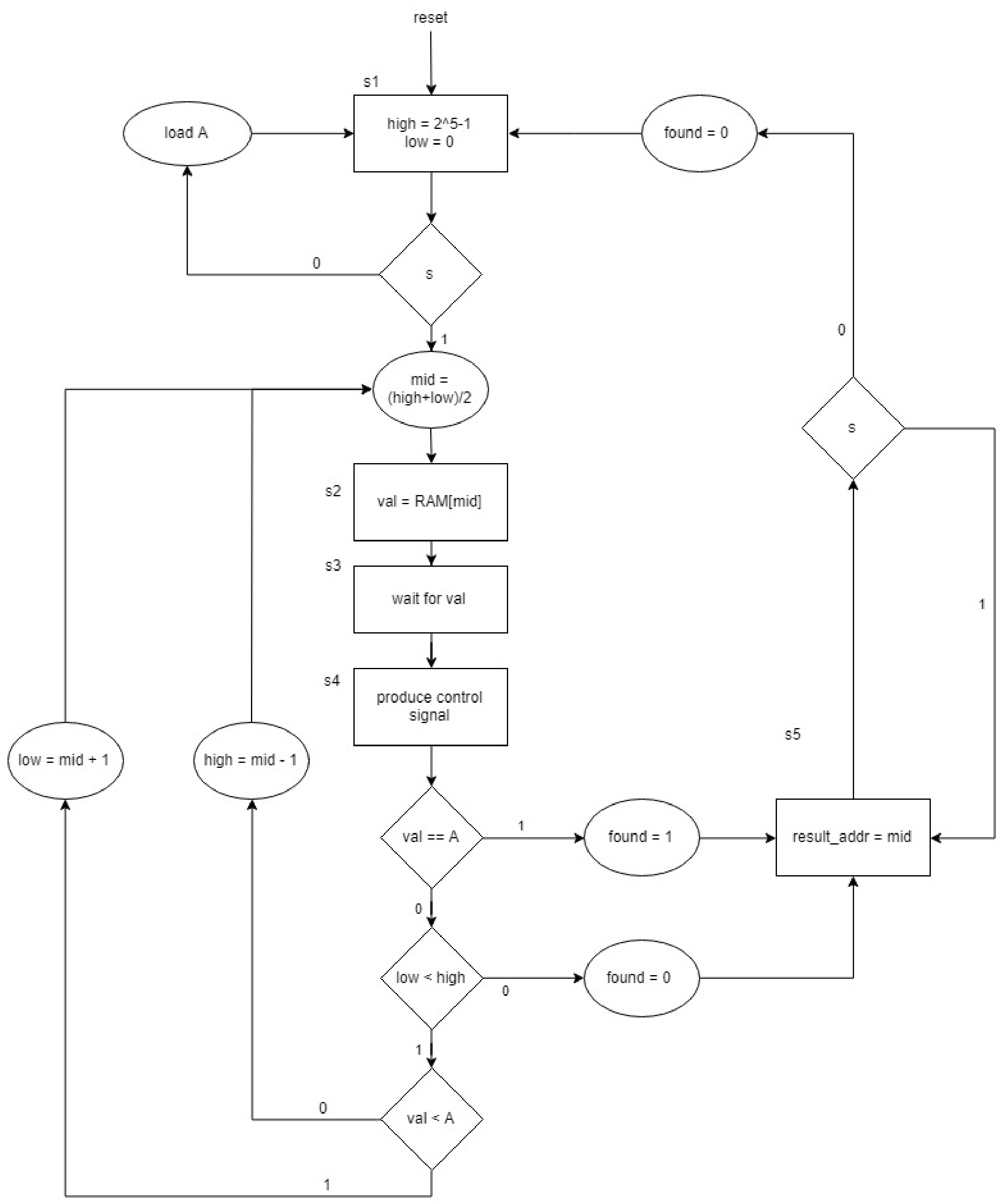


Figure 2: the ASM chart of the binary search

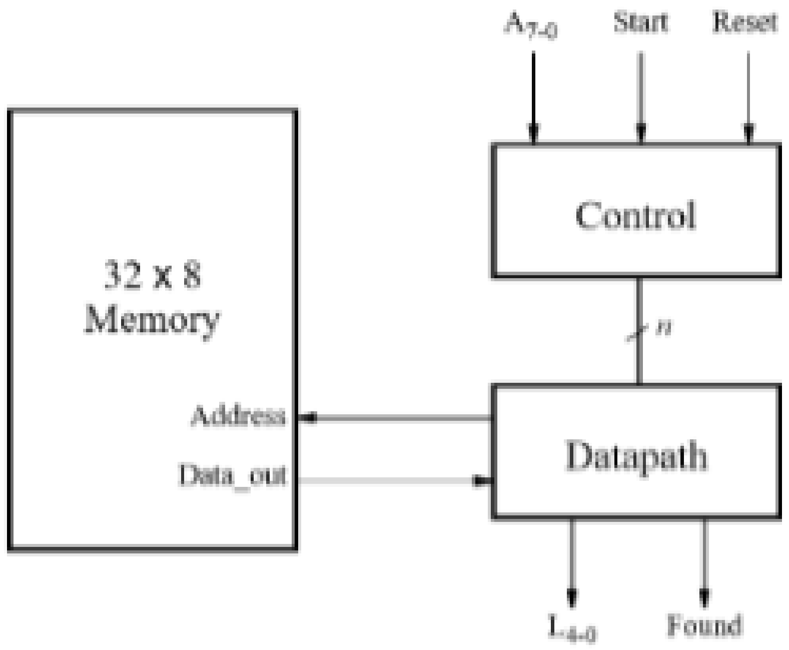


Figure 3: blocks for binary search

As shown in Figure 2 and Figure 3, we will have two modules for control and datapath, and we design five states, from s1 to s5, for the binary search module.s1 is the stage where everything starts here. When start signal is true, next state becomes s2, where we update the mid element value of the memory array. Then in s3, we wait for the value to be grabbed from the RAM. The data becomes available in s4, and thus we compare that value with the target as well as the indices to return signals to the control logic, who will determine the next actions accordingly . Next state after s4 depends on whether the circuit locates the given input value or not. If located, next state becomes s5 where we output a "found" signal and the result address if found or output only a "not found" signal. If not, next state go back to s2 to update new control signals.

Results

After designing the system, we construct testbenches for simulation.

Task 1

The block diagram of task1 bit counter design is shown as Figure 4

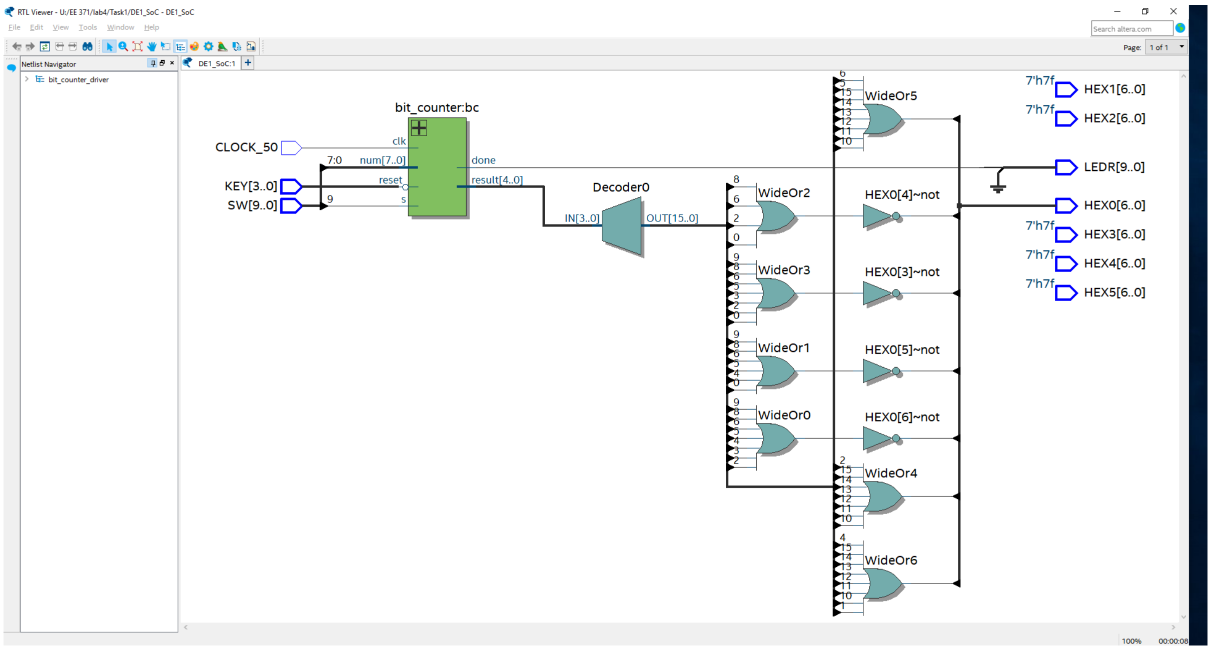


Figure 4: task1 bit counter block diagram

We also design testbences for the bit counter module Figure 5 and top level module Figure 6.

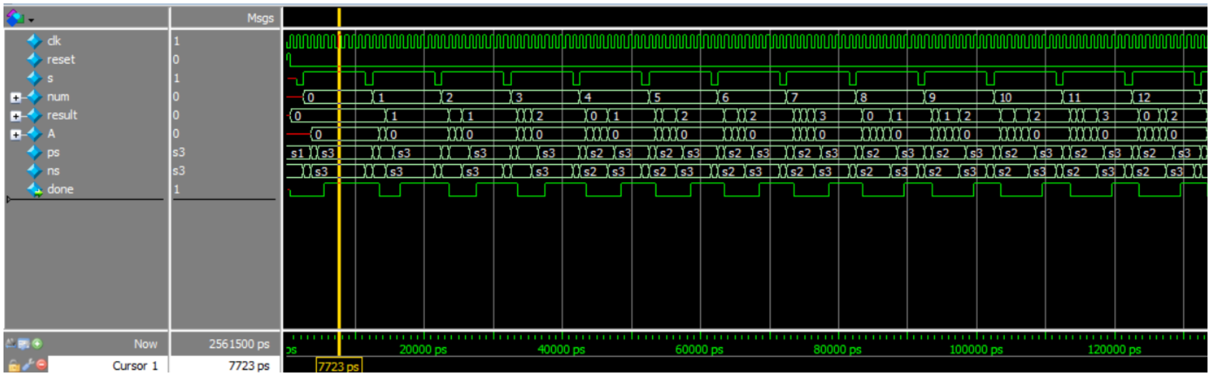


Figure 5: bit counter module simulation

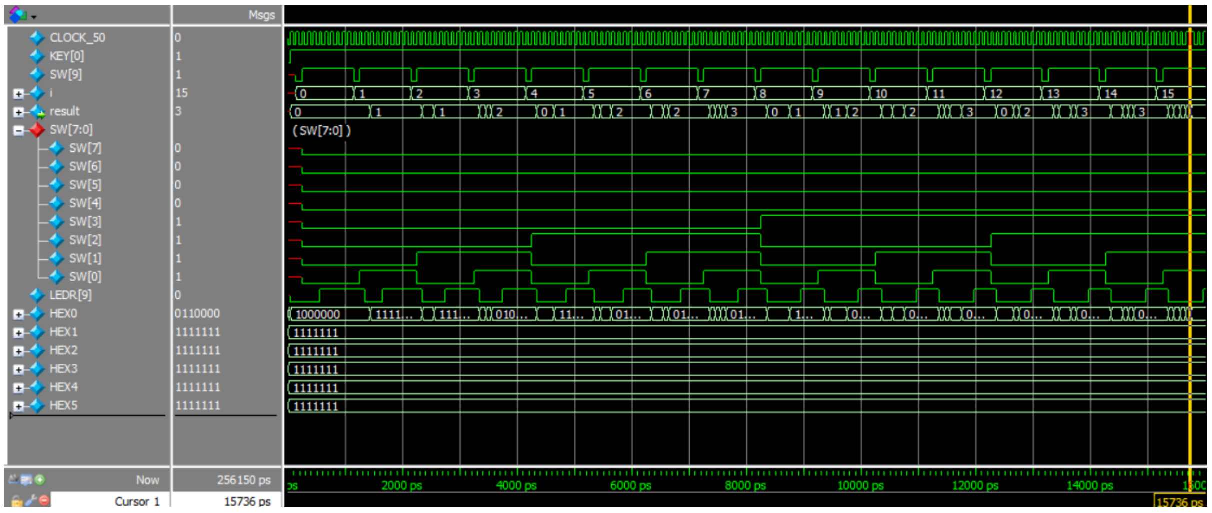
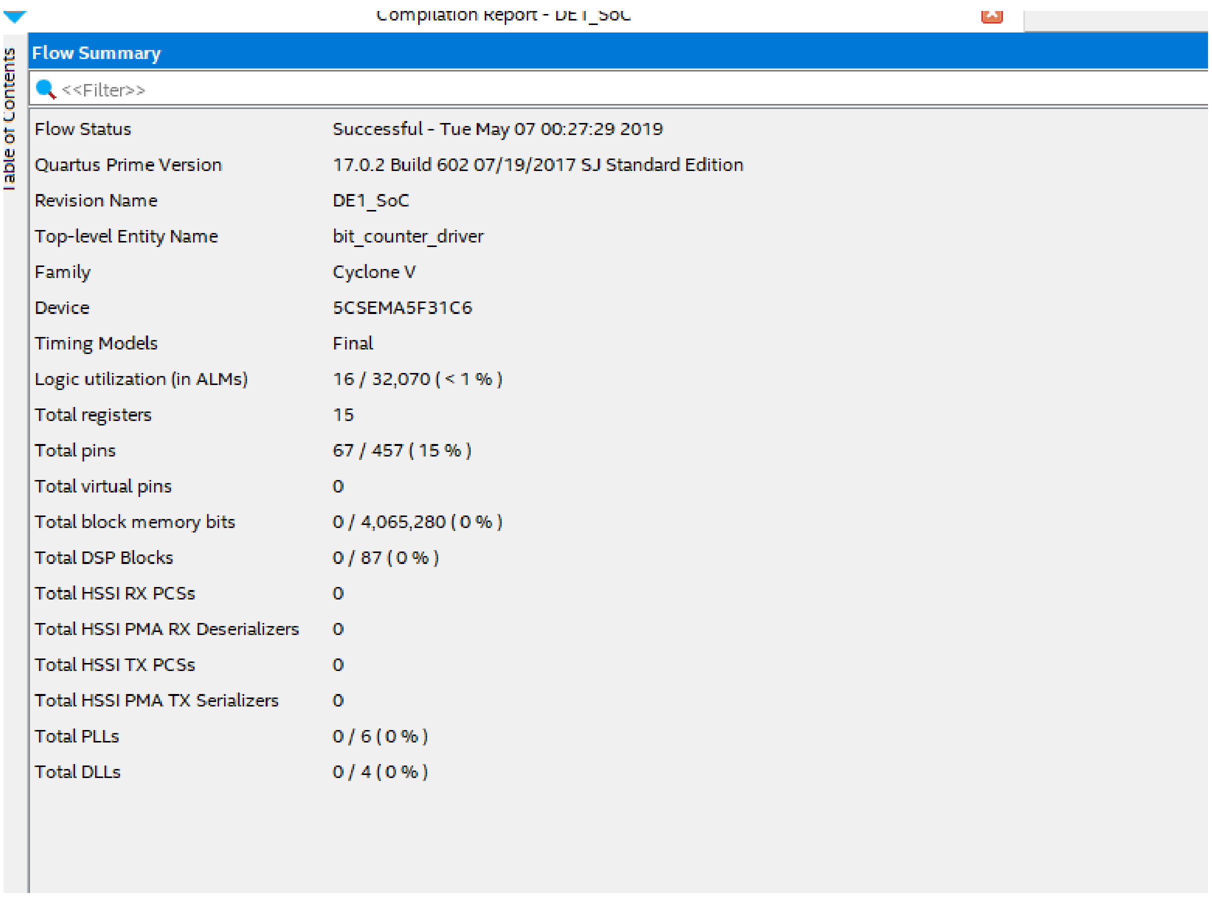


Figure 6: bit counter top level module simulation

The flow summary of task 1 design is shown as Figure 7.



Flow Status	Successful - Tue May 07 00:27:29 2019
Quartus Prime Version	17.0.2 Build 602 07/19/2017 SJ Standard Edition
Revision Name	DE1_SoC
Top-level Entity Name	bit_counter_driver
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	16 / 32,070 (< 1 %)
Total registers	15
Total pins	67 / 457 (15 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
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
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 7: task1 design flow summary

The results of all testbenches agree with the expected values.

Task 2

The block diagram of task2 binary search design is shown as Figure 8

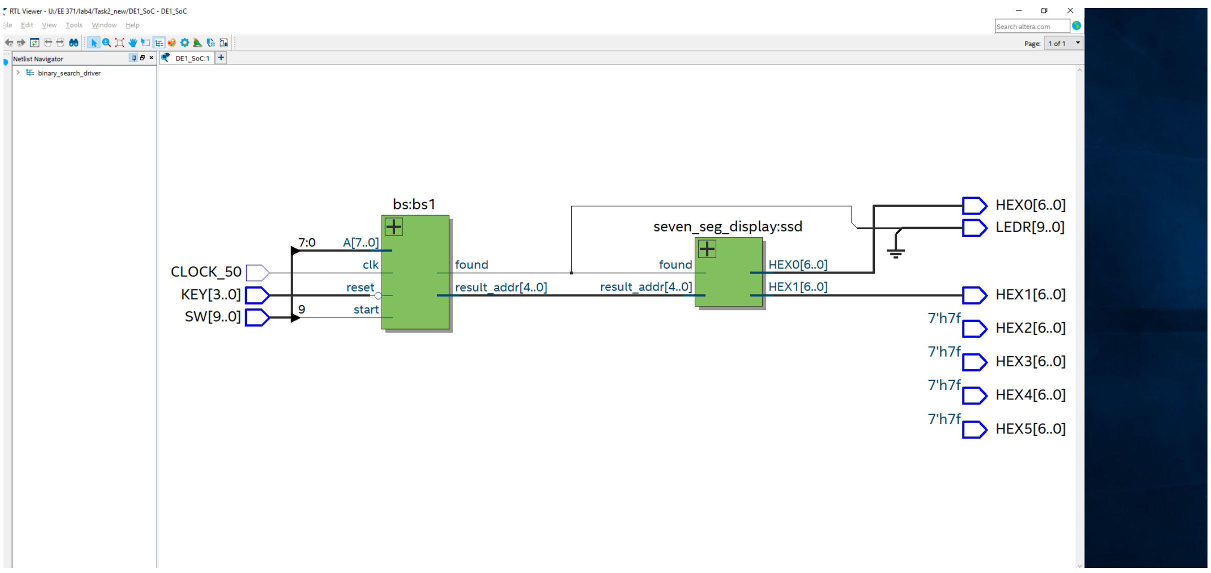


Figure 8: task2 block diagram

We design testbech for the RAM module Figure 9

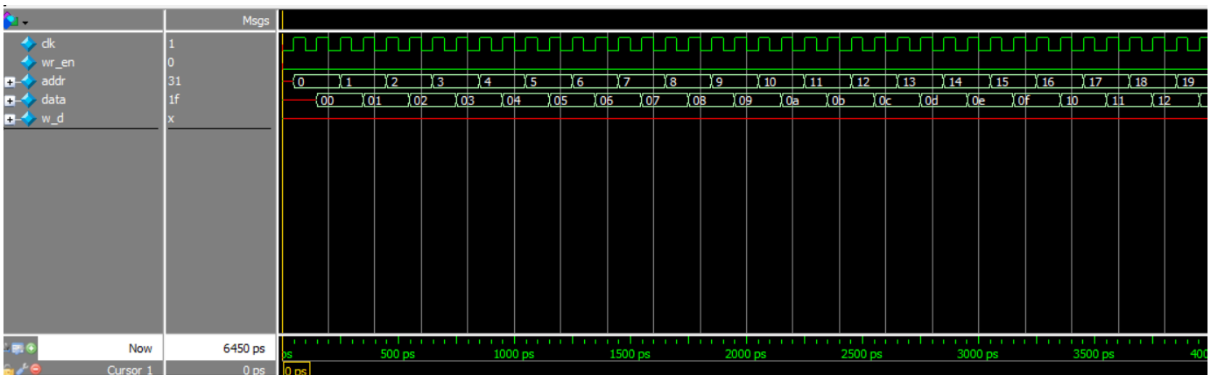


Figure 9: task2 RAM simulation

We also design testbences for the binary search module Figure 10 and top level module Figure 11.

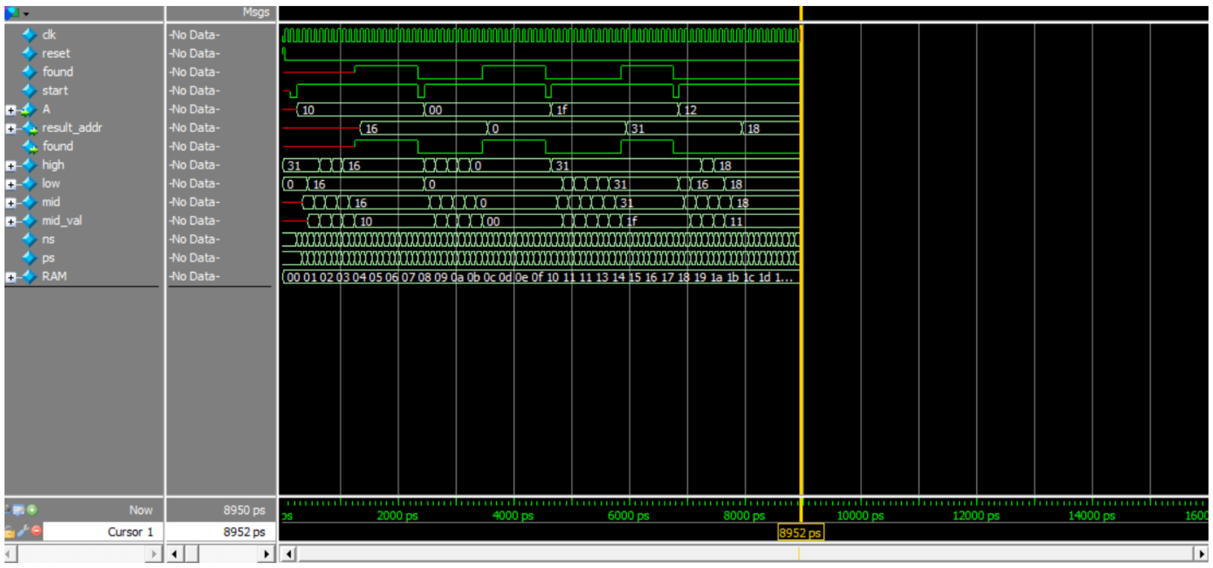


Figure 10: binary search module simulation

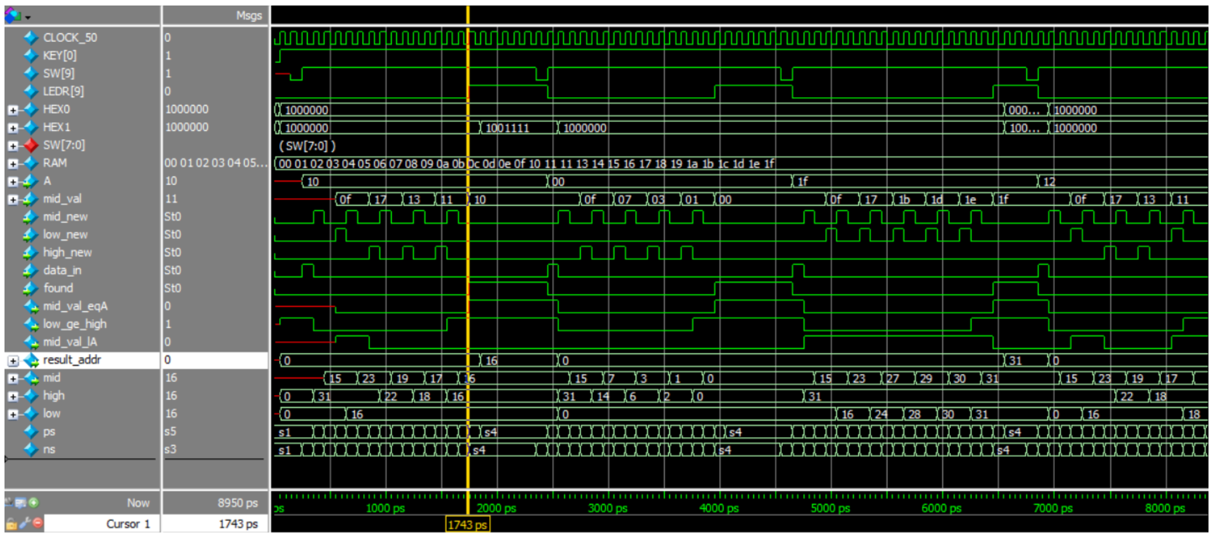
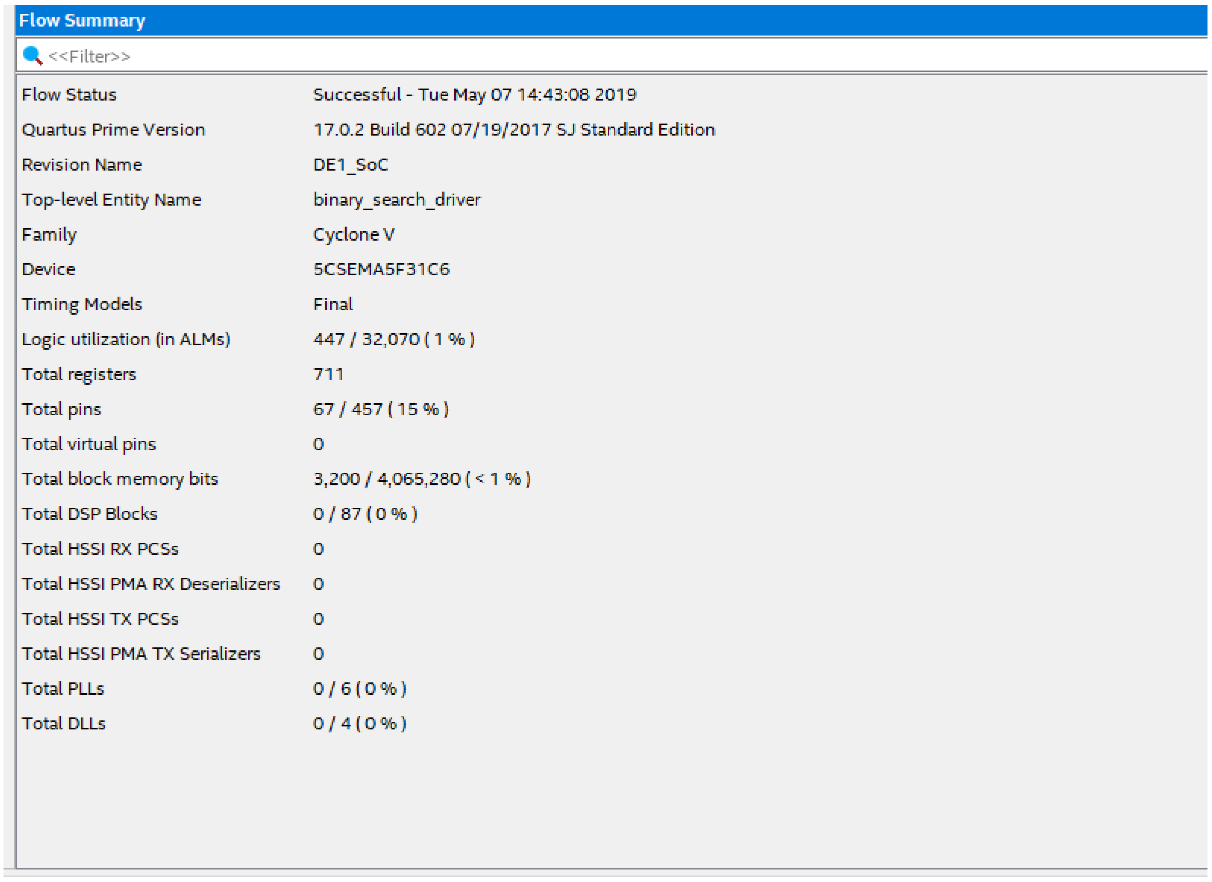


Figure 11: binary search top level module simulation

The flow summary of task 2 design is shown as Figure 12.



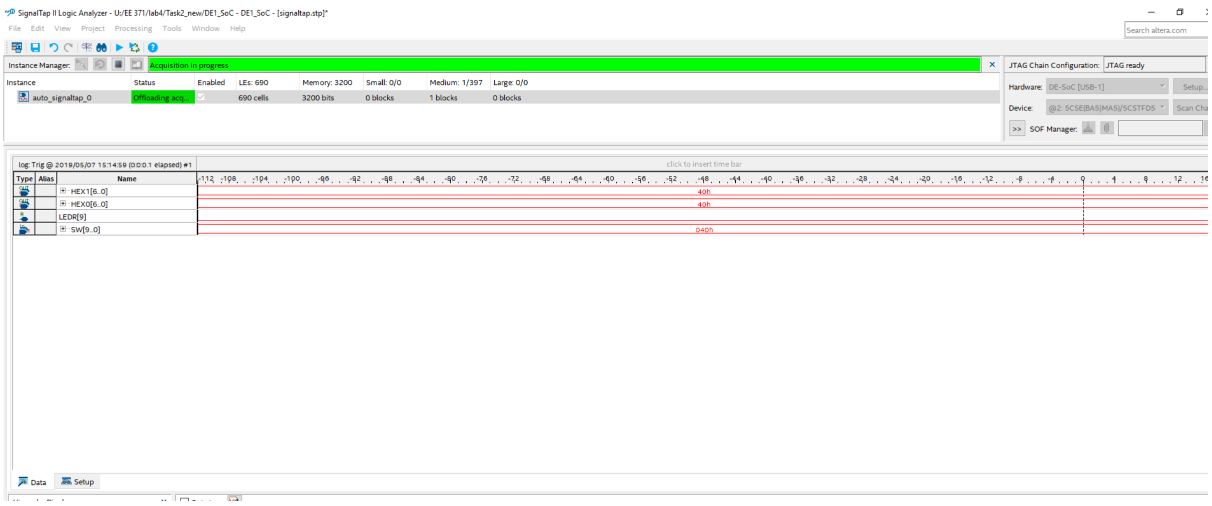
Flow Summary

<<Filter>>

Flow Status	Successful - Tue May 07 14:43:08 2019
Quartus Prime Version	17.0.2 Build 602 07/19/2017 SJ Standard Edition
Revision Name	DE1_SoC
Top-level Entity Name	binary_search_driver
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	447 / 32,070 (1 %)
Total registers	711
Total pins	67 / 457 (15 %)
Total virtual pins	0
Total block memory bits	3,200 / 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
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 12: task2 design flow summary

The signalTap results is shown as Figure 13



SignalTap II Logic Analyzer - U:/EE 371/lab4/Task2_new/DE1_SoC - DE1_SoC - [signaltap.stp]*

Instance Manager: Acquisition in progress

Instance	Status	Enabled	LEs: 690	Memory: 3200	Small: 0/0	Medium: 1/397	Large: 0/0
auto_signaltap_0	Offloading acq...	✓	690 cells	3200 bits	0 blocks	1 blocks	0 blocks

JTAG Chain Configuration: JTAG ready

Hardware: DE-SoC [USB-1]

Device: @2: 5CSE(BA5|MA5)/5CSTFD5

SOF Manager:

log: Trig @ 2019/05/07 15:14:59 (0:0:0.1 elapsed) #1

Type	Alias	Name	
		HEX1[6..0]	40h
		HEX0[6..0]	40h
		LEDR[9]	
		SW[9..0]	040h

Figure 13: task2 signalTap

The results of all testbenches agree with the expected values.

Problem Faced & Feedback

Approximatley 10 hrs were spent on this lab, consisting of 40% designing the ASMD chart and test cases, 10% coding and simulating, 50% debugging for the hardware.

The hardest part of task 1 is to synchronize the registers update, which was achieved with next-state registers.

The hardest part of task 2 is to solve the timing problem properly for the control module, the datapath module, and the RAM. We had to add two states to stall the system and wait until the data value is updated before new control signal is updated.