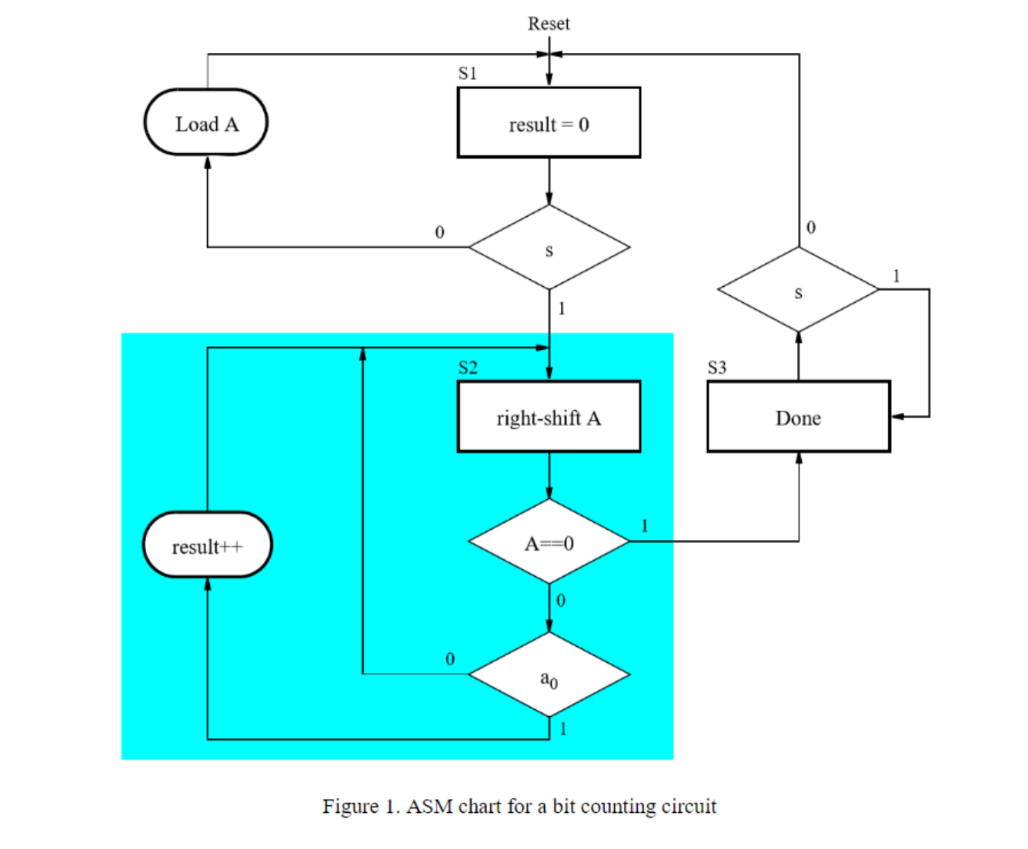
Abstract

In this lab we use algorithmic state machine charts to implement algorithms as hardware circuits. We also learned the key distinction between ASM charts and flow charts. We implemented bit-counting circuit and binary search algorithm by the concept of ASM and ASMD.

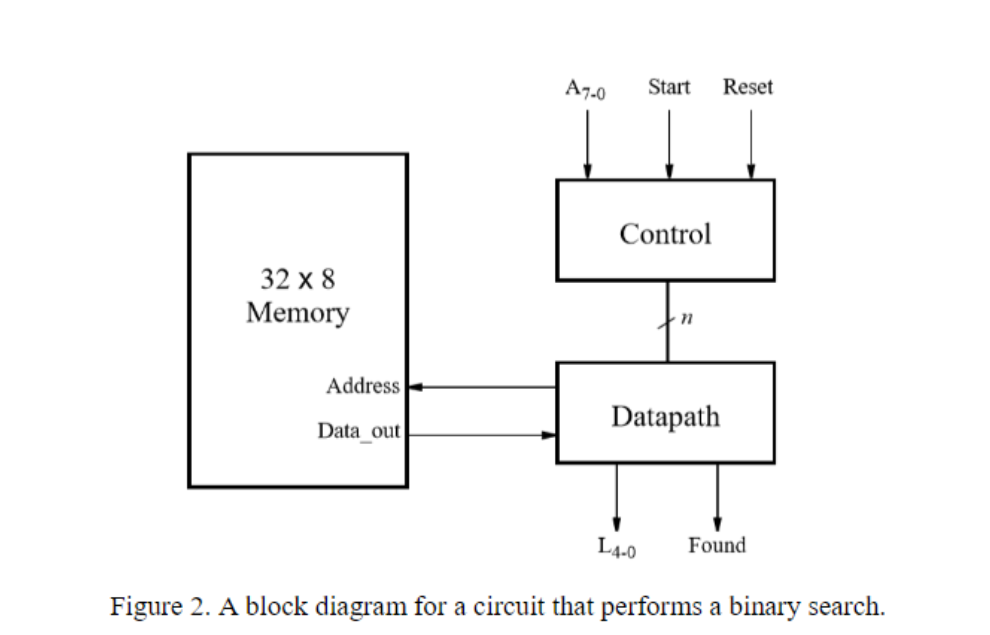
Introduction

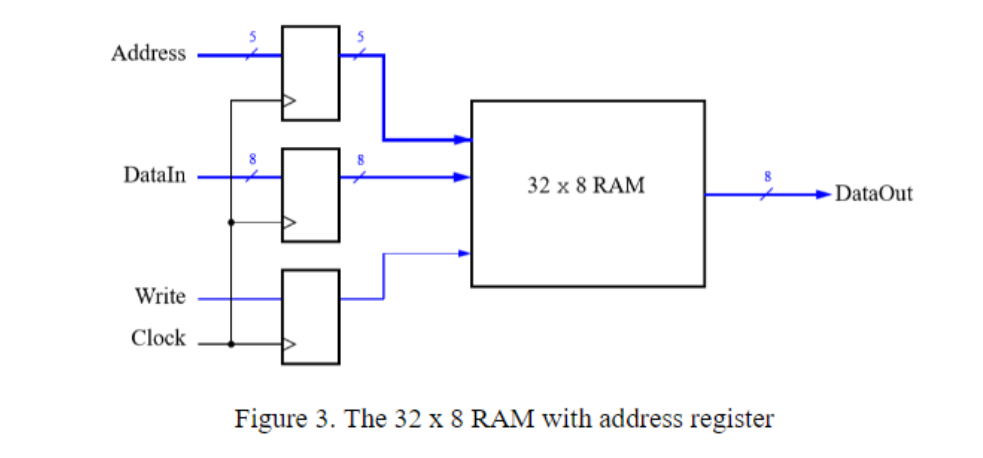
Algorithmic State Machine (ASM) charts are a design tool that allow the specification of digital systems in a form similar to a flow chart. An example of an ASM chart is shown in Figure 1. It represents a circuit that counts the number of bits set to 1 in an n-bit input A (A=an-1 an-2…a1 a0). The rectangular boxes in this diagram represent the states of the digital system, and actions specified inside of a state box occur on each active clock edge in this state. Transitions between states are specified by arrows. The diamonds in the ASM chart represent conditional tests, and the ovals represent actions taken only if the corresponding conditions are either true (on an arrow labeled 1) or false (on an arrow labeled 0).



In this ASM chart, state S1 is the initial state. In this state the *result* is initialized to 0, and data is loaded into a register A, until a start signal, s, is asserted. The ASM chart then transitions to state S2, where it increments the result to count the number of 1’s in register A. Since state S2 specifies a shifting operation, then A should be implemented as a shift register. Also, since the result is incremented, then this variable should be implemented as a counter. When register A contains 0 the ASM chart transitions to state S3, where it sets an output Done=1 and waits for the signal s to be deasserted. A key distinction between ASM charts and flow charts is a concept known as implied timing. The implied timing specifies that all actions associated with a given state take place only when the system is in that state when an active clock edge occurs. For example, when the system is in state S1 and the start signal s becomes 1, then the next active clock edge performs the following actions: initializes result to 0, and transitions to state S2. The action right-shift A does not happen yet, because the system is not yet in state S2. For each active clock cycle in state S2, the actions highlighted in Figure 1 take place, as follows: increment result if bit a0=1, change to stateS3 if A=0 (or else remain in state S2), and shift A to the right. The implementation of the bit counting circuit includes the counter to store the result and the shift register A, as well as a finite state machine. The FSM is often referred to as the control circuit, and the other components as the data path circuit. We will implement this circuit in Task 1.

In task 2 we will implement Binary Search. The binary search algorithm works on a sorted array. Rather than comparing each value in the array to the one being sought, we first look at the middle element and compare the sought value to the middle element. If the middle element has a greater value, then we know that the element we seek must be in the first half of the array. Otherwise, the value we seek must be in the other half of the array. By applying this approach recursively, we can locate the sought element in only a few steps. In this circuit, the array is stored in a memory module that is implemented inside the FPGA chip. A diagram of the memory module that we need to create is depicted in Figure 3. In a similar fashion to the first task of lab 2, create a memory that is eight-bits wide and 32 words deep.





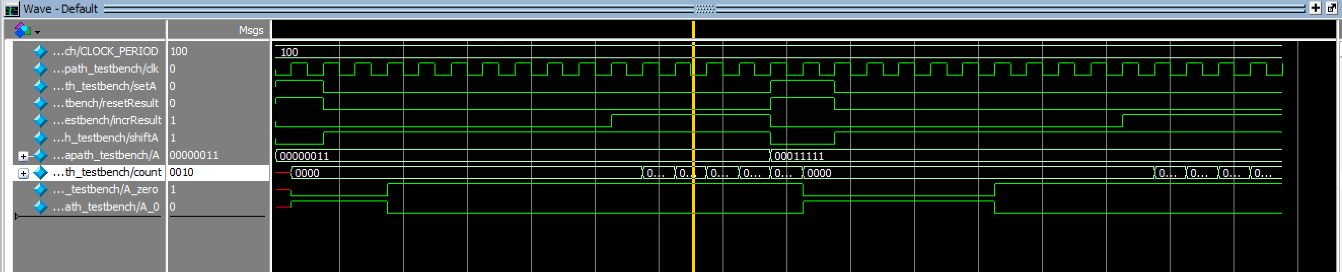
To place data into the memory, initialize the memory using the contents of a memory initialization file (MIF)and call it my\_array.mif, which then has to be created in the folder that contains the Quartus project. Set the contents of your MIF file such that it contains a sorted collection of integers. Your circuit should produce a 5-bit output L, which specifies the address in the memory where the number A is located. In addition, a signal Found should be set high to indicate that the number A was found in the memory, and set low otherwise.

Procedures

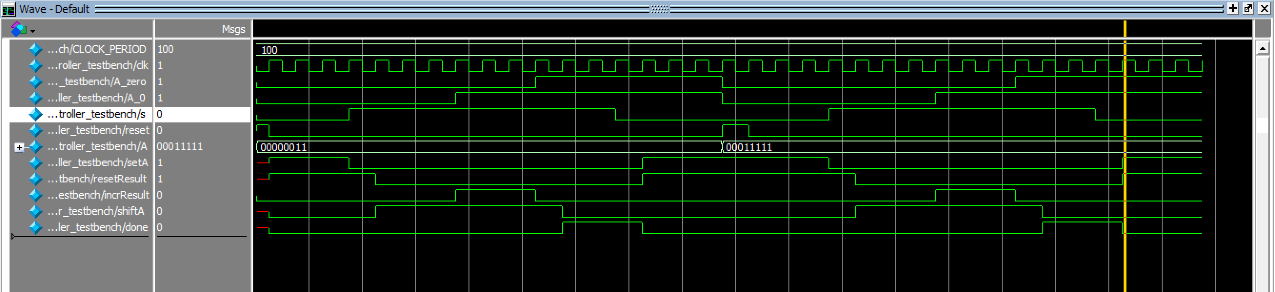
* Task 1
  + Wrote System Verilog code to implement bit-counting circuit using the ASM chart sown in Figure 1.
  + Connected the bit-counting circuit to the DE1-SoC board with the inputs of 8 bit data through SW[7:0], a synchronous reset connected to KEY[0] and a start switch connected to SW[9].
  + Wrote a test bench to test all types of input and saw desired output at HEX0 and LEDR[9].
* Task 2
  + Created the new module (binarySearch.sv) and connected it to the switches and used hex display to display value to be found, and found signal.
  + Created my\_array.mif.
  + Wrote a test bench for the binary Search and DE1\_SoC to test all types of input and saw desired output.
  + Used the Signal Tap II functionality of Quartus to verify the contents of my RAM module.

Results

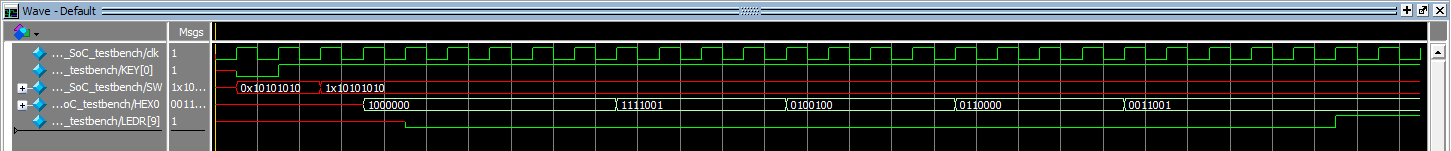
* Task 1:
  + datapath waveform

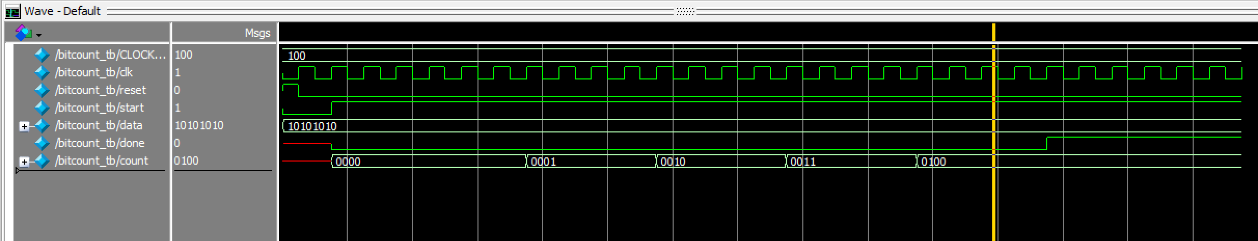


* + controller waveform

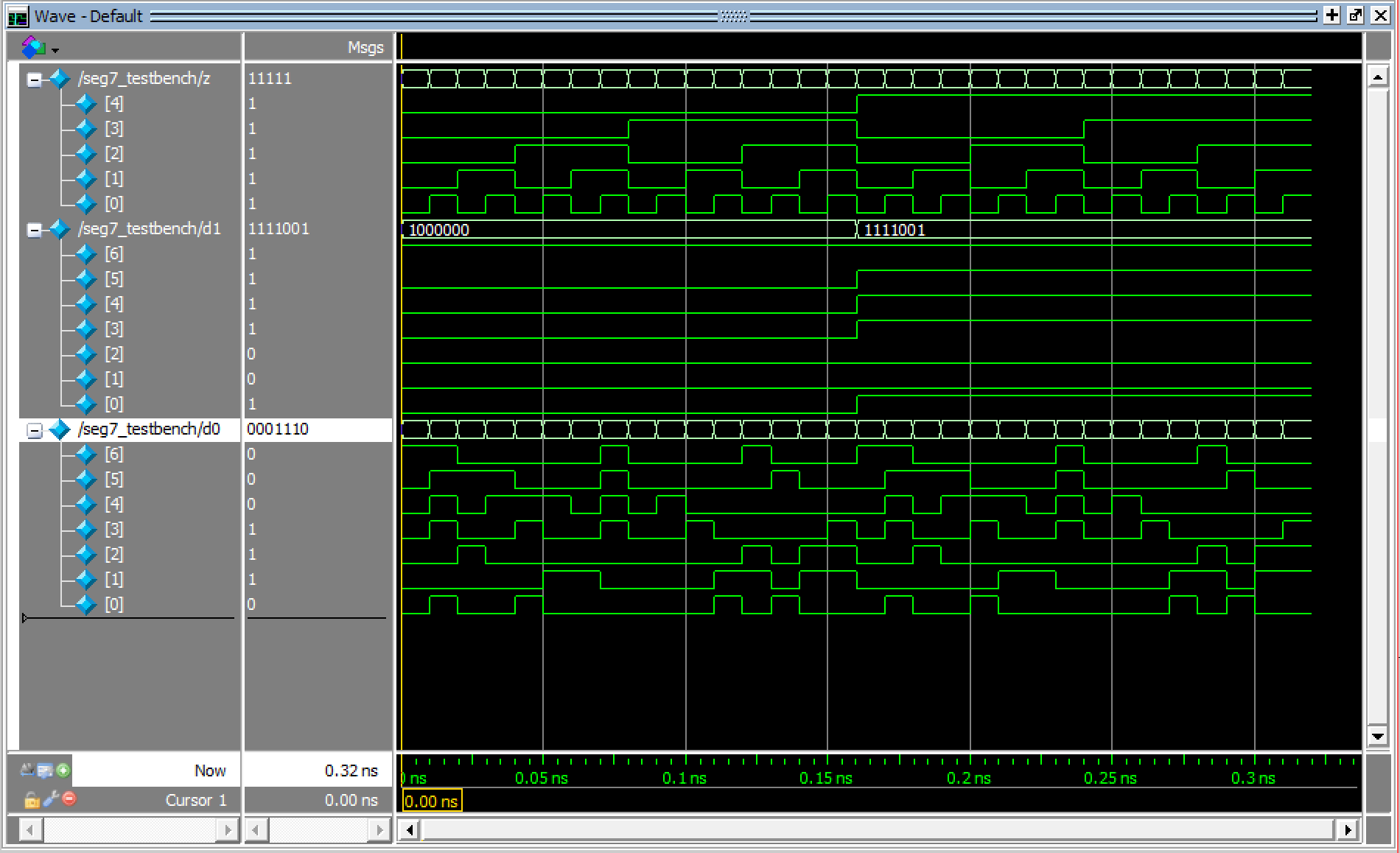


* + DE1-SoC\_testbench waveform

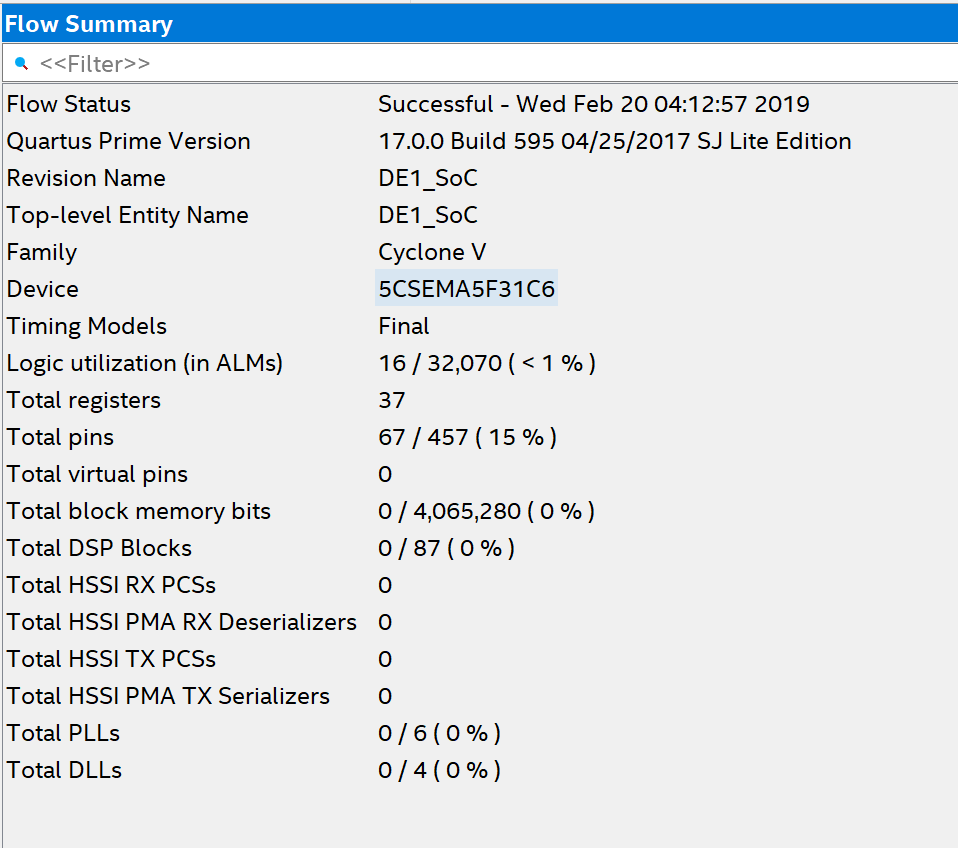




* + seg7\_testbench waveform

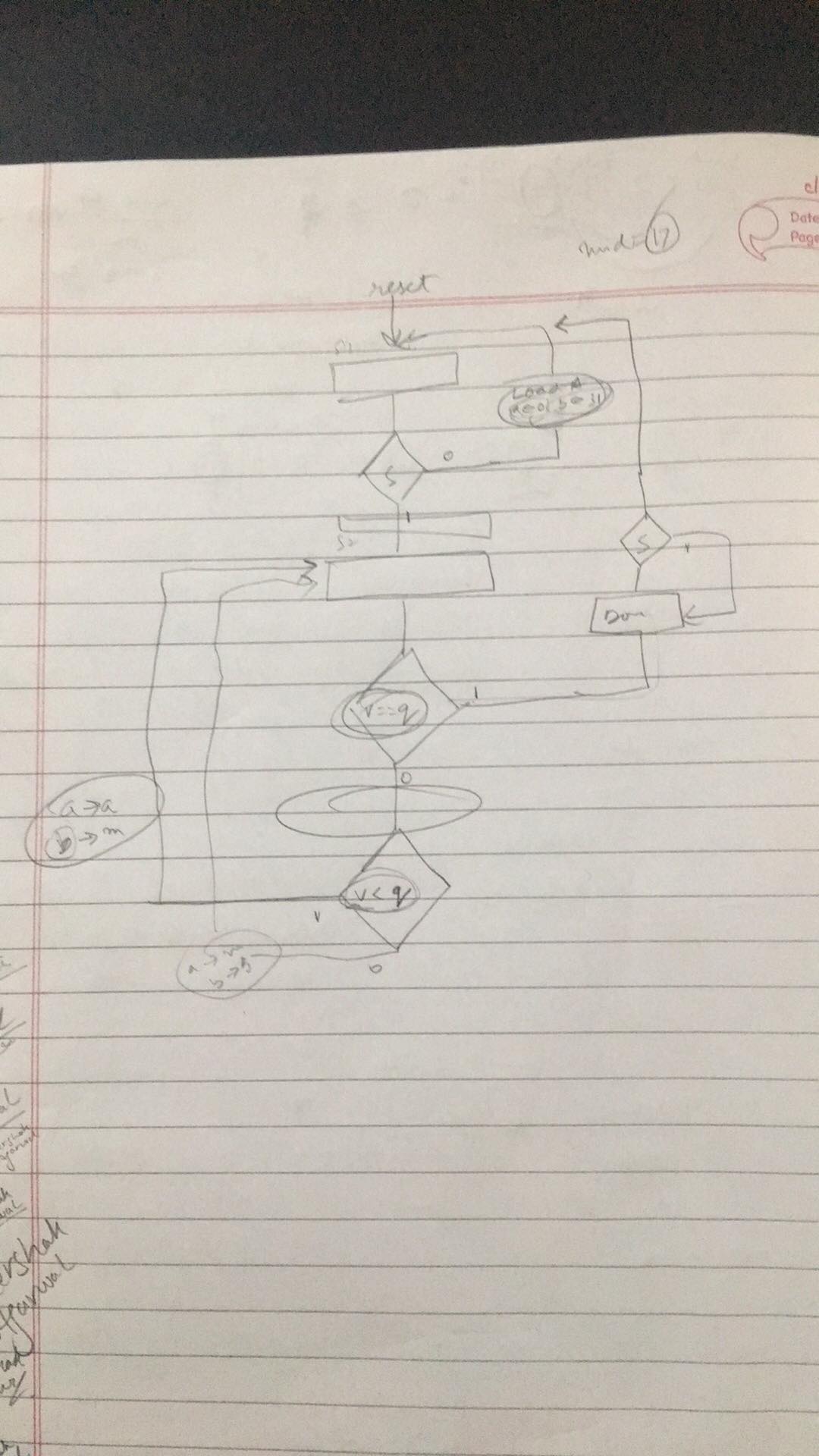


* + Task Flow Summary.

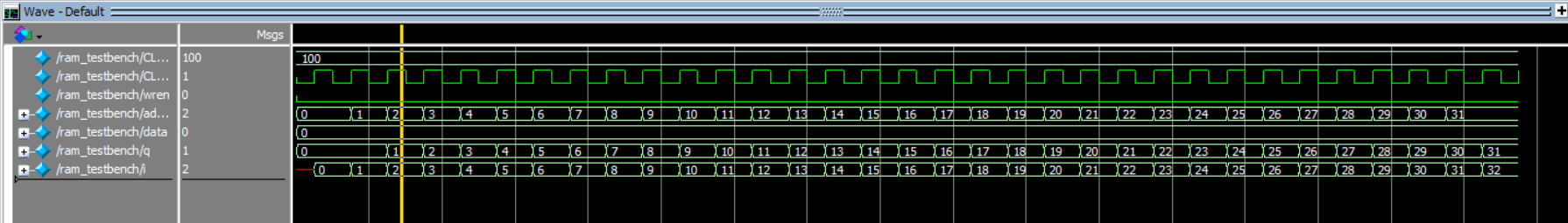


* Task 2:

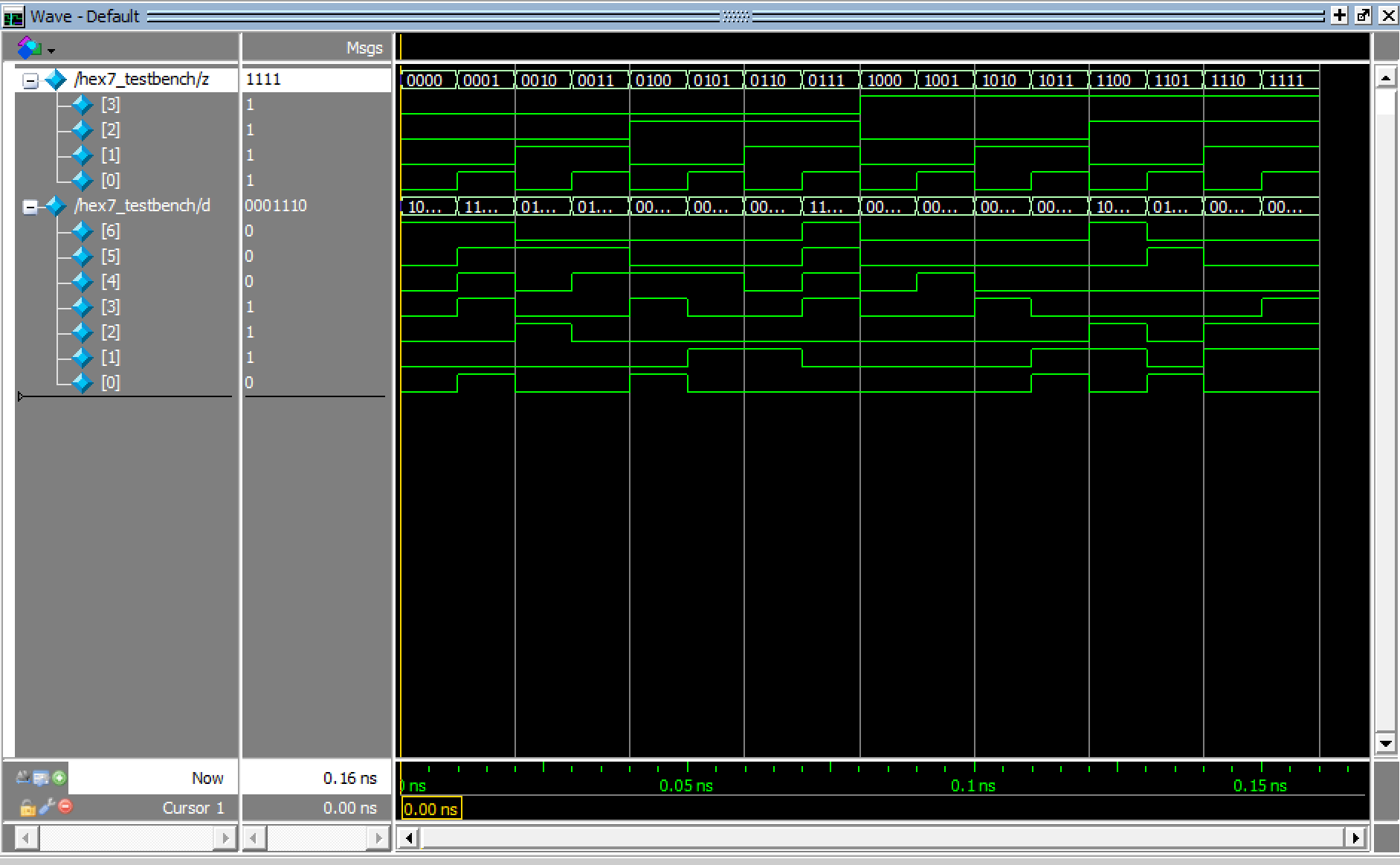
* + ASMD Chart



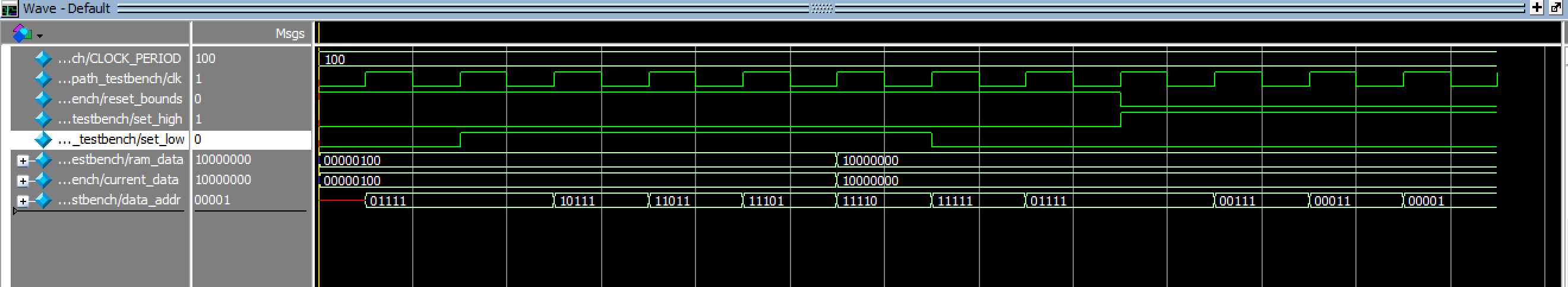
* + ram\_testbench waveform



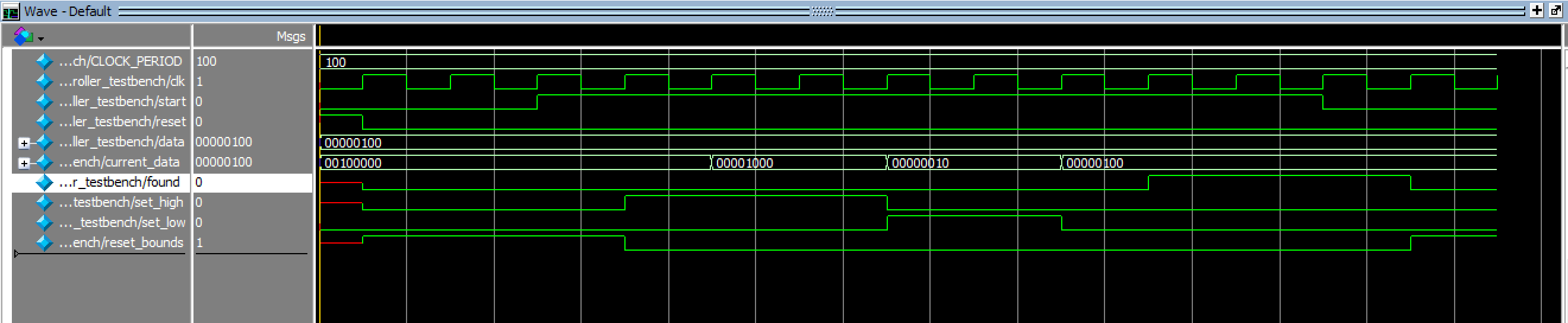
* + hex7\_testbench



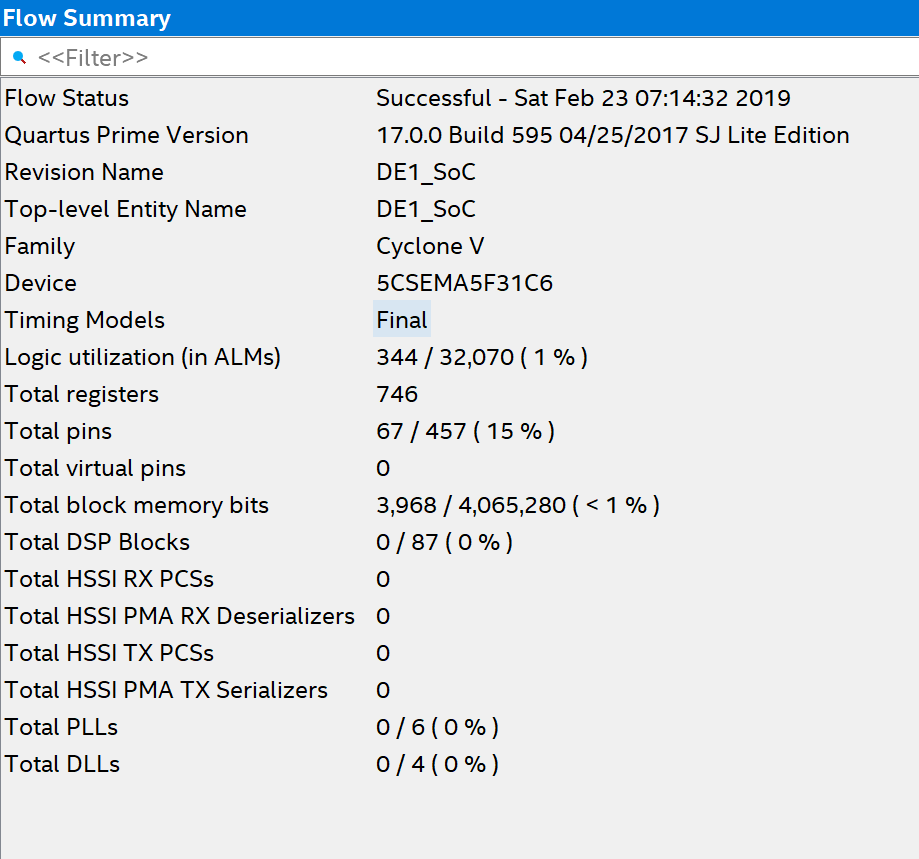
* + datapath\_testbench



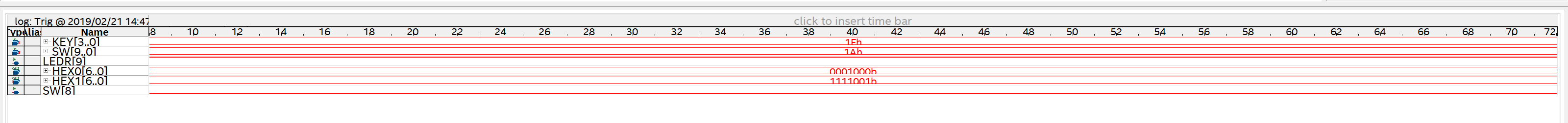
* + controller\_testbench



* + Flow summary



* + Signal Tap II



Analysis

This lab covered general issues involved in implementing algorithms in hardware. We learned to communicate with RAM for the data required in different module. This also helped us to understand ASMD charts.

Conclusion

This lab took a total time of 25 hours including reading, planning, design, coding, debugging, testing etc. Sheershak worked on Task 1 and debugging Task 2, and Hoang worked on Task 2. Problems were faced during creating a Signal Tap II functionality and debugging Verilog for binary Search. In this lab we learnt how to implement Algorithms in Hardware. We also understood why ASMD charts are important as it implies the concept of implied timing. Bot the tasks, bit-counting circuits and binary search circuit helped us to have a better understanding of ASM and ASMD. The code s commented well enough to understand the whole structure.