Jiovany Soliman, 1951338

Chloe’ Miller, 2031648

EE/CSE 371 Lab 4 Report: Implementing Algorithms in Hardware

November 06, 2023

**Design Procedure**

The purpose of this lab was to implement algorithms in hardware by translating ASMD charts to code in SystemVerilog. We first implemented a circuit that counts the number of bits set to 1 in an n-bit input A by combining the necessary datapath components and a control circuit FSM. This circuit also displayed the number of 1’s counted in A on the 7-segment HEX displays and lit an LED to indicate whenever the algorithm had finished. We then implemented a binary search algorithm, which searches through an unsigned sorted array to locate an 8-bit value A once the user presses Start. Rather than compare each value in the array to the target value, the algorithm simply compares the target value to the middle value between two bounds of the array and determines if it needs to keep searching in the first or second half of the array, repeatedly comparing the target value to the middle value between the two bounds. Lastly, we combined the functionalities of these two algorithms by adding a switch that the user could toggle to run either the bit-counting or the binary search algorithm.

**Task #1:**

For task 1, we referenced the provided ASM chart to identify our control, status, and external input signals. Then, we added to the ASM chart these signals and RTL to develop the ASMD chart needed for implementing the control and datapath in SystemVerilog. This task was straight-forward and did not take much time to code.

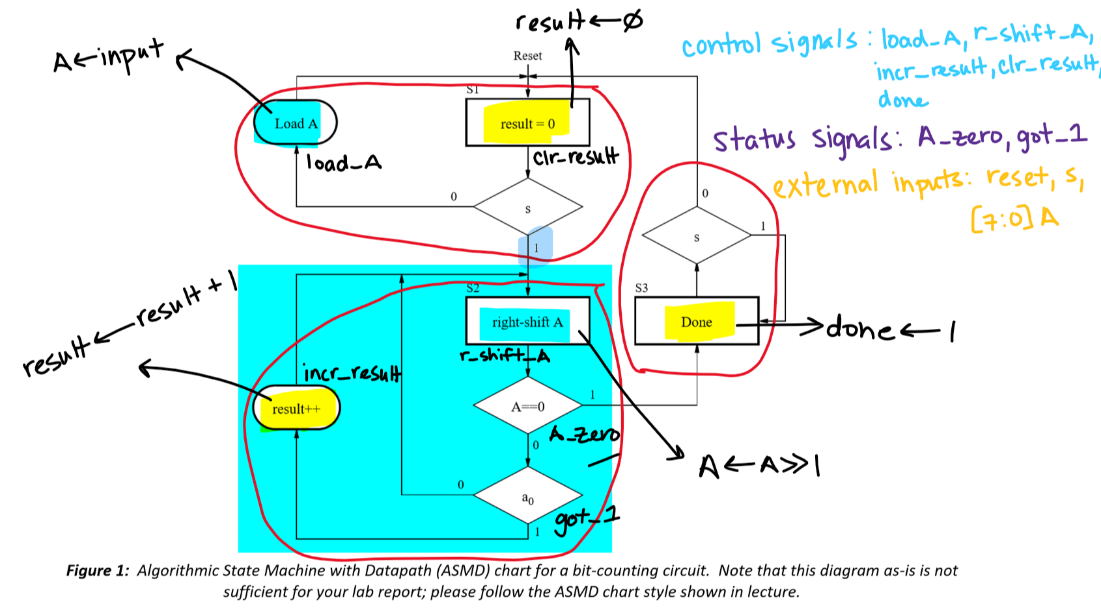


Figure 1: ASMD chart for task 1

A screenshot of a computer program

Description automatically generated

Figure 2: Snippet of module counter\_cntrl showing the control path of task 1

A screenshot of a computer program

Description automatically generated

Figure 3: Snippet of module counter\_datapath showing the data path of task 1

A screenshot of a computer

Description automatically generated

Figure 4: Snippet of task 1 model sim showing result "r" equals 2 reflecting the correct number of bits set to 1 in A.

In ModelSim, we tested our bit-counter algorithm by assigning different values to 8-bit input A, and monitored the result “r”, which indicated how many bits were set to 1 in our input. We can see in the waveform above that “r” provides the correct number of 1 bits for both of our inputs of A = 3 and A = 5 in binary, respectively.

**Task #2:**

For task 2, we needed to instantiate a 32x8 1-port RAM using a .mif file that we initialized with 32 sorted unsigned values. We then had to implement a binary search algorithm in hardware. We started by drafting an ASMD chart. The first step was to understand our control path that would later be implemented in code using a finite state machine. After we established our control path portion of the ASMD diagram, we started writing down our RTL, which would then be used to establish and implement our data path. It was very helpful to start by drawing the ASMD chart as it clarified our states, control signals, status signals, and finally our RTL. For the algorithm portion, we utilized the commonly used binary search algorithm that’s available at a variety of sources online. The main idea is that you start by checking the middle value of your search space to see if it matches your target value. Afterwards, since the search space is assumed to be sorted, you can simply determine if the target value is before the middle value or after the middle value. Once you identify which half of the search space your value is predicted to be in, you update the edges/limits of your search space to only reflect that new half. Afterwards you again select the middle value of that search space and compare it to your target value. You repeat these steps recursively until you either find your target value at which point you break out of the loop, or until you exhaust the entirety of your search space; which can be checked by comparing the 2 limits of your search space to each other. Finally, we had to include 3 additional states for timing purposes.

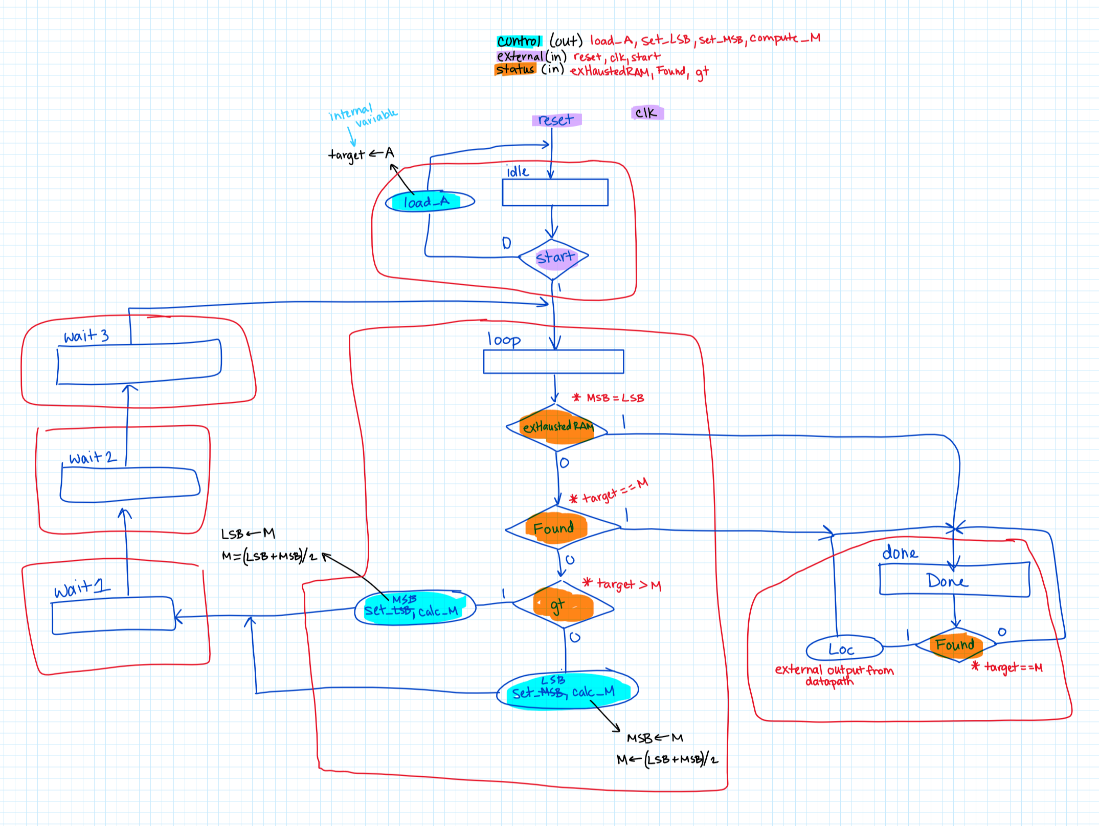


Figure 5: ASMD chart for task 2

**A screenshot of a computer program

Description automatically generated**

Figure 6: Snippet of module binaryFSM showing the control path for task 2

**A screenshot of a computer program

Description automatically generated**

Figure 7: Snippet of module BinaryDataPath showing the data path for task 2

A screenshot of a computer

Description automatically generated A screen shot of a black background

Description automatically generated

Figure 8: Snippet of binary top-level model sim showing signal F as “1” at the correct .mif file index “1” where A = 5.

In ModelSim, we tested our binary search algorithm by assigning A a value and seeing if our algorithm could find a match to our target value. We initialized our RAM with multiples of 5, as seen in the snippet above to the right. We first tested a value that would not be in the RAM, which was A = 3, and we can see that F (Found) does not become asserted because it is never found in RAM, however DONE still asserts because it indicates that the algorithm has finished comparing all the elements in RAM to the target value. Next, we tested a value that was in RAM (A = 5), and we see that both Found and DONE are asserted as soon as our algorithm finds a match to our target.

**DE1\_SoC:**

The top-level module of this lab is fairly straight forward. We simply assigned SW 0-7 to be our input to both algorithms and SW 9 to be the selector between the counter algorithm (task 1) and the binary search algorithm (task 2). We also had Key 0 as reset and Key 3 as Start to initiate our algorithms. In terms of algorithms instantiation, for counter algorithm, we made one instance of the counter control path and one instance of the counter data path and connected them together. For the binary search algorithm, we made one instance of the binary search control path and one instance of the data path and connected them together. Finally, we made 2 instances of the 7-segment display to show our counter value and the address that gets found by our binary search. We also included 2 LEDRs to demonstrate Done and Found status signals.

A screenshot of a computer program

Description automatically generated

Figure 9: DE1\_SoC top level module

A screen shot of a computer

Description automatically generated

Figure 10: Snippet of DE1\_SoC top level model sim showing the combined behaviour for task 1 and 2

For the top-level simulation, we combined the counter and the binary search testbench with the additional control of switching between the two algorithms at halfway point through the simulation. In the above ModelSim snippet, 2 yellow lines can be seen indicating the 2 points where the finished algorithm displays the final value. When SW[9] is set to 0, the counter algorithm is being selected and the result signal shows the value 2 since 2 input bits are set to 1. When SW[9] is set to 1, the binary search algorithm is being selected and the Loc signal shows the correct index when the value A = 5 is located as previously shown in the mif file. The LEDRs and HEX displays also display the appropriate behavior for both algorithms as shown in the above snippet. The Start, Done, Reset signals are all correct and behave as expected.

**Flow Summary:**

**A screenshot of a computer program

Description automatically generated**

Figure : Snippet of the Flow Summary for the top-level module DE1\_SoC

**Experience Report:**

Implementing the 2nd task; binary search algorithm was relatively difficult. Weinitially encountered issues with identifying which signals are control vs status signals. We also encountered problems with getting the DE1\_SoC top level module to work on LabsLand. We also spent a lot of time determining if the ASMD components are combinational or sequential. We had to review the class lecture slides multiple times throughout working on the lab. For teamwork, we used GitHub for easy collaboration of shared files which made code developing and debugging more efficient. It was not very helpful to reference the provided block diagrams since they were incomplete and confusing. For the next lab we would like to get a head start and aim to work together live more frequently to reduce debugging time. Additionally, using GitHub’s full functionality for version control may help make our code development process even smoother.

The estimated total time working on this lab was 24 hours broken as follows:

* Reading: 2 hours
* Planning: 2 hours
* Design: 5 hours
* Coding: 6 hours
* Testing: 5 hours
* Debugging: 4 hours.