Vincent Kwok & Hunter Hatzenbeler
EE/CSE 371
May 3, 2023
Lab 4 Report

Design Procedure:

Lab 4 centers around ASMD design to implement algorithms in hardware, particularly using datapath and controller modules to implement processes on an FPGA. The tasks in this lab show how to take an algorithm, transform it into an ASMD and block diagram representation, and incorporate it into code in SystemVerilog. Past concepts from labs are also used in the tasks of Lab 4. These include, switching between tasks with a switch, and reading from RAM memory to make comparisons between values.

Task 1:

Task 1 involves taking an 8 bit number dictated by switch inputs and counting the number of bits that are 1's. The number of bits will then be shown on a 7-segment display. This task involves taking a partially completed ASMD diagram and description and completing the diagram. The process will start on a key press of KEY3 and LED9 will light up to signal that the algorithm is finished since the code will count up to the number of 1's. If the clock cycle is slow, the number shown on the 7-segment display may be incorrect up until the LED turns on since digits could still be compared and shifted. Finally, SW7-0 will be used as input with KEY0 as a reset.

Task 2:

Task 2 involves the implementation of a binary search algorithm. The datapath again, takes 8 bits as the input. Using this input, the algorithm will search through a 32x8 RAM that is already sorted in ascending order. It will compare the input to the values that the binary search dictates to find if the value that is the input exists in the RAM. Again, reset is on KEY0 and LED9 will light up when the algorithm is completed. LED0 lights up when a location in the address is found and the Address number will be shown in the HEX displays. This task will not only use datapath and controller modules, but will also use the instantiation of RAM as well, a concept from past lab assignments.

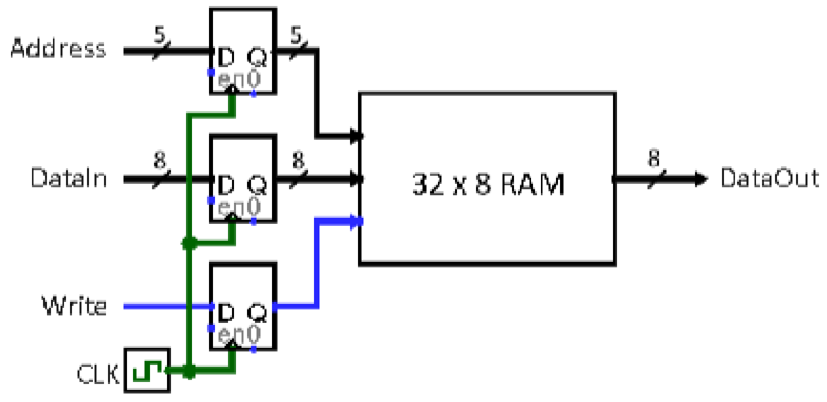


Figure 1: 32x8 RAM used for the binary search circuit. Similar to RAM modules from lab 2.

Overall System:

Task 1:

The initial ASMD drafts of the block diagram involve first viewing the rightmost bit of the input and then shifting it. But due to the diagram given by the lab specification for task 1, we were guided to change this approach to first shift the input and then viewing the rightmost bit to decide whether to increment the output. To do this, the initial loading of variables for the number of bits must start as the value of the rightmost bit instead of always being zero. Here, we also decided to go for a 3 state ASMD. While we could have made decr_P in its own state, we realized omitting it would simplify the design. Since after the decrement, the next state of S_shift would always be decr_P, making the state unneeded.

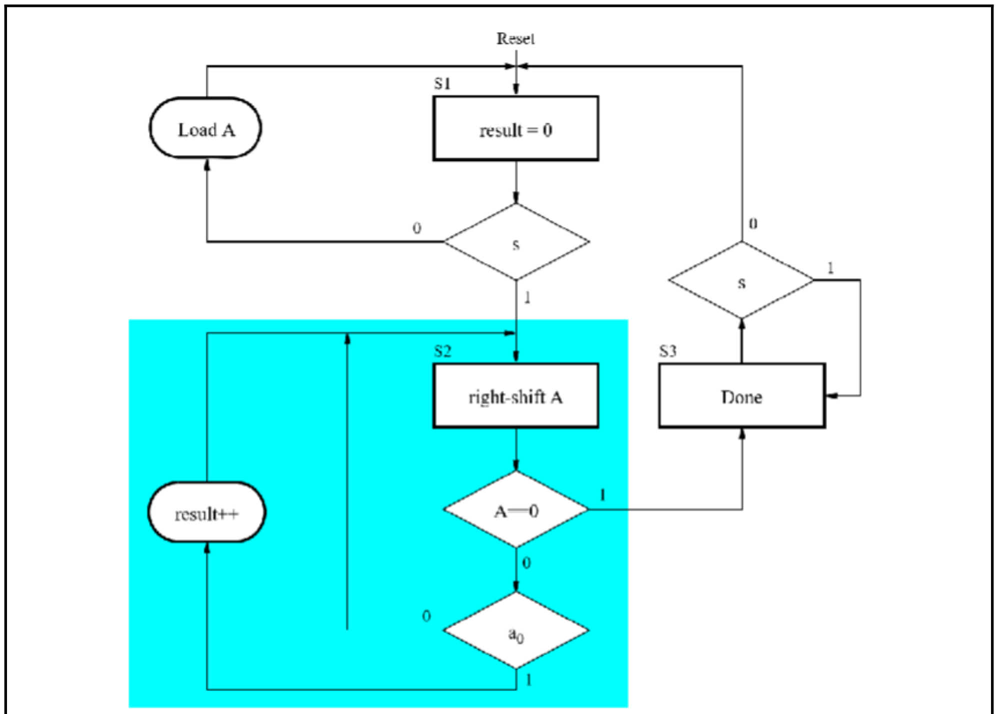
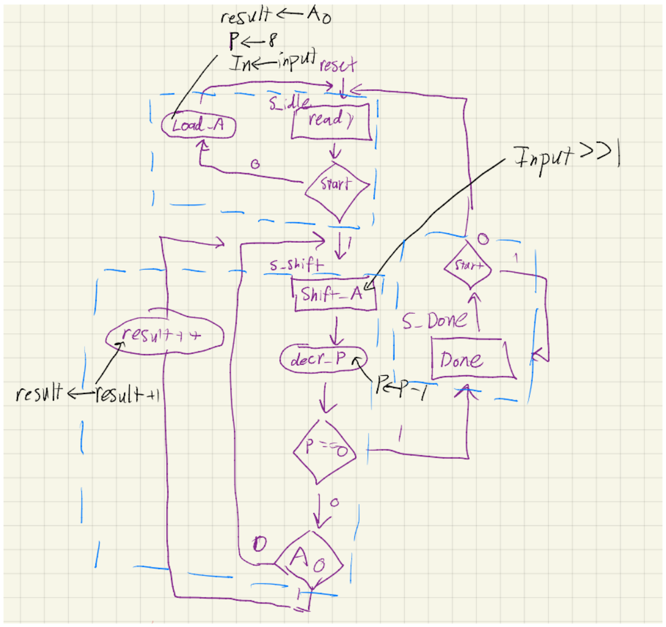
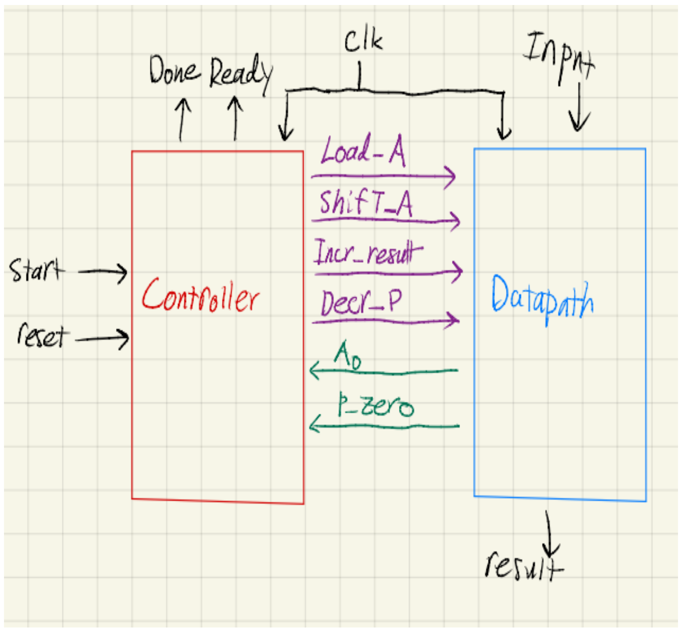


Figure 2: Task1 Provided ASMD

 result ← A0 P ← 8 In ← input reset S_idle Load_A ready 0 start Input >> 1 1 S_shift Shift_A result++ result ← result+1 decr_P P ← P-1 P == 0 0 0 A0 1 S_Done Done start 0 1 1	 Done Ready clk Input Start reset Controller Load_A ShifT_A Incr_result Decr_P A0 P_zero Datapath result
Figure 2: Task 1 ASMD	Figure 3: Task 1 Block Diagram

Task 2:

Similarly to task 1, task 2's ASMD uses 2 start conditional blocks to ensure that the button is deasserted after the algorithm completes. Because the natural clock speed of the FPGA is likely much faster than it takes for a key to be pressed and unpressed, we must ensure that the key is fully let go before the algorithm runs again. In this time between key presses, variables are loaded which is guaranteed as long as the key is let go for more than a single clock cycle (which should always happen). From this task and the previous, it seems that a 3 state ASMD is very common. 1 state to set up the algorithm, another to run the algorithm, and a last state to verify the algorithm has been completed.

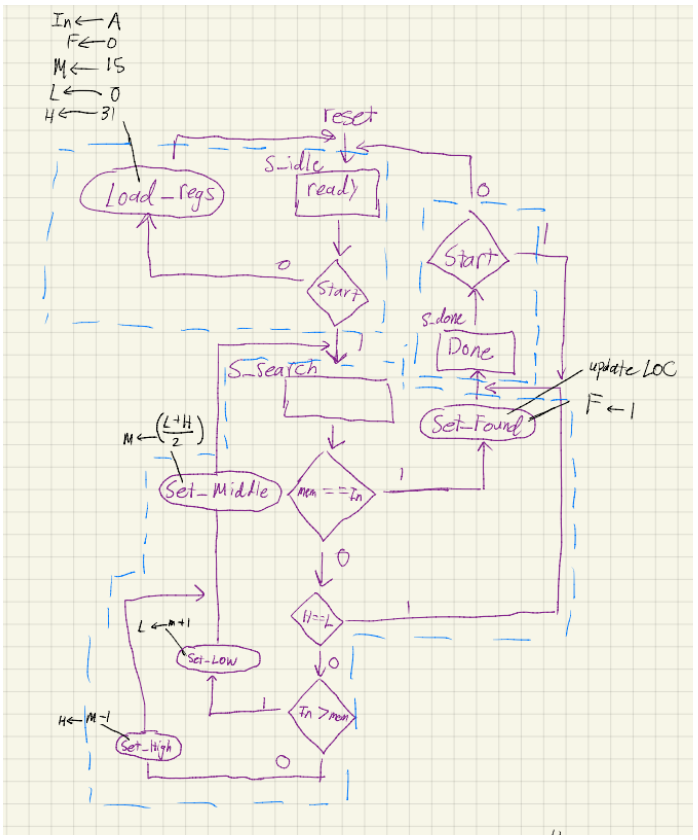


Figure 4: Task 2 ASMD

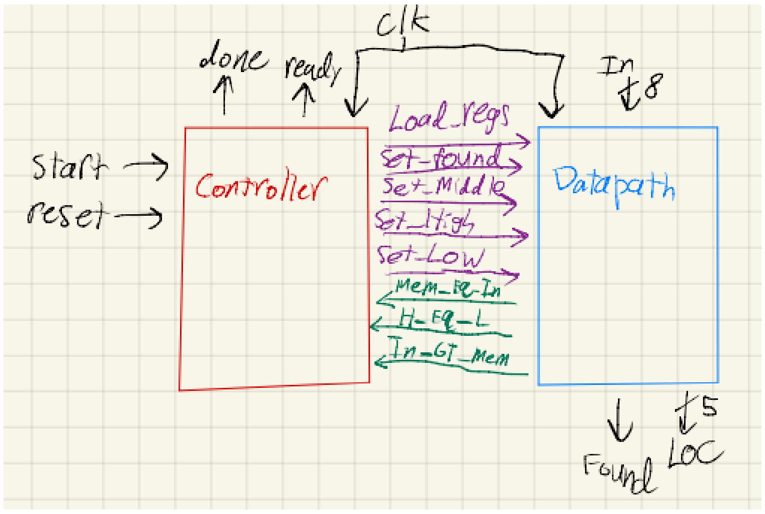


Figure 5: Task 2 Block Diagram (Excluding RAM)

Results:

Task 1:

The testbench for task 1 was written to test different cases of 8 bit combinations and seeing the output change over clock cycles as the result increments. When verifying the output, we checked the supposed output for Done (LED9) to verify that the result counts up to the value intended.

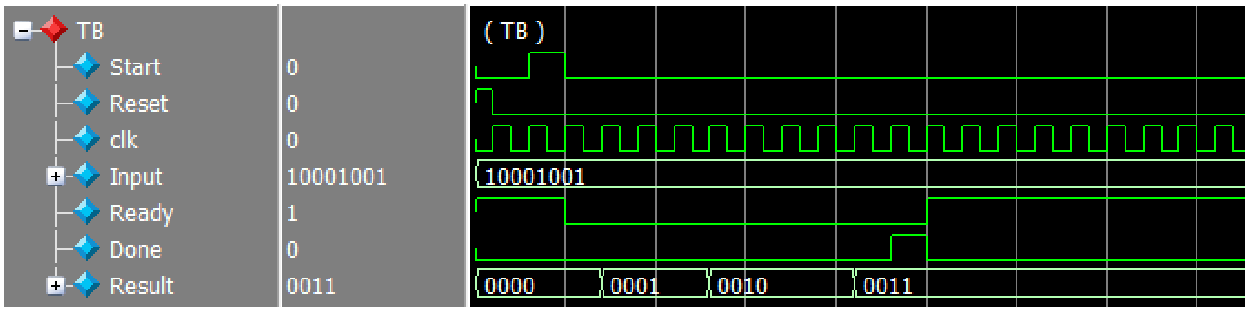


Figure 6: Task 1 simulation results

In the screenshot above, it shows an example cycle of the task running correctly. In the beginning, for two cycles before start is pressed it doesn't update anything, and ready is 1. As soon as the start signal is given, ready goes to 0, and the process begins. Given the input 10001001, the result increments 3 times. After going through all bits, the done signal is given for 1 tick, the result shows 3, and the ready signal goes to 1 again. This is the expected behavior since the input has 3 ones in it, and additionally the control signals were given at the correct times.

Task 2:

For task 2, we were able to achieve the expected results. When given an input sequence, we were successfully able to get the algorithm to find the address that contained the value we had given it. The testbench for task 2 was written to cover two primary cases. To make it easy, we initiated the .mif file with values equal to the address, so they were in order from 0 to 31.

Case 1:

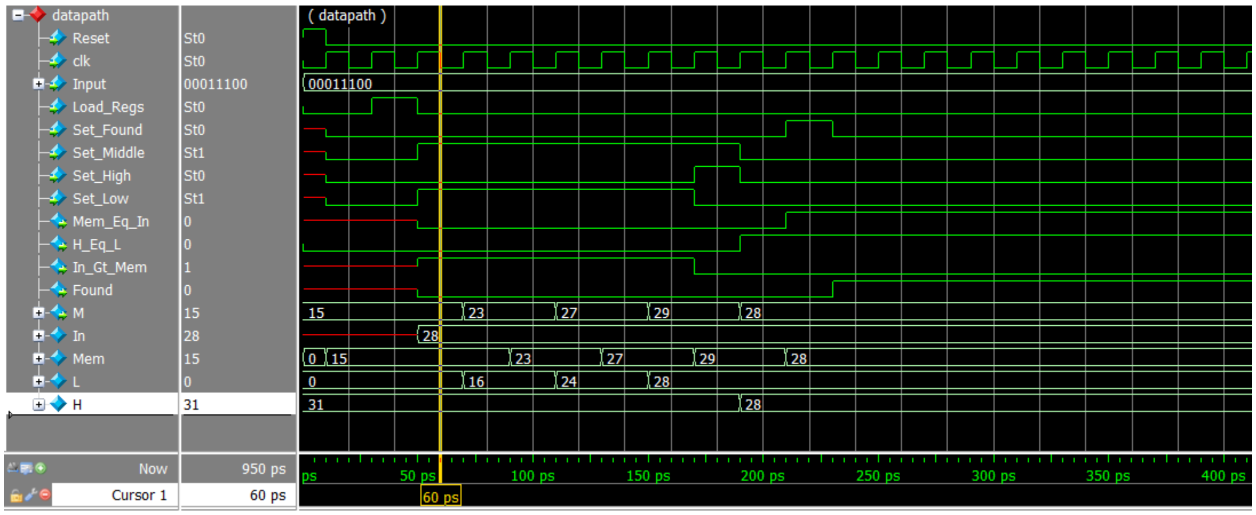


Fig 7: task 2 simulation, case 1

For the first case, we wanted to check the behavior when given an input that was within the memory. We had input the number 28, so the expected Loc should be 28 as well. As you can see, since the memory starts at 15, it then gets updated to be higher, to the value 23. Again, since this is less, it goes to 27. Now, since the next value is 29, it needs to get one more cycle to narrow between 27 and 29, hitting our target value of 28. When this happens, the signal Found is given, Loc is set to 28, and the program idles until reset.

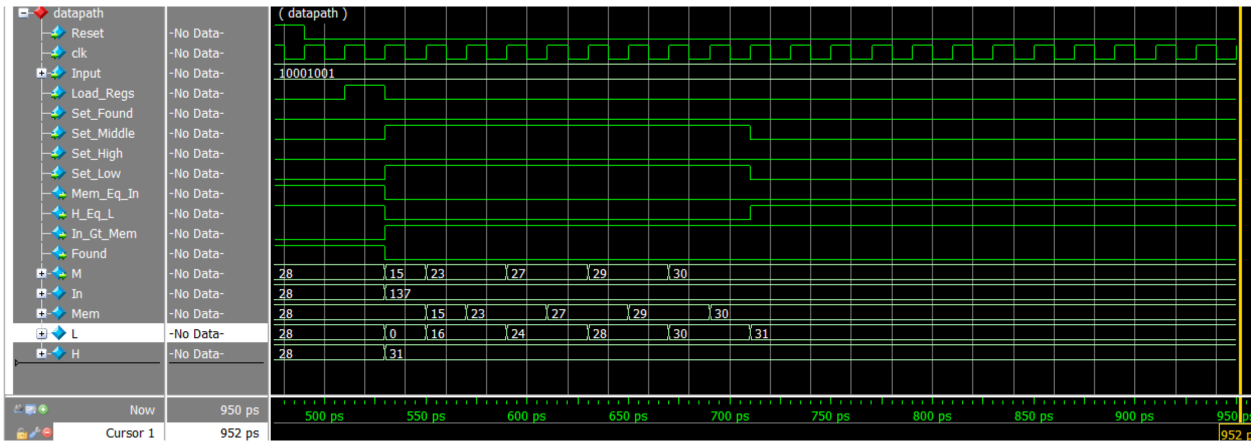
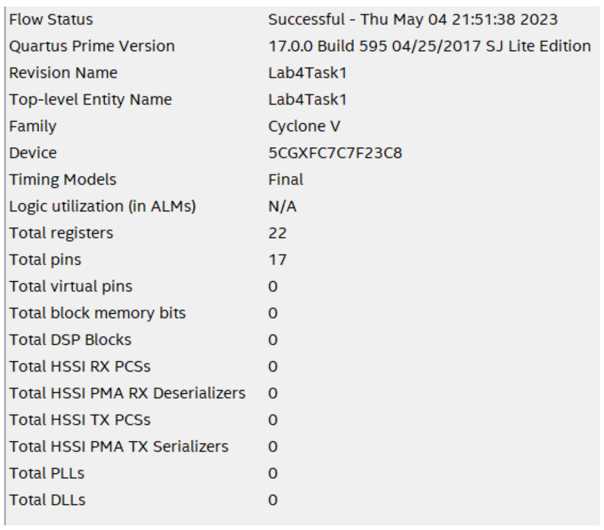


Fig 7: task 2 simulation, case 2

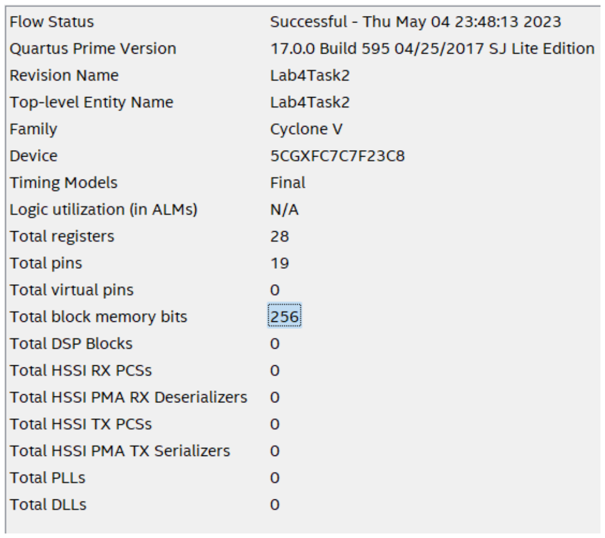
For the second case, we wanted to test what would happen if we have an input that was outside the memory, by inputting the value 137. As you can see, since the input value is always higher than the current memory value, Loc keeps rising until the low floor meets the high floor, meaning we had run through all possible memory locations. When this happen, the program is set to done, but the found signal is not given, which is the expected results.

Flow Summary:



Flow Status	Successful - Thu May 04 21:51:38 2023
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	Lab4Task1
Top-level Entity Name	Lab4Task1
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	22
Total pins	17
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	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
Total DLLs	0

Figure 9:The ModelSim Flow Summary of the task 1 module compilation



Flow Status	Successful - Thu May 04 23:48:13 2023
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	Lab4Task2
Top-level Entity Name	Lab4Task2
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	28
Total pins	19
Total virtual pins	0
Total block memory bits	256
Total DSP Blocks	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
Total DLLs	0

Figure 10:The ModelSim Flow Summary of the task 2 module compilation

Experience Report:

We found this lab to be fairly straightforward, especially when compared to lab 3. The previous lectures, especially the live demo of the isprime() algorithm was very helpful in giving us structure to programming in a simple, and efficient way. RAM implementation and task switching was explored in past labs, finally the ASMD chart templates given in the spec of the lab helped guide us even more in the correct direction. We found the instructions to be quite helpful and thorough. Task 1 was done in almost the same fashion as the demo, as long as the ASMD was completed correctly, most of the challenge was gone. Task 2 was a small extension of Task 1.

This lab took approximately 9 hours, broken down as follows:

- Reading - 30 minutes
- Planning - 40 minutes
- Design- 100 minutes
- Coding- 200 minutes
- Testing- 60 minutes
- Debugging- 100 minutes