Design of Digital Circuits and Systems, Lab 4

Implementing Algorithms in Hardware

Lab Objectives

In this lab, you will use Algorithmic State Machine with Datapath **(**ASMD) charts to implement algorithms as hardware circuits.

Introduction – ASMD Review

The implementation of algorithms in hardware is often performed by separating the data storage and manipulation components (the **datapath** circuit) from an FSM for the routing logic (the **control** circuit).

A key distinction between ASMD charts and flow charts is a concept known as **implied timing**. The implied timing specifies that all actions associated with the current state (*i.e.*, the active path in the ASM block) take place only when the *next* active clock edge occurs. For a more thorough review of ASMD charts, see the lecture slides.

Task #1 – Bit-Counting Algorithm

The ASMD chart shown in Figure 1 represents a circuit that counts the number of bits set to 1 in an n-bit input A (*i.e.*, $A = a_{n-1}a_{n-2} \ldots a_1a_0$).

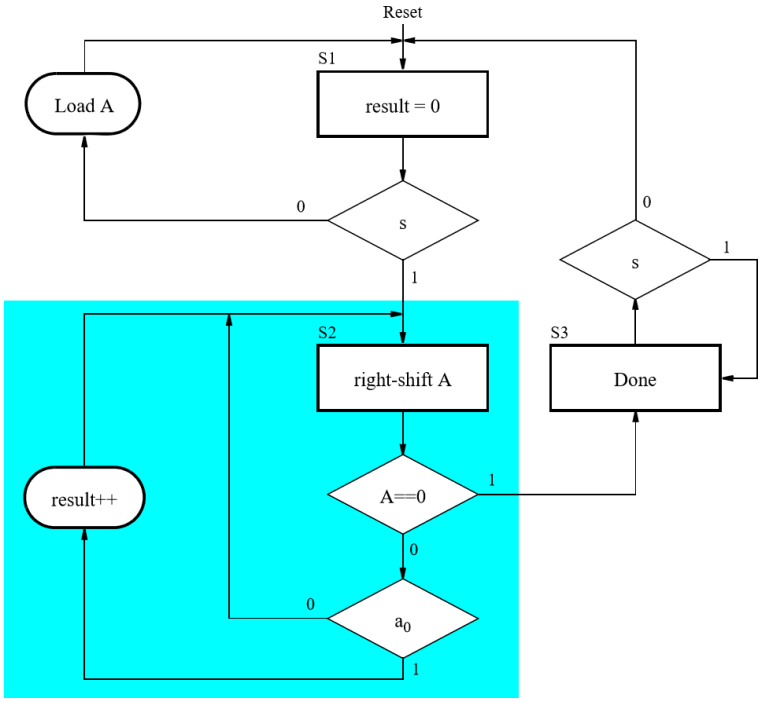

Figure 1: *Algorithmic State Machine with Datapath (ASMD) chart for a bit-counting circuit. Note that this diagram as-is is not sufficient for your lab report; please follow the ASMD chart style shown in lecture.*

Implement the bit-counting circuit given by the ASMD chart in Figure 1 in SystemVerilog to run on the DE1-SoC board, combining the necessary datapath components and a control circuit FSM.

- The inputs to your circuit should consist of an 8-bit input (A) connected to switches `SW7`–`SW0`, a synchronous `reset` connected to `KEY0`, and a start signal (s) connected to KEY3.
- Use the 50 MHz clock signal provided on the board as the clock input for your circuit. Be sure to handle metastability properly in your s input.
- Display the number of 1's counted in A on the 7-segment display `HEX0`, and signal that the algorithm is finished by lighting up `LEDR9`.

Task #2 – Binary Search Algorithm

Implement a binary search algorithm, which searches through an *unsigned* array to locate an 8-bit value A once the user presses `Start`. A block diagram for the circuit is shown in Figure 2.

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 value we seek would be in the first half of the array. Otherwise, the value we seek would be in the second half of the array. By applying this approach recursively, we can complete our search in many fewer steps.

> By default, `logic` is an *unsigned* data type in SystemVerilog (*i.e.*, the stored bits are interpreted by their non-negative binary value). Comparisons such as > and < are *unsigned* comparisons on `logic` values, regardless of what radix you are using to display their value in ModelSim.
>
> For example, if `logic [7:0] x = 8'd2;`, it turns out that `x > -1` returns false, because -1 is stored as `8'hFF`, which is interpreted as 255 in unsigned.

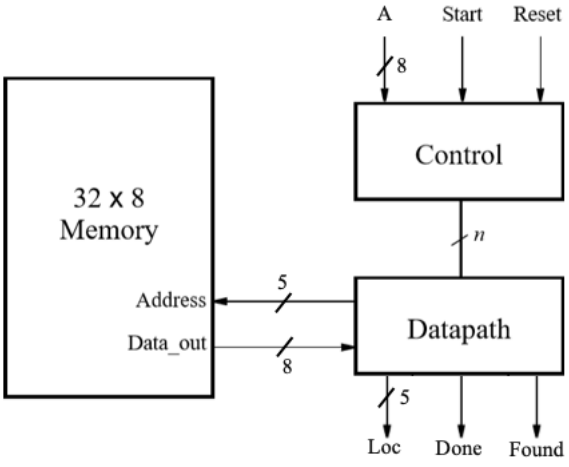

Figure 2: *Incomplete block diagram for the binary search algorithm circuit.*

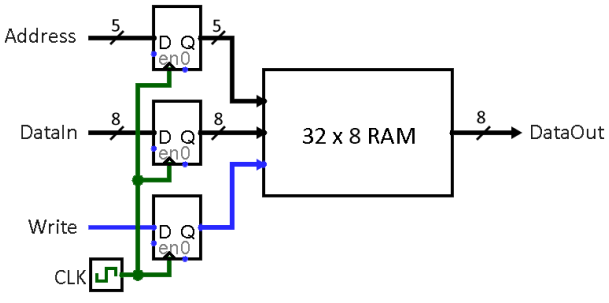

Figure 3: *32 × 8 RAM with registered inputs for the binary search algorithm circuit.*

In this circuit, the array, which you can assume has a fixed size of 32 elements, is stored in a memory module that is implemented inside the FPGA. A diagram of the memory module that we need to create is depicted in Figure 3. This memory should contain a sorted collection of 8-bit unsigned integers.

The input A should be specified on switches SW7–SW0, Start on KEY3, and Reset on KEY0. Your circuit should produce a 5-bit output Loc, which specifies the address in the memory where the number A is located, a signal Done indicating that the algorithm is finished, and a signal Found that should be high if A was found and low otherwise. Display Loc, if found, in hex on HEX1–HEX0, Done on LEDR9, and Found on LEDR0.

Perform the following steps:

1) Create an ASMD chart for the binary search algorithm. Keep in mind that the memory has registers on its input ports.
2) Implement the control FSM and the datapath for your circuit independently.
3) Create a memory that is eight-bits wide and 32 words deep in a similar fashion to Lab 2 Task #1 and then connect it to your FSM and datapath to the memory block as indicated in Figure 2. Use the DE1-SoC board's 50 MHz clock signal as the Clock input and be sure to synchronize the Start input to the clock (*i.e.*, handle metastability).
4) Create a memory initialization file (MIF) called *my_array.mif* (refer to Lab 2 Task #3 Step 2) and fill it with an ordered set of 8-bit unsigned integers of your choice. Make sure that this file is saved in the same folder as the Quartus project.
5) Make sure that your design works in simulation and on the DE1-SoC board.
6) After you have Tasks 1 and 2 working, add the ability to switch between them in the same program using SW9

Lab Demonstration/Turn-In Requirements

In-Person Demo

- Demonstrate your working Task #1 bit-counting algorithm on different inputs.
- Explain what your Task #1 reset signal does in your code.
- Show your Task #2 MIF file to the TA so they know the contents of the memory.
- Demonstrate your working Task #2 binary search algorithm for found and not found inputs.
- Explain what your Task #2 reset signal does in your code.
- Be prepared to answer 1-2 questions about your lab experience to the TA.

Lab Report (submit as PDF on Gradescope)

- Include the required **Design Procedure**, **Results**, and **Experience Report** sections.
  - Notice that Figure 1 does not currently include your control signals or ASM blocks. Also notice that Figure 2 does not currently include your control or status signals.
- Don't forget to also submit your SystemVerilog files (*.sv*), including test benches!

Lab 4 Rubric

Grading Criteria	Points
Name, student ID, lab number	2 pts
Design Procedure	16 pts
Results	16 pts
Experience Report	6 pts
SystemVerilog code uploaded	5 pts
Code Style	5 pts
LAB DEMO	50 pts
	100 pts