EE/CSE 371:
Design of Digital Circuits and Systems

Lab4: Implementing Algorithms in Hardware[1]

Lab Objectives

In this lab you will use algorithmic state machine charts to implement algorithms as hardware circuits.

Introduction

Algorithmic State Machine and Datapath (ASMD) charts are a design tool that allow the specification of digital systems in a form similar to a flow chart. An example of an ASMD 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 ($A=a_{n-1}a_{n-2}\ldots a_1a_0$). 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 ASMD 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).

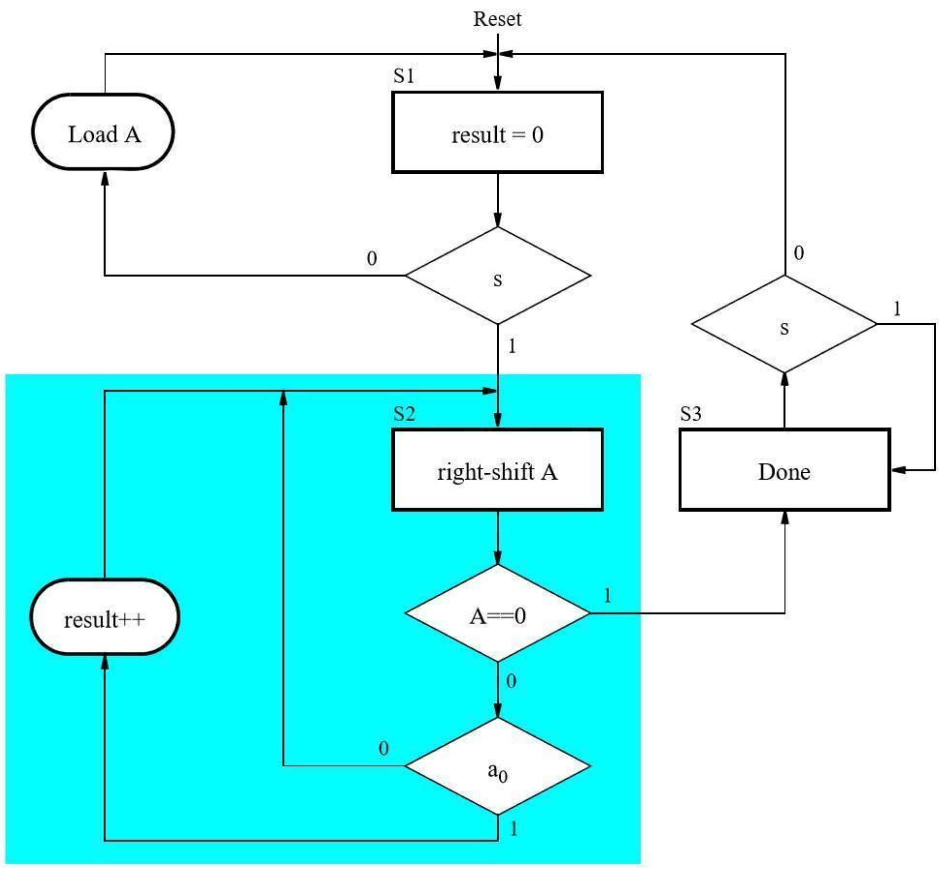

Figure 1. ASMD chart for a bit counting circuit

[1] This lab is adopted from Intel's University Program

In this ASMD 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 ASMD 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 ASMD chart transitions to state *S3*, where it sets an output *Done = 1* and waits for the signal *S* to be de-asserted.

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 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 a_0=1, change to state *S3* 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 *datapath* circuit.

Please be sure to include your ASMD Charts and Block Diagrams in your Lab Report.

Task 1

Write SystemVerilog code to implement the bit-counting circuit using the ASMD chart shown in Figure 1 on the LabsLand DE1-SoC board. Include in your code the datapath components needed, and make an FSM for the control circuit. The inputs to your circuit should consist of an 8-bit input connected to slide switches SW7−0, a synchronous reset connected to KEY0, and a start signal (*s*) connected to switch SW9. Use the 50 MHz clock signal provided on the board as the clock input for your circuit. Be sure to synchronize the *s* signal to the clock. Display the number of 1s counted in the input data on the 7-segment display HEX0, and signal that the algorithm is finished by lighting up LEDR9.

Task 2

We wish to implement a binary search algorithm, which searches through an array to locate an 8-bit value *A* specified via switches SW7−0. 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 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.

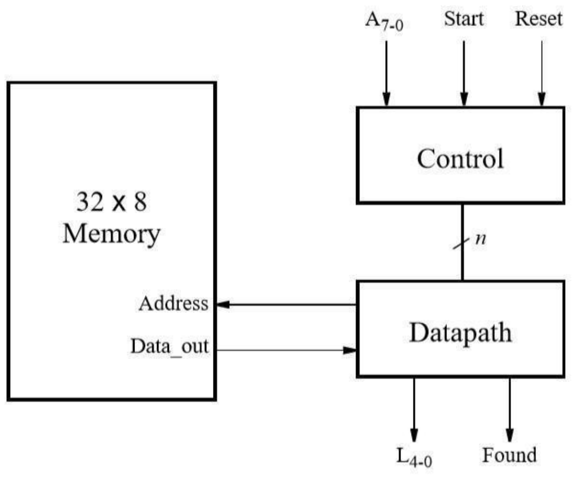

Figure 2. A block diagram for a circuit that performs a binary search.

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.

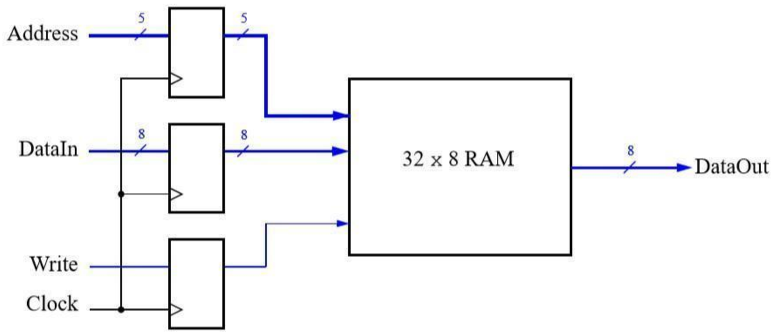

Figure 3. The 32 x 8 RAM with address register

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. If A was not found in the Binary Search, a separate logic *Not Found* should be set high.

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. Assume that the array has a fixed size of 32 elements. Make sure that in your design you account for the possibility of failure in finding a specified value.
2. Implement the FSM and the datapath for your circuit in SystemVerilog.
3. Connect your FSM and datapath to the memory block as indicated in Figure 2.
4. Include in your project the necessary pin assignments to implement your circuit on your LabsLand DE1-SoC board. The KEY/SW requirements of this lab are as follows:
   a Use switch SW9 to drive the *Start* input
   b Use SW7-0 to specify the value *A*
   c Use KEY0 for *Resetn*
   d Use the board's 50 MHz clock signal as the *Clock* input (be sure to synchronize the *Start* input to the clock).
5. Display the address of the data *A*, if found, on 7-segment displays HEX1 and HEX0, as a hexadecimal number.
6. Use LEDR9 for the *Found* signal and LEDR8 for the *Not Found* signal.
7. Create a file called *my_array.mif* and fill it with an ordered set of 32 eight-bit integer numbers.
8. Compile your design, and then upload it onto LabsLand and test it.

Lab Demonstration and Submission Requirements

- While working on the lab, on Padlet, write about a problem you had in the lab and the fix to it, and share a tip or trick you learned. You can also share an aha moment that you discovered while working on the lab. Avoid duplicating comments made by your classmates. NO videos for this Padlet task, please use textual comments. The link to the Padlet can be found in the corresponding Canvas assignment. Please note that the grace period does **not** apply to this portion of the lab.
- Record a demo video for each of the tasks in this lab. The length of each video is expected to be 2-3 minutes. You will need to demonstrate the soundness of your design by executing the design on the FPGA. You do not need to include Modelsim simulations in your demo for this lab. You only need to demonstrate the functionality of your system. Your demo video must be a screen recording created from software

like ActivePresenter or Zoom. Please refer to the grading rubric on Canvas. Please note that the grace period does **<u>not</u>** apply to this portion of the lab.

- Write a Lab Report, as framed by the Lab Report Outline document on Canvas. Comment your code. Follow commenting guidelines as discussed in the Commenting Code document on Canvas. Please be sure to include your ASMD charts and block diagrams in your Lab Report. Please include the simulation results in your lab report. Please include waveforms for all modules you used unless it is explicitly stated otherwise. Submit your lab report as a pdf file and all of your SystemVerilog files on Canvas. Please refer to the grading rubric on Canvas. As the grace period applies to the lab report and code assignment, you may submit it up to 2-days late without receiving a late penalty.