BCD to Binary Converter

ECE 2700 Final Project

Colston El-Hayek, Kyle Al-Attar, Robert Kayfish, Andrew Picklo
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: celhayek@oakland.edu, rkayfish@oakland.edu,
ajpicklo@oakland.edu, kalattar@oakland.edu

Abstract- The purpose of the project is to design and implement a 12 bit Binary Coded Decimal (BCD) to binary converter on an FPGA board through use of a keyboard input as well as 7-segment displays and LED outputs. The 12 typed bits are stored through the use of D-Flip Flops, which are controlled by a Finite State Machine (FSM). The 12 BCD bits are then converted into a 10 bit unsigned output using an algorithm. The project was completed well within the group's initial expectations and tested with a high degree of success. The project could be improved by using a multiplexer to make the full output displayed all at once, rather than changing every time a 1 or 0 is pressed on the keyboard.

I. Introduction

This report covers the design and implementation of a BCD to binary converter. The 12 bit BCD is typed into a PS/2 keyboard, and a field-programmable gate array (FPGA) board shows its binary and decimal equivalents. The next sections of the report give an overview of the top level design, a detailed description of the individual circuit components, as well as a summary of the results obtained from experimental trials.

The motivation behind this project was for the group to demonstrate their knowledge of digital circuits, as well as Very High Speed Integrated Circuit Hardware Description Language (VHDL) coding. All knowledge necessary to carry out this project was taught in class. The project was designed using components such as FSM, encoders, adders, Flip Flops, and counters. Additionally, the design is implemented in VHDL using a myriad of commands such as generic and port map statements.

BCD is commonly used in computer memory through address numbers [1]. In Addition, BCD is becoming increasingly useful in embedded microprocessors and are still used in real-time clock chips [1]. This project allows for the conversion of any 12 digit BCD to its binary equivalent.

II. Methodology

A. Top Level Design

For this design, a PS/2 keyboard is connected to an FPGA board via a USB port; the user then types in a 12 digit Binary Coded Decimal. The circuit then stores all 12 bits using a FSM. At this point, the circuit implements an algorithm that uses adders and multipliers to convert the BCD to its decimal and binary equivalents, with the Binary equivalent shown on 10 LEDs and the decimal equivalent shown on three 7 segment displays. The top level block diagram is shown in Figure 1.

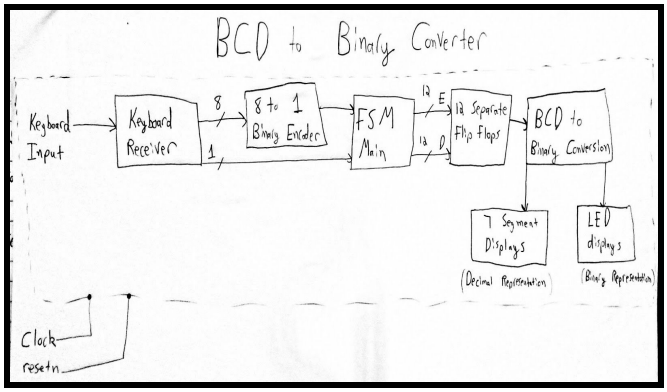

Figure 1: Top Level Block Diagram

B. Keyboard Reader

The keyboard reader portion of the project consists of an already made file taken from the class notes, the main FSM for the entire project, the binary encoder, and 12 D-Flip Flops, as shown in Figure 2, in Appendix A. The file, called my_ps2keyboard, consists of the two main components: ps2read and ps2keyboard. When a key is pushed on the keyboard, the ps2read component reads the hexadecimal scan code associated with the pressed key and outputs a 10-bit binary number in the form of TTL communication[2]. This 10-bit output then has its two most significant bits (MSB) dropped. This is because those two bits are the stop bit and the parity bit, which are not needed for the purposes of this project [6]. The resulting 8-bit hexadecimal number is then fed into a D-Flip Flop and a FSM. The enable to the D-Flip Flop is controlled by the FSM and also feeds into a counter that outputs a 1-bit "done" signal once the count is reached.

Once the 8-bit number is put through the D-Flip Flop, it then feeds into a binary encoder that outputs a 1-bit input into the main FSM along with the 1-bit "done" signal from the previously mentioned counter. The FSM then controls which of the 12 D-Flip Flops will be storing which digits. These D-Flip Flops store all of the 12 typed inputs.

C. Binary Encoder

When the user types the number '1' into the keyboard, the 8 bit scan code 0x16 is outputted from the keyboard receiver block. Similarly, when the user types in '0' into the keyboard, the scan code 0x45 is outputted from the keyboard receiver block [4]. The binary encoder is responsible for converting these scan codes into the digit that was typed into the keyboard (either 0 or 1). That way, the circuit can store the actual typed digits, rather than their hexadecimal scan codes, into the D-Flip Flops.

D. Main FSM

If the user types in more than one digit into the keyboard, the previous scan code is overridden and replaced with the scan code of the newly typed digit. As a result, 12 D-Flips Flops are required to store all the 12 typed digits into memory. In order to control the storing of all the typed digits, a FSM was used. In addition to receiving a clock and "resetn" signal, the FSM receives a one bit signal from the binary encoder, which represents the digit typed into the keyboard. The FSM also receives a one bit "done" signal from the keyboard receiver block, which indicates that the scan code output of the keyboard receiver block is valid. The FSM outputs 12 bit signals named "D" and "E", with each bit of the output signals being connected to the "D" and "E" signals, respectively, of 12 separate D-Flip Flops. A Block Diagram of the FSM is shown in Figure 3.

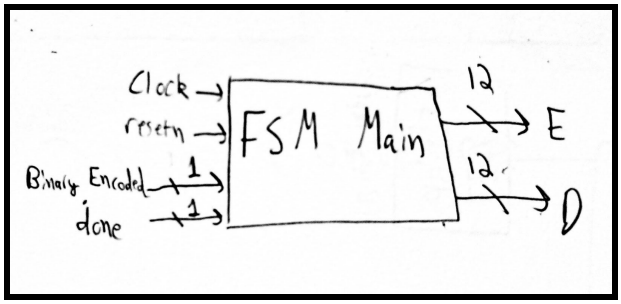

Figure 3: Main FSM Block Diagram

The FSM contains 12 states, with one state for each typed number[4]. To start off, the "resetn" input is set to 0, and the FSM remains in State 1. Once the "done" input is high, the FSM moves to State 2. When this happens, the Binary Encoded Input is outputted from the FSM, and it becomes the "D" input to the most significant flip flop, and only the MSB of the FSMs "E" output is high. This way, each typed value can be stored into its appropriate flip flop, and the entire 12 bit typed input can be saved. For example, if the FSM is at State 2 and it receives a "done" signal, the 2nd MSB of "D" is set to the Binary Encoded input. In addition, the 2nd MSB of "E" is set to '1' and the rest of the bits of E are set to '0'. As a result, the value of the second bit typed into the keyboard will be stored only into the second flip flop, as the rest of the flip flops are not enabled. This cycle repeats until all 12 states have been completed, meaning that all 12 typed numbers have been input into the keyboard. At that point, the FSM moves to State 1 and the process repeats. An Algorithmic State Machine (ASM) chart of the FSM is shown in Figure 4 in Appendix B.

E. 7 Segment Displays

The seven segment display portion of the project uses 3 different components: a counter, a FSM, and a hex-to-7 segment converter. This portion of the project consists of code from a provided example. Its XDC file, counter, and number of active displays were modified to fit the project's desired specifications. Its FSM has an input of 3 nibbles, which are taken from the 12 bit output of the D-Flip Flops from the keyboard receiver block, along with an input from the counter. The input from the counter controls both the "AN" and "segs" signal. The AN signal is responsible for controlling which seven segment displays are turned on. The "segs" signal maps to the hex-to-7 segment converter, which then maps directly to the currently

active 7-segment display in order to show the correct nibble[4]. The FSM changes both the “AN” signal and “segs” signal during the same clock tick so that the 7-segment displays show their corresponding numbers. A block diagram of the 7-segment display is shown in Figure 5.

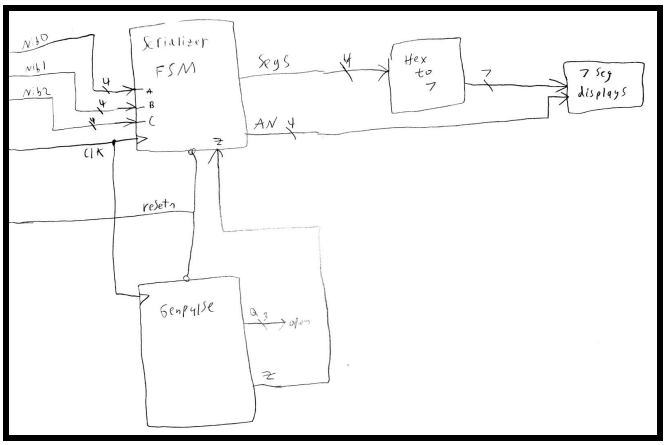

Figure 5: Seven-Seg Display Block Diagram

F. BCD to Decimal Conversion Algorithm

The BCD to Decimal converter uses a pair of multiplier blocks and a pair of 10 bit adders. The block takes a 12 bit BCD input, which is divided into three 4 bit nibbles named “nib0-2”. “Nib0” gets six 0's concatenated to it to become a 10 bit "nib0a" signal. “Nib1” gets multiplied by ten, then gets two 0's concatenated to it to become a 10 bit "nib1b" signal. “Nib2” receives three 0's concatenated to it, then is multiplied by 100, producing a 14 bit "nib2b" signal. Because “nib2” can take on values from 0-9, the largest value “nib2b” can take is 900, which requires 10 bits in Binary. Thus, the leading four digits are always '0' and can be removed to create a 10 bit "nib2c" signal. The “nib1a” and “nib2b” signals are then fed into a 10 bit adder, with the output and “c” signals feeding into a second 10 bit adder, along with the “nib2c” signal. The result of this calculation is a 10 bit unsigned binary output [3][4]. A block diagram of the BCD converter is shown below in Figure 6.

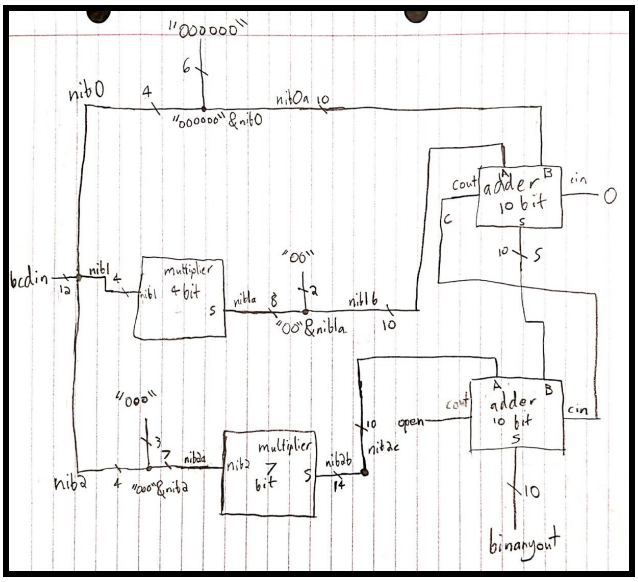

Figure 6: BCD Converter Block Diagram

III. Experimental Setup

To verify the functionality of the project, a testbench was written in VHDL, through Vivado, in order to simulate the converter’s behavior. Additionally, an external interface test was performed on an FPGA board in order to test the project.

For the behavioral simulation, the keyboard reading block was removed, and a pair of simulated keyboard inputs were provided to the BCD converter and 7-segment blocks. The BCD converter was expected to correctly convert the 12 bit BCD input to a 10 bit binary signal , and the 7-segment displays were expected to correctly display the decimal equivalent

For the external interface test, the project code was programmed onto a Basys 3 FPGA board, and a PS/2 keyboard was connected to the board via a USB port. A set of five sample 12 bit BCD test inputs were used to test the functionality of the board. The five inputs were randomly selected to determine how the code would perform given a variety of inputs. Each test input was then typed into the PS/2 keyboard (MSB to LSB) for the circuit to interpret and convert. The FPGA was expected to show the correct binary equivalent on the LEDs and the correct decimal equivalent on the 7-segment display.

IV. Results

The results from the external interface test were well within expectations and are viewable in the linked video provided in Appendix A. For each of the five external interface trials, the decimal outputs on the seven segment displays matched the hand-calculated values that were computed using the skills learned in class. Similarly, the binary outputs matched the hand-calculated values that were computed using the skills learned in class. One unexpected result that occurred while checking the LED results was that

one of the hand-calculated binary values was incorrect; the project was used to correct said value. The only time the LED binary values deviated from normal values was when an invalid BCD was input into the keyboard, since the maximum value of a 4 bit BCD cannot exceed 1001. The BCD to binary conversion code was not designed to correctly handle these types of inputs, so it output meaningless data to the FPGA's LEDs. By using knowledge gained from class, the group ensured that both tests did not contain any invaild BCD values.

The second set of results were obtained from a test bench file run in Vivado. For these test results, the keyboard reader block was removed, so that a 12 digit BCD could be directly input into the BCD to binary converter. The results from this test are shown in Appendix B in Figure 7.The results from this test were also as expected. For each 12 bit BCD input, the correct 10 bit binary output was observed. These results prove the statement made in class that BCD is a less efficient way to store data, since a 12 bit BCD value can be represented with only 10 bits when converted to binary. While setting up the test, the group determined that it was beneficial to avoid meaningless data, so only valid BCD numbers were used as inputs. As a result, all binary value outputs had an explainable reason as to how they were calculated in the code.

Conclusions

This project was able to successfully convert a 12 bit BCD input from a PS/2 keyboard to a displayed binary output on a set of LEDs, as well as a decimal output on the seven segment displays of a Basys 3 FPGA board.

Through the knowledge gained in class, it was determined that a binary encoder, an FSM, and a set of D-Flip Flops were necessary to correctly read and interpret the signals from a PS/2 keyboard. This project also showed that a serializer is necessary to correctly display multiple digits at a time on a seven segment display on the Basys 3 board. One issue that remains to be solved would be coding a testbench file that includes the keyboard reader block. Simulating the testbench with the keyboard reader block proved to be difficult, so it was more efficient to test the keyboard reader block using an external interface rather than a coded testbench.

The project could be improved by changing the design to display the final converted output of the circuit all at once, rather than sequentially as each individual bit of the BCD input is typed. A multiplexer could have been used to accomplish this, with a signal from the main FSM controlling the selector signal.

Despite these potential improvements, the project was still completed with a high degree of success and the knowledge gained through this project can be applied towards the group member's continued learning experience.

References

[1] What is BCD and How is it Used in Automation? (2020, March 23). Retrieved December 12, 2020, from https://realpars.com/bcd

[2] Skill Level: Beginner, & Jimblom | November 23, 2. (n.d.). Retrieved December 12, 2020, from https://www.sparkfun.com/tutorials/215

[3] Lecture Notes - Unit 4

[4] Lecture Notes - Unit 5

[5] Lecture Notes - Unit 6

[6] Lecture Notes - Unit 7

APPENDIX A

(https://drive.google.com/file/d/1vA4av1TpVnwTqaM3dTbfXN0yEs8A_fMb/view?usp=sharing)

APPENDIX B

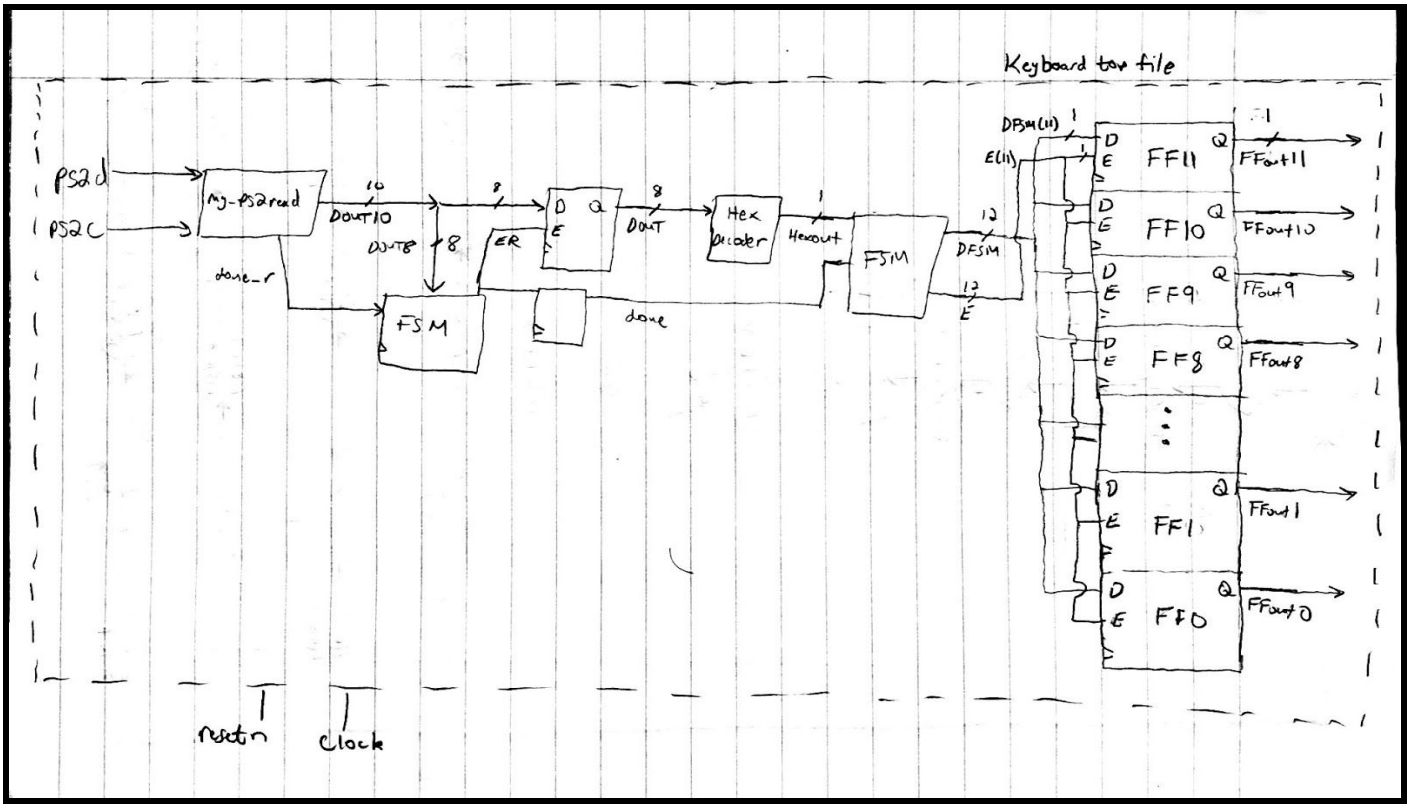

Figure 2: Keyboard Reader Block Diagram

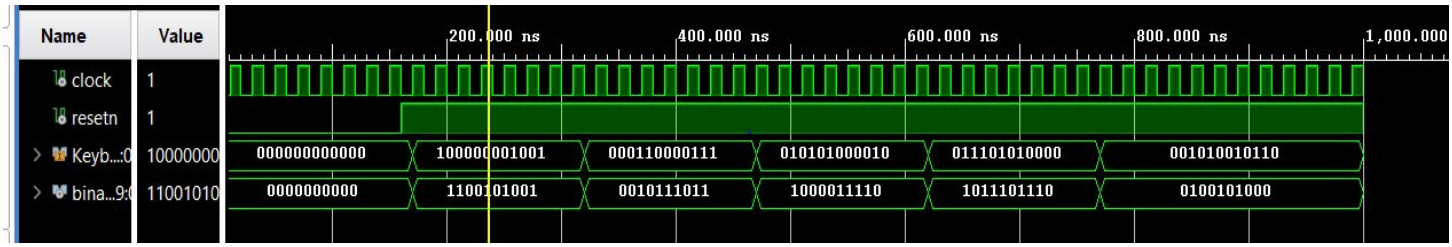

Figure 7: Keyboard Reader Block Diagram

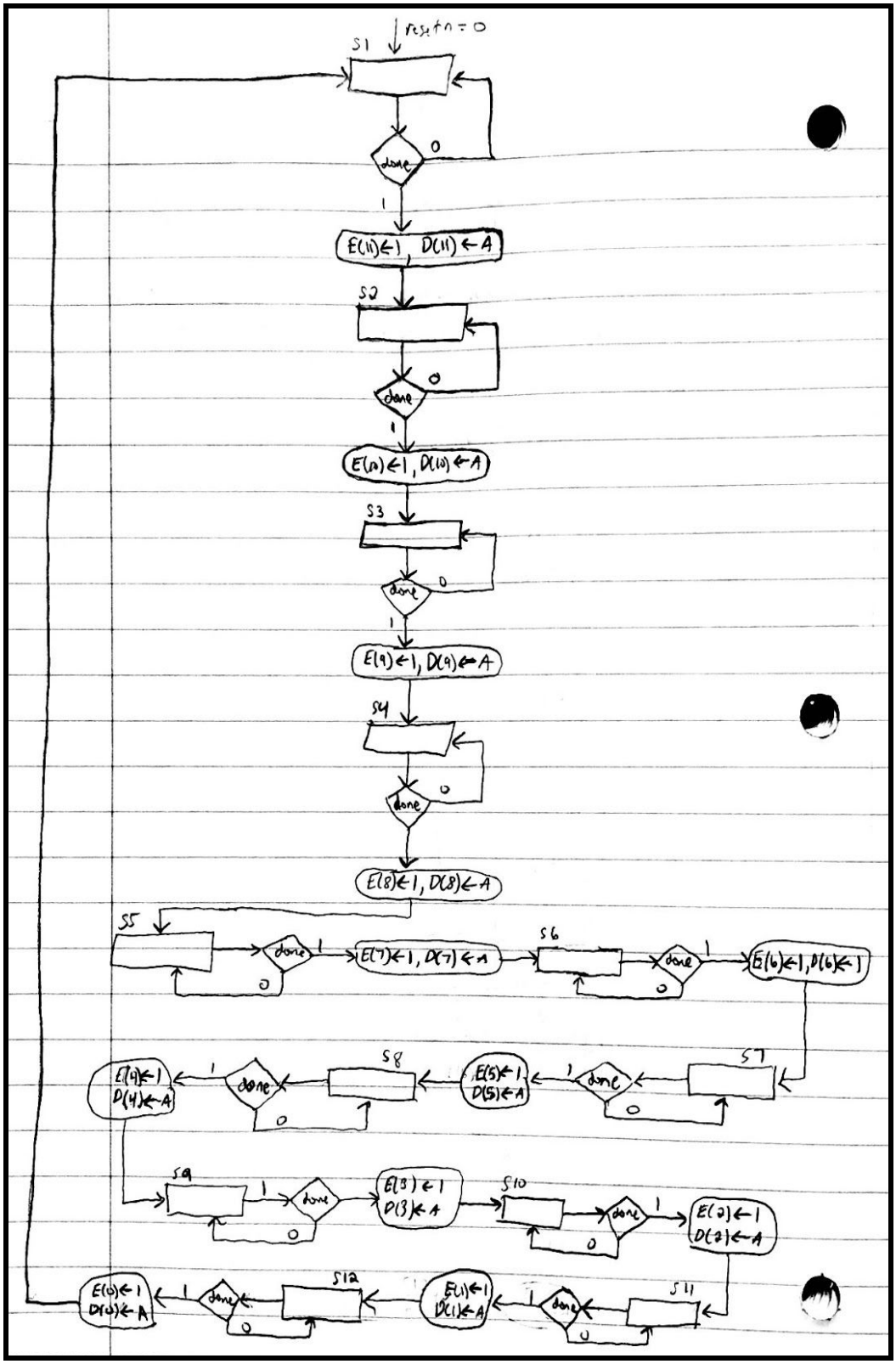

Figure 4: ASM Chart of Main FSM