Benjamin Straus
May 9, 2018
DSF Final Project
Robot Arm Report
Video Link: https://youtu.be/ygAyl9CfRJU

Introduction

This project was broken down into various stages. Each step was thoroughly executed which involved designing, implementing, and testing the relevant parts of the project. The hypothesis is that if each part works as expected (taking specified inputs and outputting correct outputs), the system as a whole will function as expected. This logic is exactly how the project was structured. About 1 month was spent making solidified schematics and test circuits of the non-FSM components (12-bit counter, up & down logic, edge detector, synchronizer, Schmitt trigger, debouncer, register file, and comparator). Then, about 1 week was spent on understanding the system design and using that to design, make a schematic for, and simulate the FSM. Finally, 1 week was spent on implementing the components (including the FSM) on a breadboard, debugging, and testing the system as a whole. This will only be a fail-proof process as long as the hypothesis that the system will work if the individual components work holds. And, this hypothesis only will hold if the theory of the system holds. So, we begin with stage 1: the theory of the system.

Stage 1: Understanding the Circuit as a Whole (Assumptions, Preliminary Analysis)

This system has two main inputs, phase (Ph) and quadrature (Qt), and two main outputs, step and direction (dir). The inputs are the outputs of a printed circuit board (PCB) that is connected to a motor. The details of this PCB and the motor outputs are beyond the scope of this report. However, the essential part of them are the outputs used here as inputs (Ph, Qt). These signals pulse to indicate a number of ticks the motor has turned and whether the motor has turned clockwise or counterclockwise is indicated by looking at both Ph and Qt. The outputs of this system are step and direction. Step is a pulsing signal fed to the PCB then motor to indicate to rotate one "tick" and the direction output indicates the direction to turn.

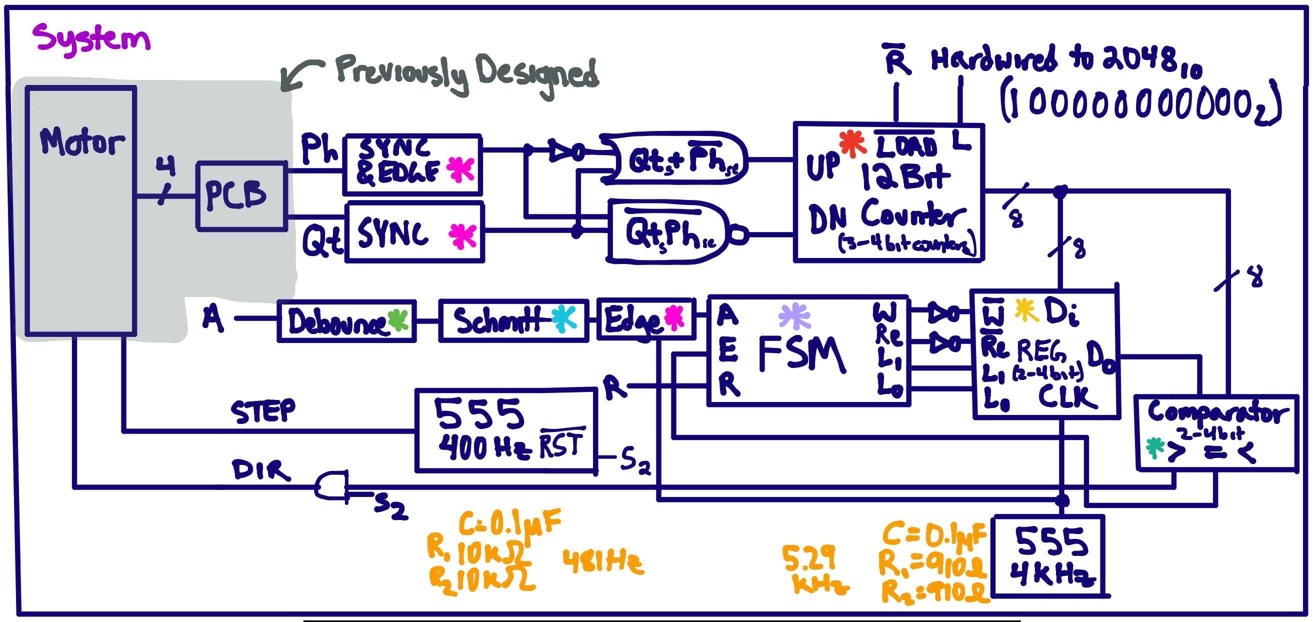


Figure 1: Main System Diagram

Benjamin Straus
May 9, 2018
DSF Final Project
Robot Arm Report
Video Link: https://youtu.be/ygAyl9CfRJU

Once the inputs and outputs of the system are understood, the system that takes inputs to outputs must be designed. A high level schematic is seen in Figure 1. We see that in a very broad way, that the inputs are counted and stored in a register. This is managed by an FSM which regulated the address and writing condition of the register. The FSM also manages the address and reading condition so that the comparator receives the goal address and the current position. The comparator then outputs a bit that represents direction (that goes to the motor) and an equal output (that is fed to the FSM). This is the basic understanding of the circuit. However, to understand the logic behind when certain actions occur, the FSM must be explained. This is done in stage 2.

In understanding the system, we see that each component is designed to receive specific inputs and then output specific outputs. The assumption here is that if each component works as designed, then the system as a whole will function correctly. Each component is rigorously tested (later on in stage 3) to ensure that they will work, then the system as a whole should function correctly as observed (later in stage 6).

Stage 2a: Design of the FSM (Design Procedure, Simplification, Engineering Decisions)

The FSM was the main part of the system that required designing. My FSM is thoroughly described in the FSM simulation report submitted separately on Blackboard. For convenience, the following design explanation, decision process, and procedure comes from that report.

The goal of this FSM is to carry out the design of the robot arm as specified by the final project guidelines. To do this, I made the decision to use eight states (shown in Figure 2). This was a thoughtful choice as it uses all values of three state bits, thus wasting 0 bits of potential data. To progress the states, 3 inputs are needed (<u>A</u>ction Button, <u>R</u>eset, <u>E</u>qual output from comparator; See legend in Figure 2 for more info). The output of the system are two address bits, L_1L_2, and a read (Re) and write (W) output.

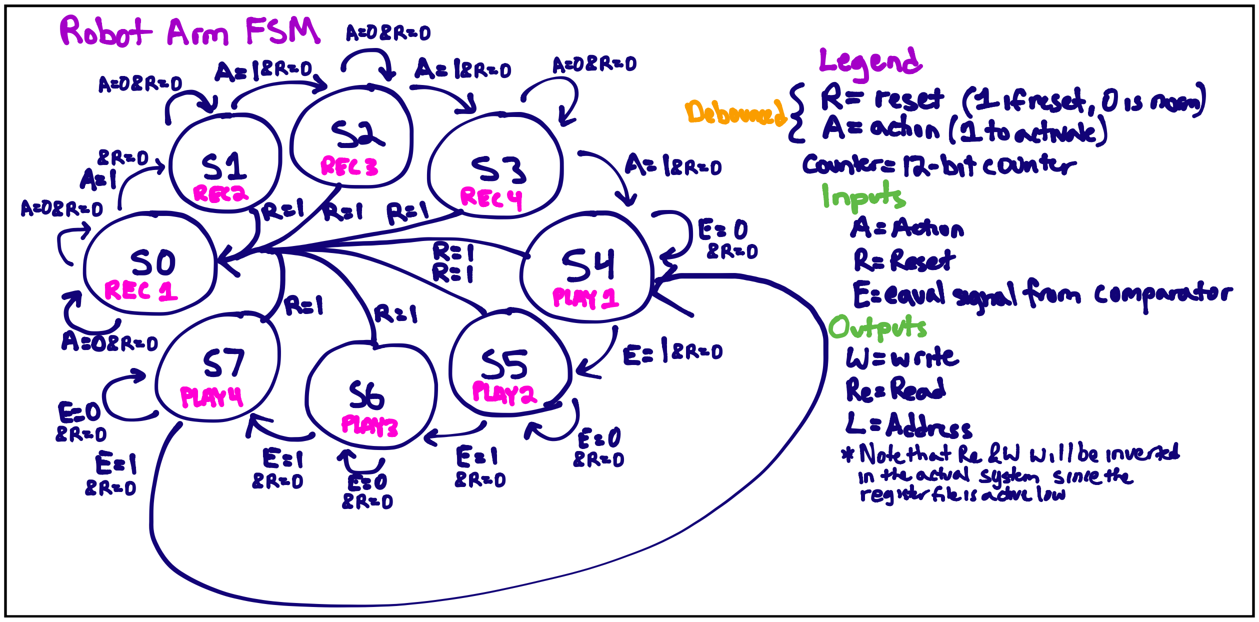


Figure 2: Finite State Machine Design

Benjamin Straus
May 9, 2018
DSF Final Project
Robot Arm Report
Video Link: https://youtu.be/ygAyl9CfRJU

State Encoding & Outputs

State	$S_2S_1S_0$	W Re	L_1 L_0
S0	0 0 0	1 0	0 0
S1	0 0 1	1 0	0 1
S2	0 1 0	1 0	1 0
S3	0 1 1	1 0	1 1
S4	1 0 0	0 1	0 0
S5	1 0 1	0 1	0 1
S6	1 1 0	0 1	1 0
S7	1 1 1	0 1	1 1

Next State Logic Table

Current State	Inputs	Next State
$S_2S_1S_0$	A R E	$S_2'S_1'S_0'$
0 0 0	0 0 x	0 0 0
0 0 0	1 0 x	0 0 1
0 0 1	x 1 x	0 0 0
0 0 1	0 0 x	0 0 1
0 0 1	1 0 x	0 1 0
0 1 0	x 1 x	0 0 0
0 1 0	0 0 x	0 1 0
0 1 0	1 0 x	0 1 1
0 1 1	x 1 x	0 0 0
0 1 1	0 0 x	0 1 1
0 1 1	1 0 x	1 0 0
1 0 0	x 1 x	0 0 0
1 0 0	x 0 0	1 0 0
1 0 0	x 0 1	1 0 1
1 0 1	x 1 x	0 0 0
1 0 1	x 0 0	1 0 1
1 0 1	x 0 1	1 1 0
1 1 0	x 1 x	0 0 0
1 1 0	x 0 0	1 1 0
1 1 0	x 0 1	1 1 1
1 1 1	x 1 x	0 0 0
1 1 1	x 0 0	1 1 1
1 1 1	x 0 1	1 0 0

Next State Logic

$$S_2' = (\overline{S_2}S_1S_0A + (S_2\overline{S_1}\overline{S_0}E + S_2\overline{S_1}\overline{S_0}\overline{E}) + (S_2\overline{S_1}S_0E + S_2\overline{S_1}S_0\overline{E}) + (S_2S_1\overline{S_0}E + S_2S_1\overline{S_0}\overline{E}) + (S_2S_1S_0E + S_2S_1S_0\overline{E}))\overline{R}$$

$$S_2' = (\overline{S_2}S_1S_0A + S_2\overline{S_1}\overline{S_0} + S_2\overline{S_1}S_0 + S_2S_1\overline{S_0} + S_2S_1S_0)\overline{R}$$

$$S_2' = (\overline{S_2}S_1S_0A + S_2\cancelto{1}{(\overline{S_1}\overline{S_0} + \overline{S_1}S_0 + S_1\overline{S_0} + S_1S_0)})\overline{R}$$

$$\rightarrow S_2' = (S_1S_0A + S_2)\overline{R}$$

$$S_1' = (\overline{S_2}\overline{S_1}S_0A + \overline{S_2}S_1\overline{S_0}\cancelto{1}{(A+\overline{A})} + \overline{S_2}S_1S_0\overline{A} + S_2\overline{S_1}S_0E + S_2S_1\overline{S_0}\cancel{(E+\overline{E})} + S_2S_1S_0\overline{E}$$

$$S_1' = \overline{S_2}S_0(\overline{S_1}A + S_1\overline{A}) + S_1\overline{S_0}\cancel{(\overline{S_2} + S_2)} + S_2S_0(\overline{S_1}E + S_1\overline{E}))\overline{R}$$

$$\rightarrow S_1' = (\overline{S_2}S_0(S_1 \oplus A) + S_1\overline{S_0} + S_2S_0(S_1 \oplus E))\overline{R}$$

$$S_0' = (\overline{S_2}\overline{S_1}\overline{S_0}A + \overline{S_2}\overline{S_1}S_0\overline{A} + \overline{S_2}S_1\overline{S_0}A + \overline{S_2}S_1S_0\overline{A} + S_2\overline{S_1}\overline{S_0}E + S_2\overline{S_1}S_0\overline{E} + S_2S_1\overline{S_0}E + S_2S_1S_0\overline{E})\overline{R}$$

$$= (\overline{S_2}\overline{S_1}(S_0 \oplus A) + \overline{S_2}S_1(S_0 \oplus A) + S_2\overline{S_1}(S_0 \oplus E) + S_2S_1(S_0 \oplus E))\overline{R}$$

$$= (\overline{S_2}(S_0 \oplus A)\cancel{(\overline{S_1} + S_1)} + S_2(S_0 \oplus E)\cancel{(\overline{S_1} + S_1)})\overline{R}$$

$$\rightarrow S_0' = (\overline{S_2}(S_0 \oplus A) + S_2(S_0 \oplus E))\overline{R}$$

Output Logic

$\rightarrow W = \overline{S_2}$

$\rightarrow Re = S_2$

L_1

S_2S_1 \ S_0	0	1
00	0	0
01	1	1
11	1	1
10	0	0

$\rightarrow L_1 = S_1$

L_0

S_2S_1 \ S_0	0	1
00	0	1
01	0	1
11	0	1
10	0	1

$\rightarrow L_0 = S_0$

Circuit Schematic *

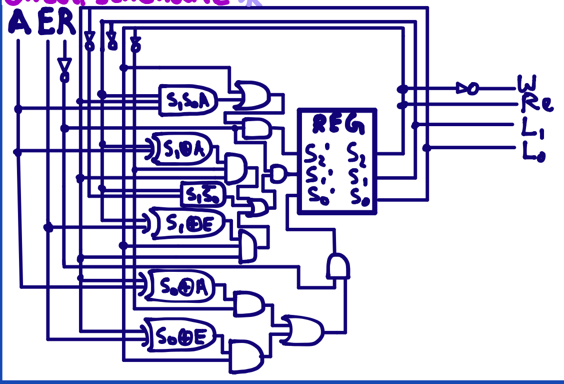


Figure 3: FSM Logic Tables and Equations, Schematic, and Output Logic
(note: this is explained in depth in the FSM Report which is a separate Blackboard submission)

Benjamin Straus
May 9, 2018
DSF Final Project
Robot Arm Report
Video Link: https://youtu.be/ygAyl9CfRJU

The first four states (S0,S1,S2,S3) are where the robot arm is recording the four stored positions. Therefore, in these states the write output is HIGH (but will be inverted when connected to the register file since it is active low) (and read is LOW) and the address is L_1L_0=00,01,10,11 respectively for the four states. Four all four of these states the input condition that R=0 and A=1 (see the legend for input descriptions) will progress the system to the next state (S0—>S1—>S2—>S3). As well, the condition on the inputs R=1 will return the system to state S0. During these first four states, the E input is not used and has no effect on the circuit. Upon being in S3, the final input condition of A=1, R=0 will bring the system to S4. This concludes analysis of the first four states with their respective inputs and outputs.

The second group of four states (S4,S5,S6,S7) is the playback stage. During these, the read output is HIGH (but inverted when connected to the active low register file) (and write is LOW) and the address again cycles through L_1L_0=00,01,10,11. During all of these states, the system progresses to the next state (S4—>S5—>S6—>S7) when E=1 and R=0. When R=1, the system returns to state S0 regardless of other inputs. During these states the input A is not used and has no effect on the function. When in state S7, the progress condition E=1 and R=0 progresses the circuit back to S4 and the circuit cycles through these four states repeatedly until the R=1 input condition is satisfied.

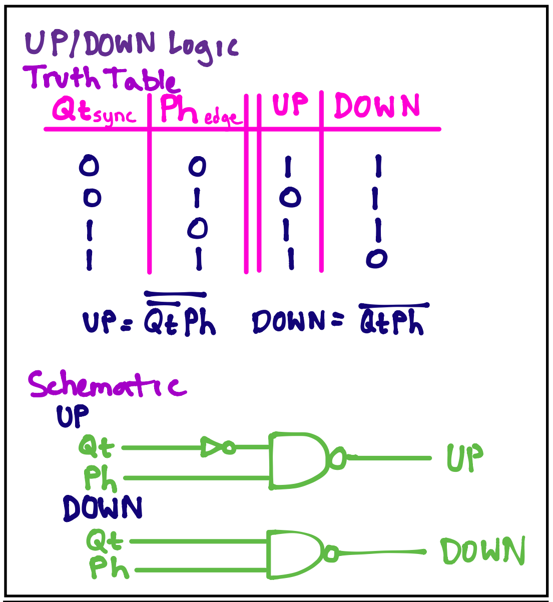
The process used design this circuit was: (1) drawing the FSM flow diagram, then (2) assigning state and correlated outputs to each state, then (3) writing the next state logic table, then (4) writing next state logic boolean equations, then (5) simplifying those equations using Karnaugh maps and boolean simplification, then (6) writing output logic equations and simplifying those, and finally (7) drawing the circuit diagram. This process (including all steps) and the equations and schematic are shown in Figure 3.

Stage 2b: Designing Other Component Circuits (Design Procedure, Simplification, Engineering Decisions)

The other components were the up/down logic (Figure 4), 12-bit counter (Figure 5), edge detection and synchronization circuit (Figure 6), debouncing and Schmitt trigger circuits (Figure 7), register file connections (Figure 8), and comparator connectors to make an 8-bit comparator out of two 4-bit comparators (Figure 9).

Most of these components required much less design than the FSM. Rather, they required reading the data sheets for the integrated circuits (ICs) used and determining which connections to make. But, a few of these components required more in depth design.

The first component is the up/down logic and is a component that required more design. The purpose of this component is to translate the Ph and Qt signals to UP and DOWN signals. The table in Figure 4 was constructed



Qt_{sync}	Ph_{edge}	UP	DOWN
0	0	1	1
0	1	0	1
1	0	1	1
1	1	1	0

$UP = \overline{\overline{Qt}Ph}$ $DOWN = \overline{QtPh}$

Figure 4: Up & Down Logic using Ph and Qt

based on analysis of the conditions on phase (Pt) and quadrature (Qt). These use active low signals for up and down which is consistent with the rest of my circuit as the 12-bit counter (discussed later) uses active low logic. From the table, I was able to derive logic equations to describe UP and DOWN. This logic was then used to convert Ph and Qt into UP and DOWN which go to the counter.

The next component, the 12-bit counter, required less design. The purpose of this counter is to record the current location of the robot arm (and occasionally write the output of the counter to a register file). The three 4-bit counters were simply connected to create a 12-bit counter by respectively connecting BO,CO output of the less significant counter to the DN,UP input of the more significant counter. The least significant counter has the DN,UP input connected to DOWN,UP from the up/down logic. The most significant counter has nothing connected to its BO,CO outputs.The not(LOAD) input is connected to the inverted universal reset button and A,B,C,D are the load values and are connected such that 1000000000 is the “origin” location of the system. This value is used because it allows for the most possible values in either direction (clockwise or counterclockwise) as it is the most centered number in a 12-bit binary number

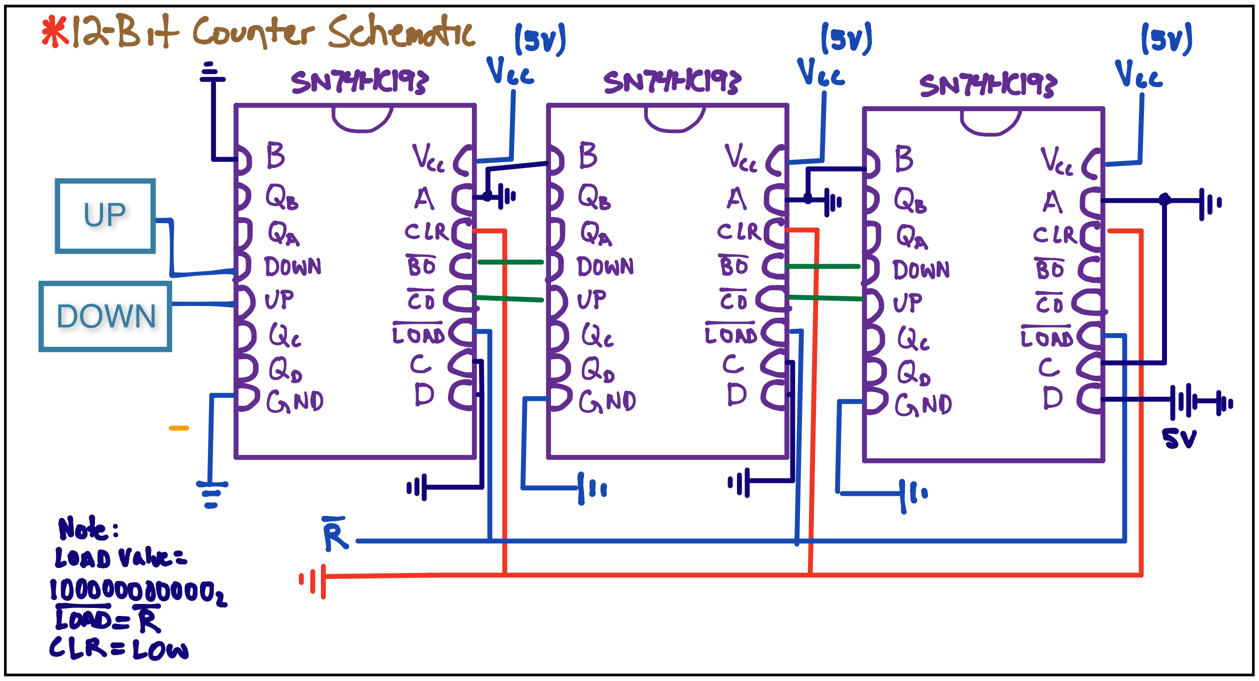


Figure 5: 12-Bit Counter Component Schematic

(2048_2 out of 4095_2). This way, the system can record the largest range of locations in either direction. The CLR input is constantly ground because the load input will do the work of reseting the counter. The wiring for the 12-bit counter is in Figure 5.

The next component is the edge detection and synchronization circuit which was another than required a bit of design. The purpose of this is to synchronize the signals from the phase, quadrature, and action button inputs of the entire system. Then the edge detection is used on the

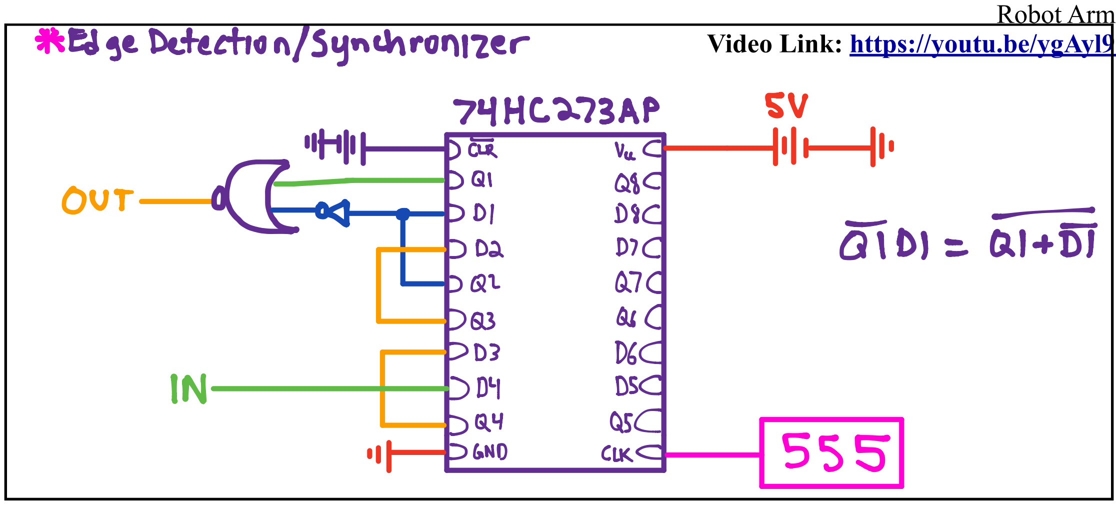

*Edge Detection/Synchronizer
74HC273AP
5V
OUT
IN
CLR
Q1
D1
D2
Q2
Q3
D3
D4
Q4
GND
Vcc
Q8
D8
D7
Q7
Q6
D6
D5
Q5
CLK
555
$\overline{Q1}D1 = \overline{Q1 + \overline{D1}}$

Figure 6: Phase/Quadrature/Button Edge Detection & Synchronization Schematic

button and the phase input so that a single pulse is the only thing seen by the system. This is required so that holding the button or a constant signal from phase is only seen as one press or tick rotation of the motor. The quadrature input is only required to be synced (not edge detected) because of the up/down logic needed (see Figure 4). And, the reset button doesn't need to be synchronized or edge detected as reseting multiple times or at any time isn't an issue. To accomplish this, the circuit used a series of connections within a single flip-flop chip to cascade the signal through the flip-flop, thus synchronizing the signal. Then, to edge detect the signal, a NOR gate on the final two connections of the flip-flop. The exact setup is shown in Figure 6.

The next component of the system is the debounce and Schmitt Triggers. These were used only on the action button so that the signals from the button do not fluctuate when the button is released. If these weren't used, a single press of the button could read as multiple presses which would cause malfunctioning of the circuit. The reset button did not require the use of these because if the reset button is pressed, it doesn't matter if it resets multiple times. The debouncing circuit simply uses a capacitor in parallel with the button so that the output is smooth (rather than "bouncing"). The Schmitt trigger serves a similar purpose and was done with an IC. It was simply connected and input to the input

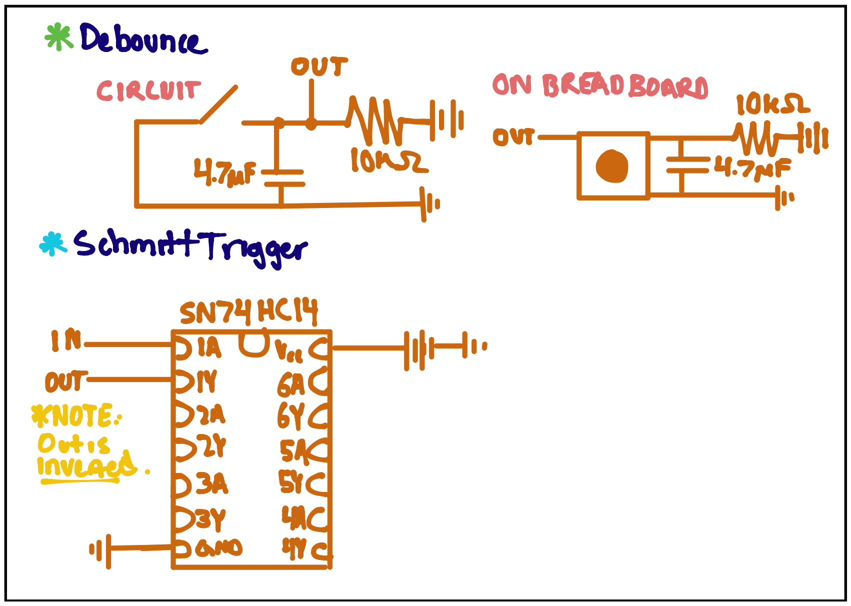


Figure 7: Button Debouncing & Schmitt Trigger Schematic

of trigger 1 and the output to the output of trigger 1. The exact wiring is shown in Figure 7.

The next component is the register file which is used to store the locations recorded in states S0, S1, S2, and S3. However, each location has 12 bits, but each register file only has 4 bits. I chose to store the most significant 8 bits which will provide a high enough level of accuracy (as we have 0.26 degrees per pulse, making for ±4 degrees). The calculations for this come from a 200 steps/revolution motor, 27 is the gear ratio, and 4 is a natural divider in measuring. This give 200*27/4=1350 pulses/revolution or 0.26 degrees per pulse. Since I stored 8 bits, I needed two 4-bit counters. By analyzing the data sheets, I saw that I needed to connect the respective counter to the Q_{A-D} input bits and the respective comparator to the Q_{1-4} output bits. I also needed to connect the same FSM address bits L_1L_2 to the R_A, R_B, W_A, W_B address bits of both registers. Lastly, I connected the W and R output from the FSM (but inverted) to the $\bar{G}_W$ and $\bar{G}_R$ inputs of the register file. The exact connections are shown in Figure 8.

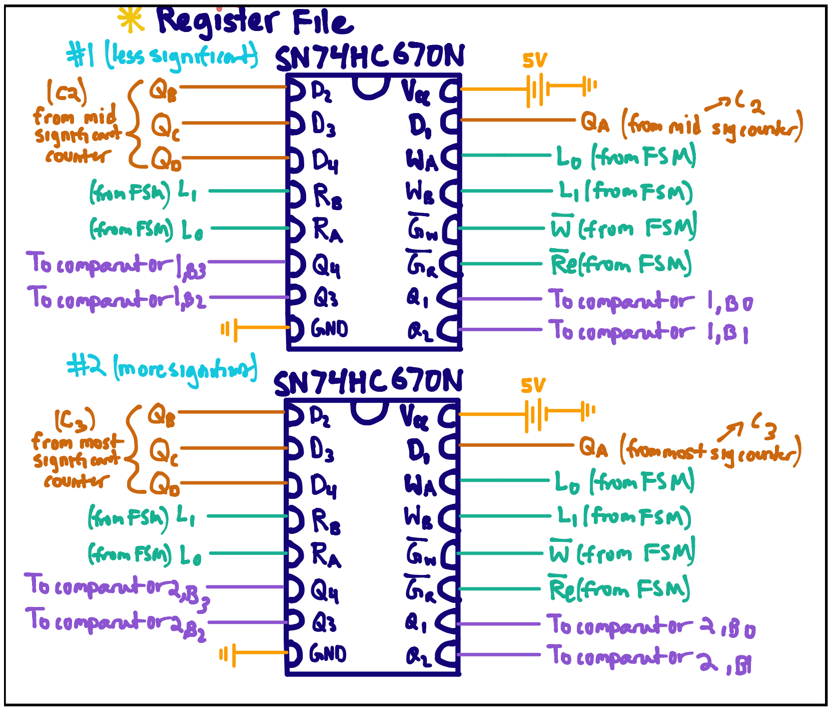


Figure 8: Register File Schematic

The final extra component is the 8-bit comparator, which much like the register file required just looking at the data sheet. This was a bit difficult because I had to concatenate two 4-bit comparators. This required some connections between the comparators as the output of the entire comparator could depend on either comparator depending on how different the numbers are. The purpose of the comparator is to determine which way the motor must rotate to reach the desired position and to determine when it has reached that position. This was done using the output from the A<B output of the most significant comparator (which takes into account the less significant one) and the A=B output of the most significant comparator (also which takes into account the less significant one). The A<B output is high when A<B and low when A>=B. So, if that is the direction bit outputted to the motor, then the motor will turn correctly. Note that the A=B case won't matter because the A=B output becoming high (which is sent to the FSM E=equal input) will trigger the FSM to progress states and start moving the motor to the next position. So, technically, the motor never will stop turning, but this is not an issue because there will be a new position to turn to as soon as the motor reaches the previous goal. One other remark is that the choice of using the A<B output is arbitrary and is based on whether DIR=1 corresponds to clockwise or counterclockwise. This case be determined post implementation and

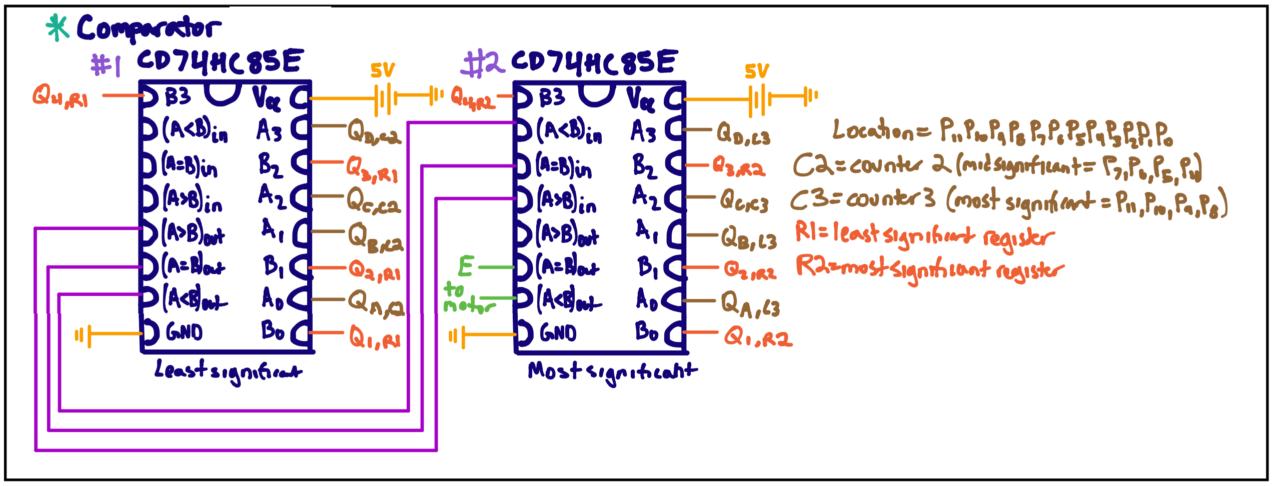


Figure 9: 8-Bit Comparator Schematic

easily switched by changing to the A>B output if the wrong bit is chosen. The wiring done for the comparator requires inputs from both the register (desired location) and counter (current location). As well, it has the $(A>B)_{out}$, $(A=B)_{out}$, and $(A<B)_{out}$ from the less significant comparator connected to the corresponding "in" pins on the more significant comparator to concatenate the outputs. The exact wiring is shown in Figure 9.

This concludes the analysis of each individual component's design and schematic. Next, each sub component is tested to ensure that it will work as desired.

Stage 3: Simulating the FSM & Testing Other Components (All Inputs, Outputs Tested)

The first part of the circuit that needed to be simulated was the FSM. This component required the most logic that was completely self-fabricated (as the other components required almost sole reliance on data sheets). So, it had potentially the most fundamental flaws. This being said, it was critical to be simulated in Logism prior to implementation. The testing for the FSM was explained in the FSM simulation report, but is pasted here for convenience. I tested the circuit by going through each state and ensuring that:

- The correct input conditions progress the state (A=1,R=0 for S0-S3; E=1,R=0 for S4-S7; R=1 return any state to S0)
- An inputs that should not effect the state do not (E=1 or 0 for S0-S3; A=1 or 0 for S4-S7)
- The correct outputs are set for each state (see the state encoding and output table in the diagram for correct outputs)

The Logisim circuit schematic is shown in Figure 10. A video demonstrating the behavior of the simulation is available at: https://youtu.be/SSdCTM96p20.

Next, I had to test each of the individual additional components. Since these were designed step by step and the error involved would mostly be in incorrect assumptions/analysis of the pins of various integrated circuits, the most efficient method of testing was by

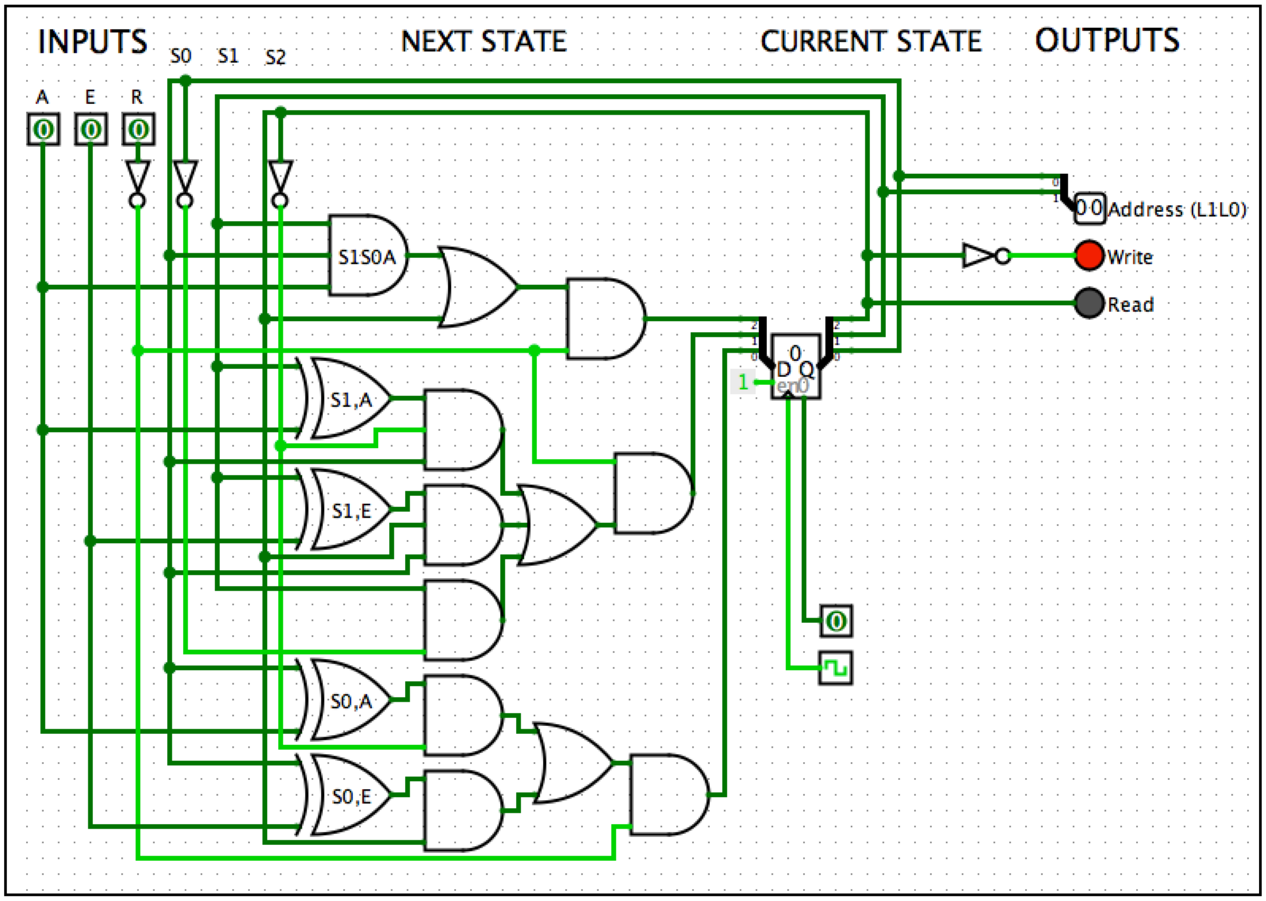


Figure 10: FSM Simulation Schematic

implementing these separately on a breadboard. In each case, the inputs and outputs could be set to see if the correct behavior is obtained. The testing for each component is explained below:

- 12-bit counter (see Figure 11)
    - Connected least significant counter's UP to a clock and DOWN to V_{cc}.
    - Read outputs at the first, fourth, eighth, and twelfth bit position with LEDs.
    - Saw that counting occurred and incremented as desired.
- Edge Detection/Synchronization (see Figure 12)

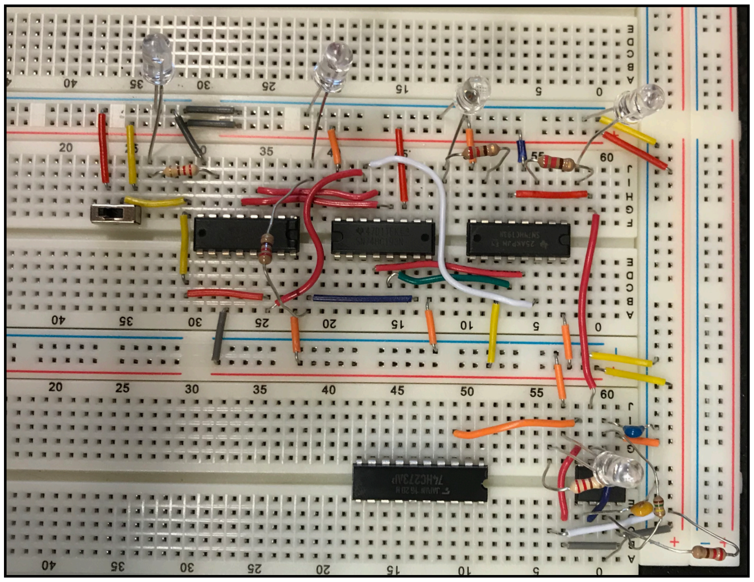


Figure 11: Counter Test Circuit

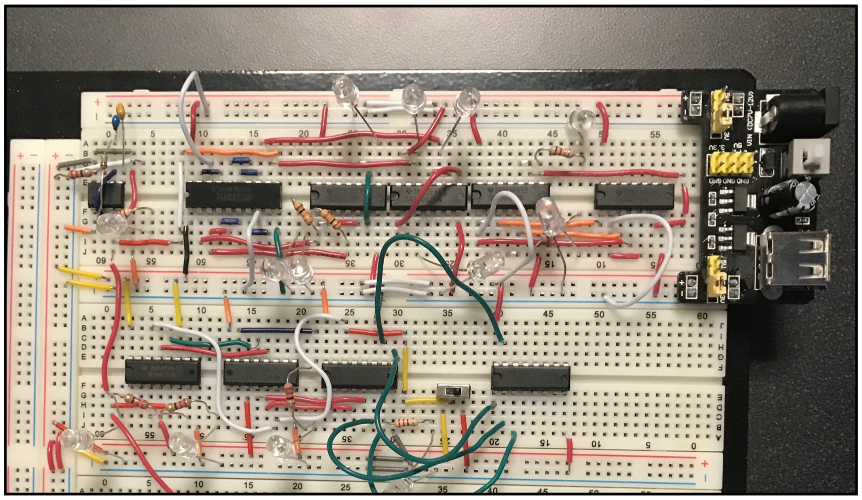


Figure 12: Edge Detection & Synchronization Test Circuit

  - Connected to a button and a clock
  - Pressed the button
  - Observed that for a press of the button (whether held down or released), the result is one single, steady pulse is observed at the output. This demonstrates edge detection as we see a single pulse and synchronization because that pulse is steady thus in count with a clock.
- Debouncing/Schmitt Trigger
  - Connected to a button
  - Used an oscilloscope to see the oscillations the result from pressing a button. The desired outcome was very small to no oscillations when the button is pressed.
- Register-file
  - Connected to various hard wired position at the input pins
  - Pulse the write input low (recall: active-low logic) to write the input at the input pins to a hardwired memory location.
  - Hold the read input low (recall: active-low logic) and use LEDs to read the data stored at the hardwired memory location
- Comparator
  - Hardwired two binary numbers to the 8-bits available
  - Used LEDs to observe the outputs at the A=B,A<B,A>B output pins of the most significant comparator (recall that the most significant comparator output takes into account the number passed to the less significant one by having connections between the comparators; see Figure 9)
- Up/Down Logic
  - Connect the edge-detected and synchronized phase output and the edge-detected quadrature output of the motor to the logic, and observed the up and down output values. When the motor turns clockwise, UP should pulse “1-1-1-0-1-1-1”. When the motor turns counterclockwise, DOWN should pulse “1-1-1-0-1-1-1”. Recall that the logic is active-low.

This concludes the testing and simulation of all individual components of the circuit. According to the original hypothesis since each individual component has been tested and works, then the system is expected to behave as desired. The next stage involves implementing the FSM on the breadboard

Stage 4: Implementing the FSM

This stage requires little discussion having described the function and design of the FSM in previous sections. However, an important note is that the availability and function of various ICs changed which ones were used. For example, when ANDs were not available or when there were many NANDs free on the breadboard, I used a NAND and inverted the output (by NANDing the output with itself). This made it so that I simplified my design and was as space conscious as possible so that the breadboard didn't run out of space.

Stage 5: Implementing and Connecting the Additional Components

After implementing my FSM, I then built and tested each individual additional component. Once they were tested (as described in Stage 3), I connected them to each other as shown in Figure 1. The final layout of the breadboard is shown in Figure 13. Note that a very specific color convention was used. As well, only the wiring that travels from one sub-system to another arches above the plane of the breadboard (for example, clock outputs [red wires in the middle] and counter to register [yellow wires at the top]). Any intra-component wiring is flat (for example, the FSM next state wiring).

Stage 6: Testing System (Inputs, Outputs, Behavior Tested)

The very final stage is testing the completed circuit. This was done by changing each input to observe every possible case and observe the output to ensure the desired behavior is obtained. The various cases (not including those described in Stage 3 for the FSM) are explained below:

- Action: Reset pressed, [motor moved, action pressed] x 4 ($S_2S_1S_0$=000,001,010,011)
- Result: Motor begins moving, first to the motor's location when the action button was first pressed, then to the second, and so on. ($S_2S_1S_0$=100,101,110,111)
- At any point in the cycle, pressing the reset button returns the system to the reading state and it awaits the pressing of the action button

In order to know whether the system was in the right state at a given point, I used LEDs (the three most down and to the left in Figure 13 & 14) to represent the state. I saw that in record mode, the S_2 bit was always 0 while in playback it was 1. This is as expected. Further, I could see that S_1S_0 represented which stored value the motor is recording (in the recording states) or which one it is moving to (in the playback states). The complete testing is executed and explained in the video here: https://youtu.be/ygAyl9CfRJU.

Problems Faced:

- In simulating my FSM, I found unpredictable behavior that resulted from R being 0. I found that this was because I had at first neglected to include R=0 in my state progression condition. This was easily fixed by adding that condition which just required using a single AND gate with R' and the existing output.

- In the initial testing phase, I observed very strange clock behavior (I observed this with an oscilloscope). I found that this was the result of a short in my circuit that was the result of not powering a few of my ICs (since they weren't being used yet, but were connected to outputs). This was easily fixed once I connected those ICs properly.
- At first, I both cleared and loaded values to the counter which resulted in an indeterminate state since I may have either loaded or reset last (depending on whether CLR or LOAD was active more recently). This was easily fixed by wiring CLR to ground and have only LOAD used to "reset" (to preset value) by connecting to reset.
- There were a few wiring errors in the FSM when first implemented. This was easily debugged by seeing which state bits were misbehaving when the action button was pressed (i.e. they didn't progress from $S_2S_1S_0$=000 to 001 to 010 to 011, and so on). I found that I had wired one output to pin 8 rather than 7. I also found that I had inverted an input because I used a NAND rather than AND and forgot to invert the output. These were both fixed once the problem was identified by making the correct connections.

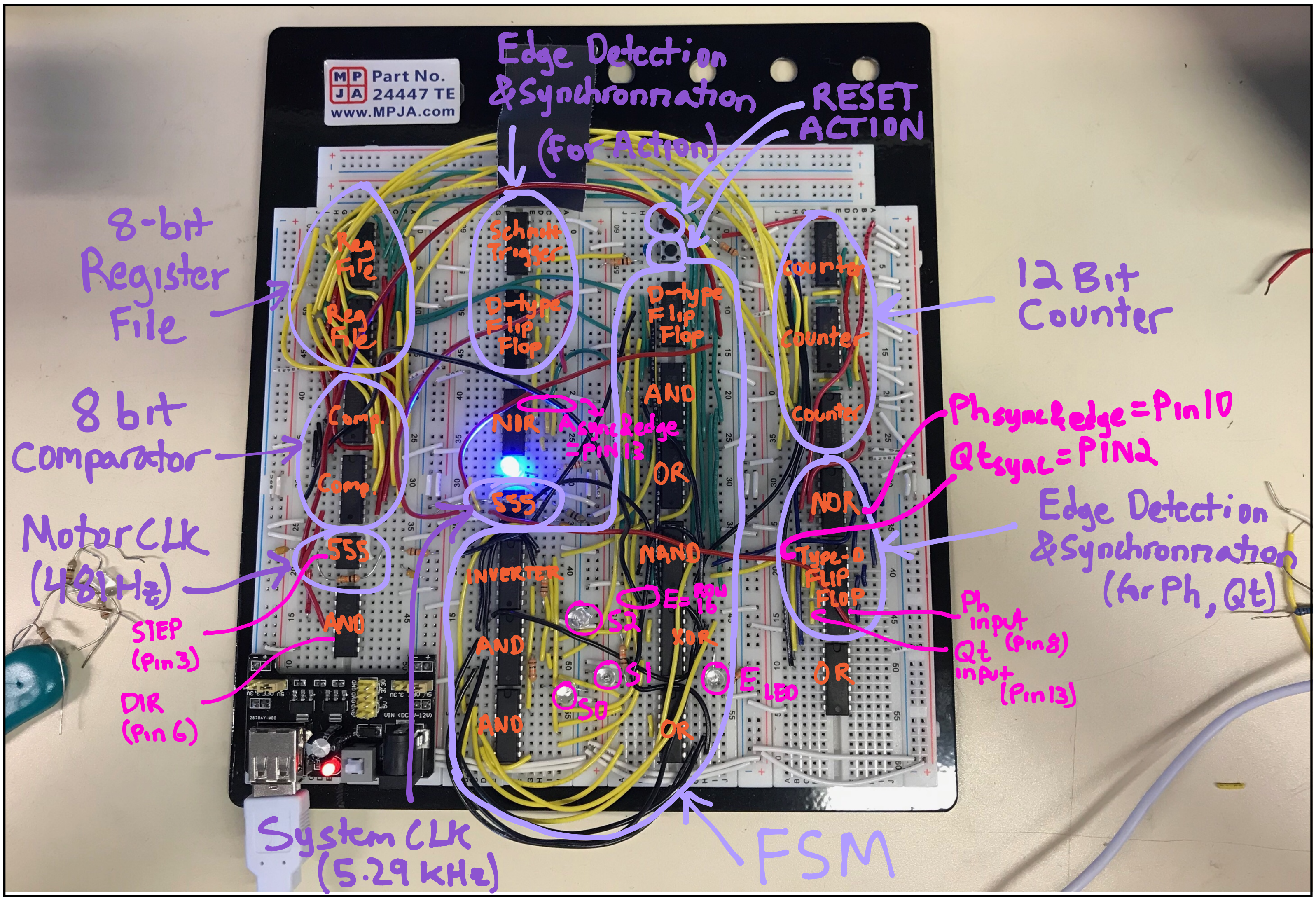


Figure 14: Final Implemented System (labeled for components)

Benjamin Straus
May 9, 2018
DSF Final Project
Robot Arm Report
Video Link: https://youtu.be/ygAyl9CfRJU

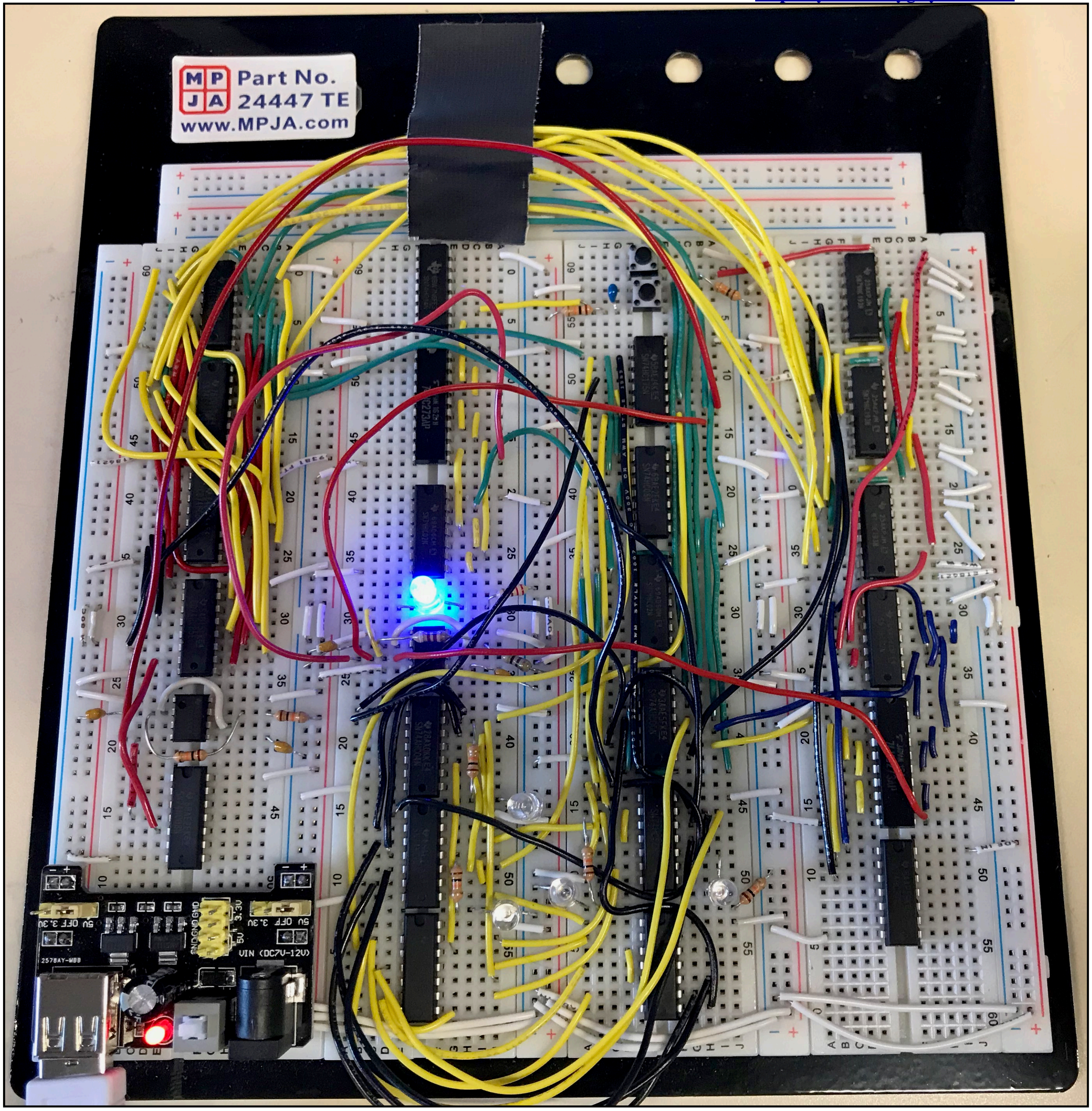


Figure 13: Final Implemented System (unlabelled to show wiring)
White = V_{cc} & Ground
Column 1 top half (Registers): [Red] Register to comparator; [Yellow] Counter to register; [Green] FSM to register
Column 1 bottom half (Comparators): [Black] Comparator to Comparator; [Yellow] Counter to Comparator; [Red] Reg to comparator
Column 3 (FSM): [Green] S_2 next state logic; [Yellow] S_1 next state logic; [Black] S_0 next state logic
Red over arching wires: System clock output
Black & Red over arching wires: State bits from FSM to other sub-components