Abstract

The main design principles in this experiment were to use the sequential logic design that we used in the lectures, and put it to some good use. We first part was to make the logic that would make the stepper motor turn with two external controls, a GO/$\overline{\text{STOP}}$ and a UP/$\overline{\text{DOWN}}$. This part was basically to take a state transition diagram, and turn this into a truth table and a state excitation table. The logic was then determined through minimising the equations and then constructing the actual circuit. The next part was to make the stepper motor turn through 30° and stop for 5 seconds, then turn back through another 30°, and repeat. The motor worked almost as planned, instead of turning through 30°, it turned through 28° instead. Apart from that, it worked the way it should.

Contents Page

Introduction

The main component of the experiment was the stepper motor. It turned through 2° when the correct logic was applied to the windings. The stepper motor has four windings, which may be turned on and off, to make the motor turn. The axis upon which the motor turns is permanently magnetised, with the four windings around the outside, each winding consisted of 45 coils. Hence there are 180 poles, and each step is 2°. When power is applied to a specific winding, it asserts a magnetic force on the rotor, pulling it words the coil, in a circular fashion. Stepper motors are found in many everyday situations, for example floppy drives, flatbed scanners, printers and plotters. The stepper motor uses a Darlington Pair and TTL transistors to provide the 7.4V and 1.3A needed to drive the motor. The Darlington Pair is a single device that combines two bipolar junction transistors. The reason this is used is because the gain is very high and takes up less space than using two discrete transistors. The Darlington Pair is named after the inventor of the configuration, Sidney Darlington while working at Bell Labs. Modern integrated circuits use this idea, but using many transistors.

To make the motor step forward (UP) the following logic has to be applied to the windings:

W4	**W3**	**W2**	**W1**
1	0	1	0
1	0	0	1
0	1	0	1
0	1	1	0

The logic required to make the stepper motor do the GO/$\overline{\text{STOP}}$ and a UP/$\overline{\text{DOWN}}$ controls was quite a lengthy process. First a state transition diagram was constructed. The truth table was constructed and a state excitation table was made. From the state excitation table, the logic was determined. These logic functions were simplified with Karnaugh maps. The logic functions for the four windings were then constructed on the breadboard, and tested. It was found to be working correctly.

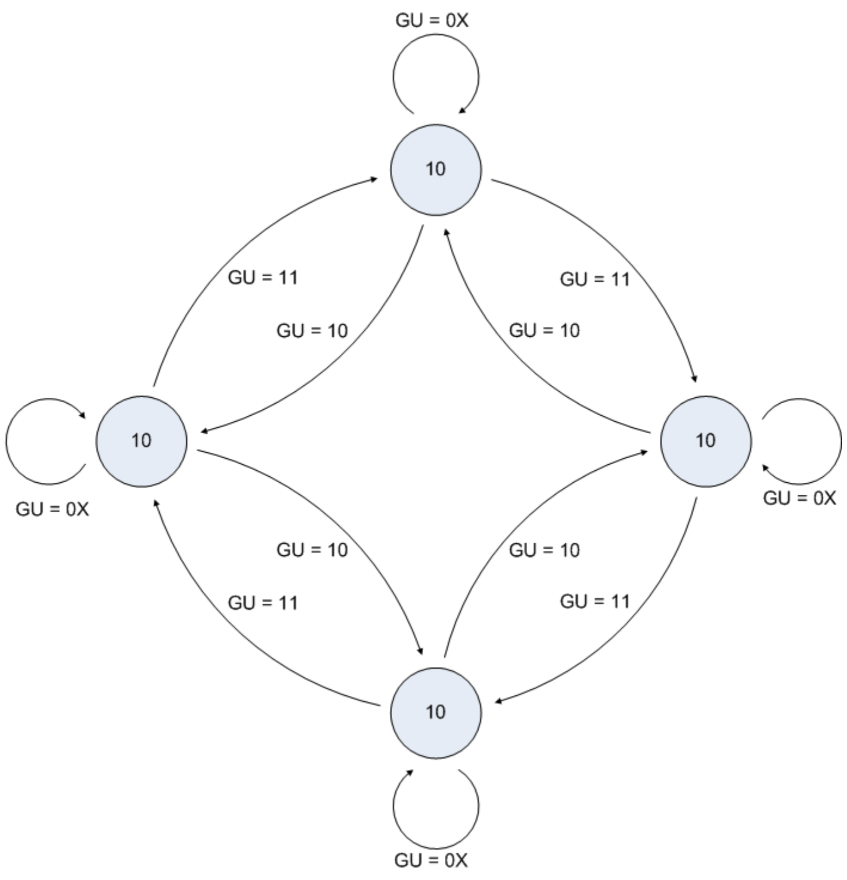
The next part was to make the motor step through a sequence of movements. The sequence was go forward (UP) for 30°, stop for 5 seconds, and then turn backward ($\overline{\text{DOWN}}$) for another 30°. The implementation of this was reasonably straight-forward. The motor had to turn through 30°, so that was 15 steps (as each step was 2°). Once the count reached 15, the clock was stopped and the motor would wait for 5 seconds. Then, depending on the direction it turned last, it would turn back to the original position. The external GO/$\overline{\text{STOP}}$ would be kept, but the UP/$\overline{\text{DOWN}}$ control would be removed, as the motor would control itself.

Design Principles

First of all, the individual state labels for the windings needs to chosen. These labels are chosen at random.

W4	W3	W2	W1	State
1	0	1	0	00
1	0	0	1	01
0	1	0	1	11
0	1	1	0	10

From here, the state transition diagram was constructed, this is shown below:



From this diagram, the truth table and state excitation table were constructed:

Inputs		**Present**		**Next**		**Requires**		**Present**			
G	**U**	**Q_1^n**	**Q_0^n**	**Q_1^{n+1}**	**Q_1^{n+1}**	**D_1**	**D_0**	**W4**	**W3**	**W2**	**W1**
0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	1	0	1	0	1	1	0	0	1
0	0	1	0	1	0	1	0	0	1	1	0
0	0	1	1	1	1	1	1	0	1	0	1
0	1	0	0	0	0	0	0	1	0	1	0
0	1	0	1	0	1	0	1	1	0	0	1
0	1	1	0	1	0	1	0	0	1	1	0
0	1	1	1	1	1	1	1	0	1	0	1
1	0	0	0	1	0	1	0	1	0	1	0
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	1	1	1	1	0	1	1	0
1	0	1	1	0	1	0	1	0	1	0	1
1	1	0	0	0	1	0	1	1	0	1	0
1	1	0	1	1	1	1	1	1	0	0	1
1	1	1	0	0	0	0	0	0	1	1	0
1	1	1	1	1	0	1	0	0	1	0	1

We chose to implement the logic using D Flip-flops, because it was thought that this would lead to the simplest result.

Now the logic for the windings and for the inputs for D_1 and D_0 functions were minimised with Karnaugh mapping. The results are shown below.

D_1:

Q_1Q_0 \ **GU**	11	01	00	10
11	x	x	x	
01	x			
00				x
10		x	x	x

$$\begin{aligned} D_1 &= GUQ_0 + \bar{G}Q_1 + G\bar{U}\bar{Q}_0 \\ &= G(UQ_0 + \bar{U}\bar{Q}_0) + \bar{G}Q_1 \\ &= G(\overline{U + Q_0}) + \bar{G}Q_1 \end{aligned}$$

D_0:

Q_1Q_0 \ **GU**	11	01	00	10
11		x	x	x
01	x	x	x	
00	x			
10				x

$$\begin{aligned} D_0 &= GU\bar{Q}_1 + \bar{G}Q_0 + G\bar{U}Q_1 \\ &= G(U\bar{Q}_1 + \bar{U}Q_1) + \bar{G}Q_0 \\ &= G(U + Q_1) + \bar{G}Q_0 \end{aligned}$$

To make the circuit easier to implement, D_1 and D_0 are changed to:

$$D_0 = \overline{\overline{\bar{G}Q_1} \cdot \overline{G(U + \bar{Q}_0)}}$$

$$D_1 = \overline{\bar{G}Q_0 \cdot \overline{G(U + Q_1)}}$$

$$\begin{aligned} D_1 &= GUQ_0 + GQ_1 + GUQ_0 \\ &= G(UQ_0 + UQ_0) + GQ_1 \\ &= G(U + Q_0) + GQ_1 \end{aligned}$$

$$\begin{aligned} D_1 &= GUQ_0 + GQ_1 + GUQ_0 \\ &= G(UQ_0 + UQ_0) + GQ_1 \\ &= G(U + Q_0) + GQ_1 \end{aligned}$$

The logic for the windings was next determined, again with Karnaugh mapping. The results are shown below:

W4:

		GU			
		11	01	00	10
Q_1Q_0	11	x	x	x	x
	01	x	x	x	x
	00				
	10				

$W4 = GQ_1 + GQ_1 = Q_1$

W3:

		GU			
		11	01	00	10
Q_1Q_0	11				
	01				
	00	x	x	x	x
	10	x	x	x	x

$W3 = GQ_1 + GQ_1 = Q_1$

W2:

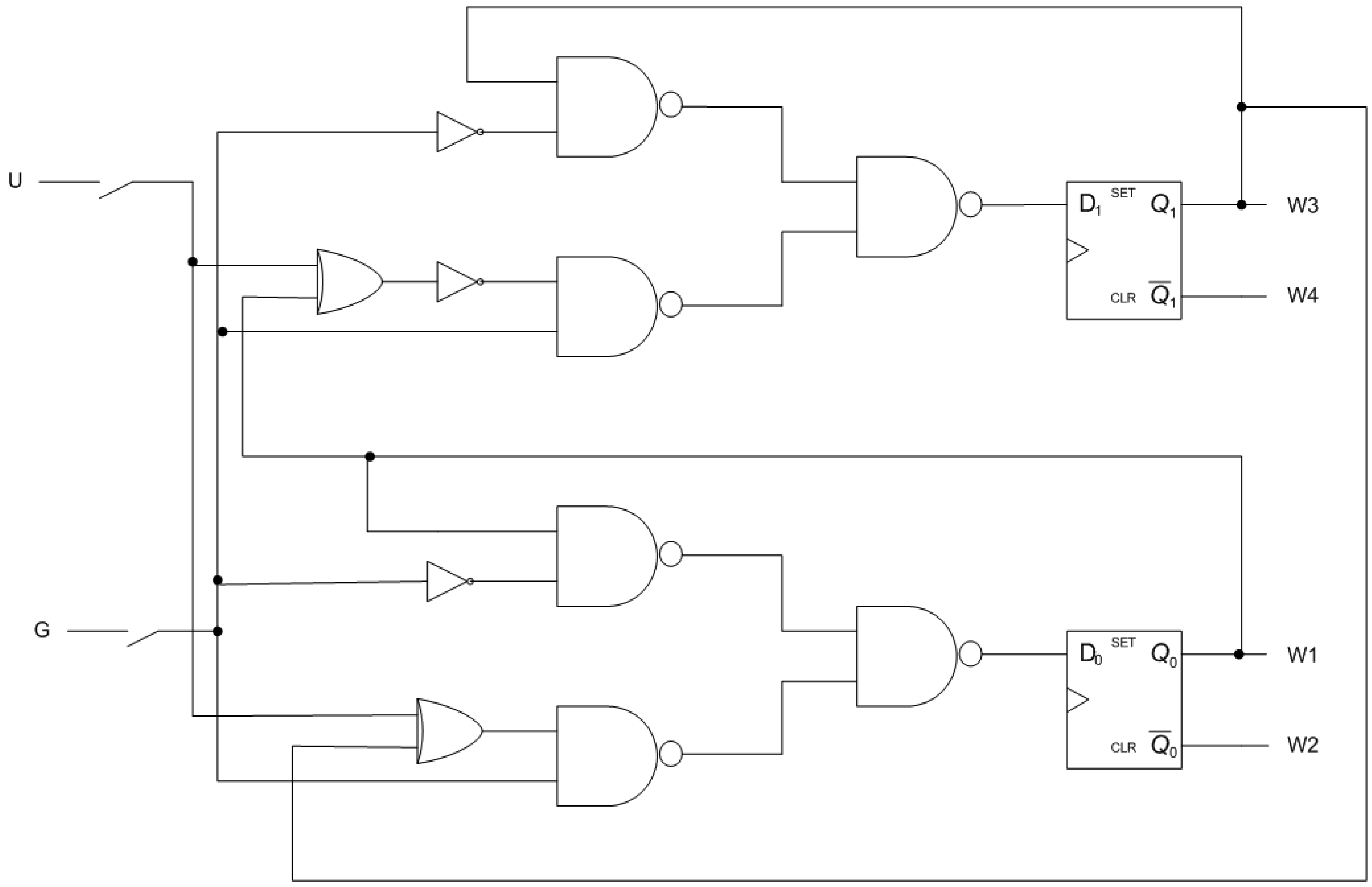
		GU			
		11	01	00	10
Q_1Q_0	11	x	x	x	x
	01				
	00				
	10	x	x	x	x

$W2 = \overline{Q_1}\,\overline{Q_0} + Q_1\overline{Q_0} = \overline{Q_0}$

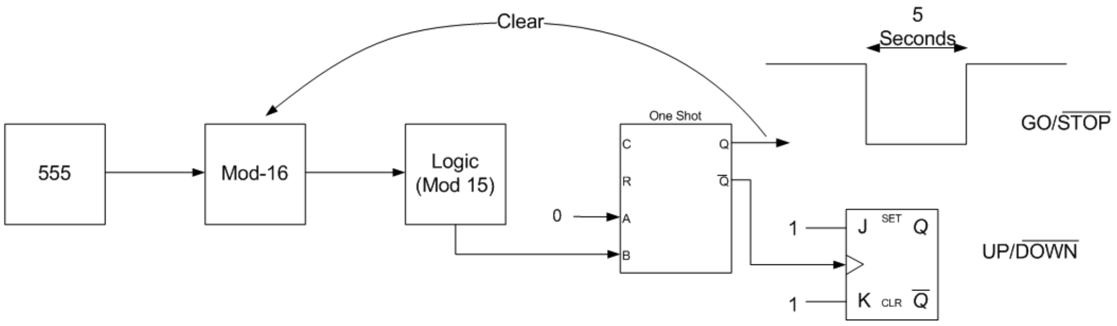
W1:

		GU			
		11	01	00	10
Q_1Q_0	11				
	01	x	x	x	x
	00	x	x	x	x
	10				

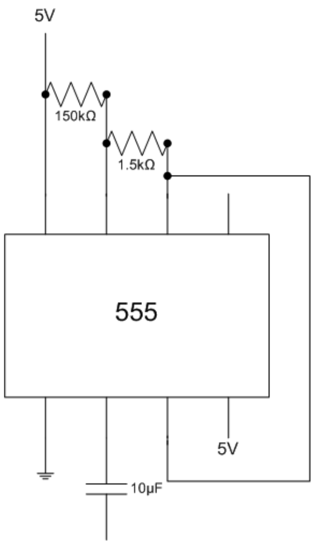
Now that the logic has been determined, the circuit can be created. This is shown below:


U
G
D_1
SET
Q_1
CLR
$\overline{Q}_1$
W3
W4
D_0
SET
Q_0
CLR
$\overline{Q}_0$
W1
W2

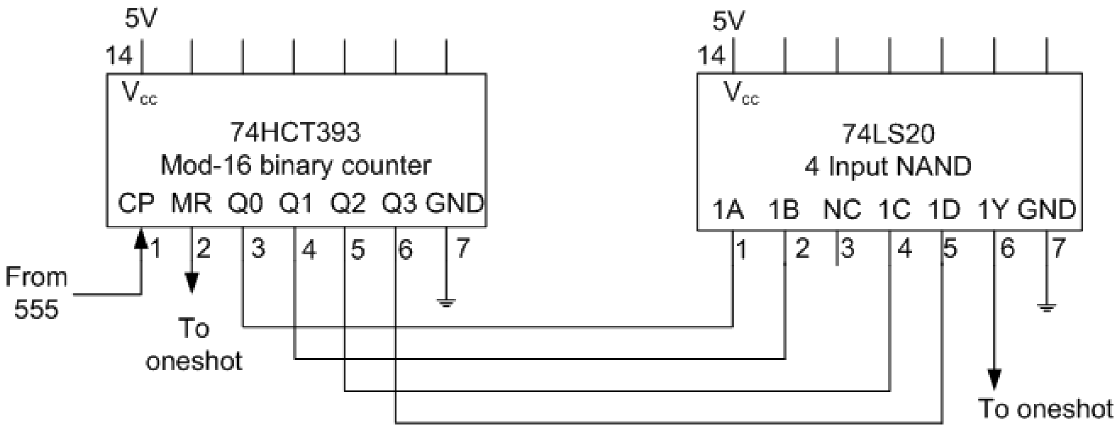
The logic for the first part was implemented and tested, and found to be working fully correctly. The next part was then implemented. The schematic diagram is shown below:



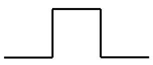
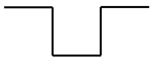
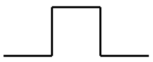
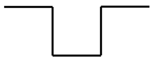
The 555 clock (LS 555) has external resistor and capacitors to set the time taken for each pulse. These values were set so as make the clock cycle one second. This was done by using 150kΩ and 1.5kΩ resistors and a 10µF capacitor. The block diagram is shown below:

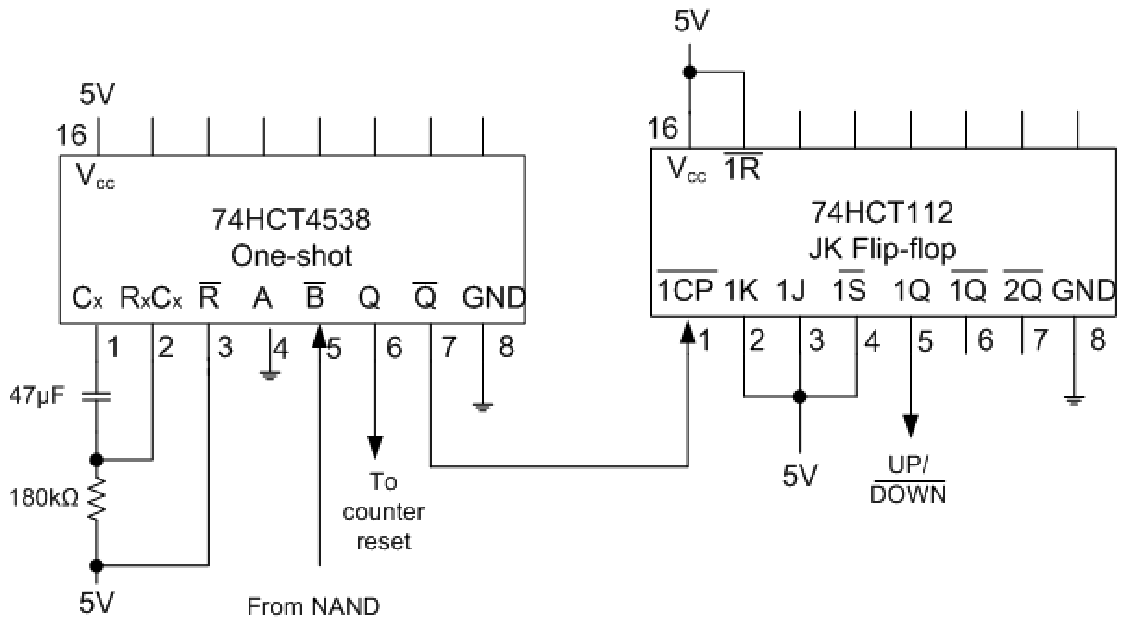


The next component to be used was the mod-16 counter. This was the 74HCT393 Dual 4-bit Binary Counter. This was for counting the 15 steps, when the motor would stop for 5 seconds. As we only wanted 15 steps, not 16 from the counter, a 4 input NAND gate was used to clear the counter when it got to 1111, thus producing 15 counts from 0000 (0) to 14 (1110). The block diagram is shown below of the counter and logic to make it a mod-15 counter.



The next section to be implemented was the one-shot. This component controlled the actual action of the motor. It also had external resistor and capacitor values which control the time the one-shot stays at low. When the one-shot goes low, the motor stops, and thus the capacitor and resistor values are chosen to make the time stopped 5 seconds. The resistor value was 180kΩ and the capacitor was a 4.7µF. These values were worked out from the time constant of a capacitor, T = RC. The complement of the output of the one-shot, $\overline{Q}$, goes to a JK flip-flop to determine whether the next direction should be UP or DOWN. The JK flip-flop is in toggle mode, as both inputs are 1. This means that whatever the last direction was, it turns the opposite next. This is the part actually controlling the total action of the motor, both if it moves at all and if so, which way it moves. The truth table for the one-shot and diagram are shown below.

Inputs			**Outputs**	
$\overline{\textbf{R}}$	**A**	$\overline{\textbf{B}}$	**Q**	$\overline{\textbf{Q}}$
L	X	X	L	H
X	H	X	L	H
X	X	L	L	H
H	L	↑		
H	↓	H		



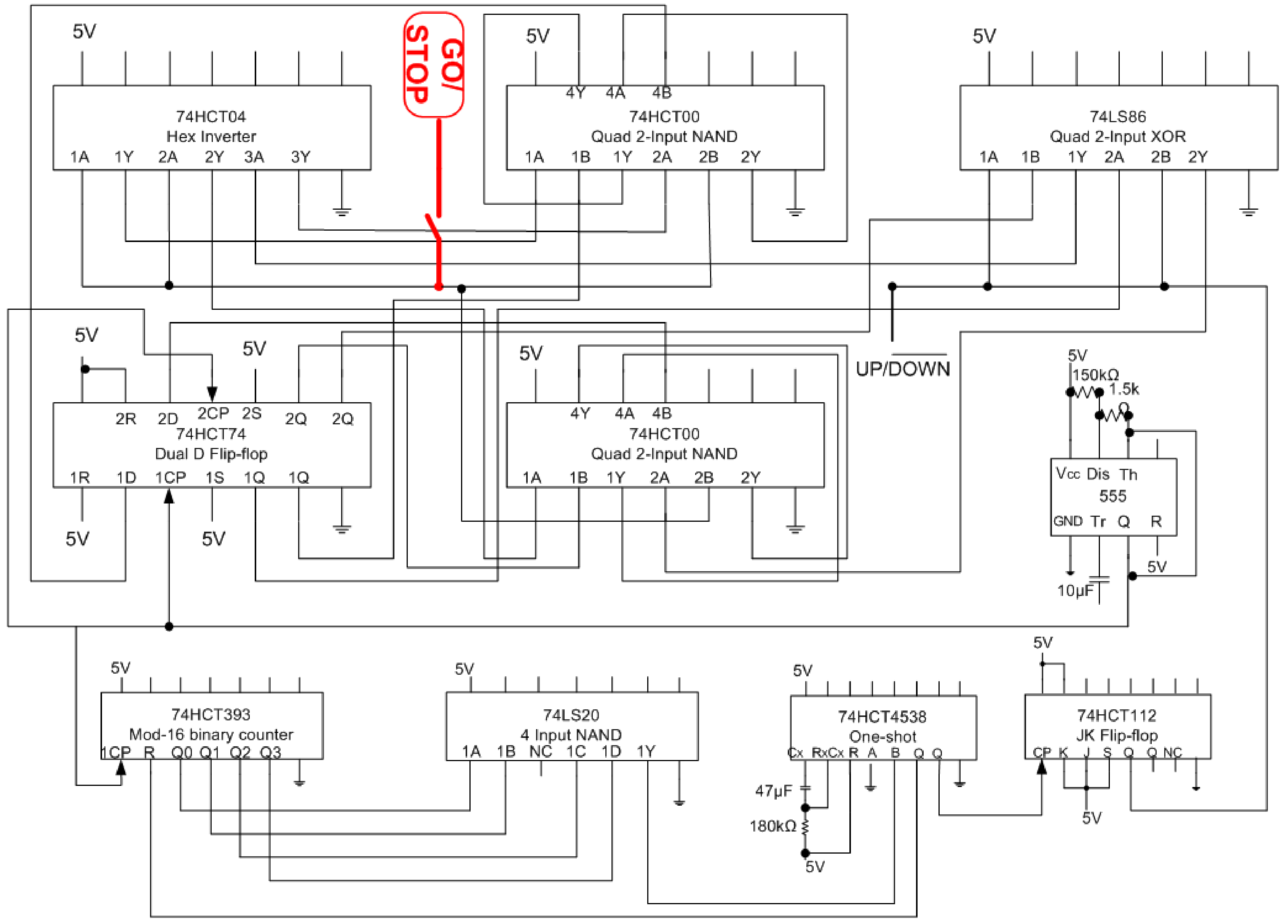
The circuit is now complete, with each component built individually and tested to be working correctly before combining it with the rest of the circuit. The operation is quite simple. When the one-shot is low, the motor stops for 5 seconds because of the external capacitor and resistor. When 5 seconds is up, the one-shot goes HI, and the motor starts to turn. When a count of 15 is up on the binary counter, the motor now stops. When another 5 seconds is up, the motor starts to turn backwards due to the JK Flip-flop in toggle mode, for another 15 steps. While the external GO/$\overline{\text{STOP}}$ control is HI, this keeps happening.

Performance and Discussion

Our circuit didn't quite work as expected. Instead of stepping 15 steps between stoppings, it stepped only 14 times; even though we cleared the counter at 1111 (i.e. 0-14 is 15 steps). The demonstrator checked our circuit and our design and found nothing wrong. My partner and I spent a lot of time checking the circuit and components and found nothing wrong either. One possibility is the propagation delay in passing through the gates, but this is a remote possibility as the clock was only going a 1Hz. Other than that, the motor worked the way it was designed to do. The time that the motor was stopped was 5 seconds, due to the correct values of capacitor and resistor.

Apart from the number of steps problem our design worked perfectly. We had spent quite a time getting the logic as simple as possible, to make the circuit easier to construct and trouble-shoot. Also, in the real world, if you can make a design with the least amount of gates possible, it will cost less, thus making it more desirable for consumers. For example, we used De Morgan's theorem $\overline{A + B} = \overline{A} \cdot \overline{B}$ to change AND gates into OR gates to make the number of chips needed a minimum. This is a double edged sword because it uses less gates, but makes the logic harder because rather than having a AND gate, you have to have an OR and an inverter, which makes harder to check it for correctness. The logic of the first section to get the motor working, with the two external controls worked perfectly. That circuit was changed very slightly, with only the control configuration changed, for the next section.

This section where the motor had to turn 30° and stop for 5 seconds was more challenging. The idea behind the design was reasonably simple, but implementing the logic needed was quite difficult. The main challenge was trying to work out why the motor was only stepping 14 times instead of the required 15. The rest of the design was quite straight-forward. According to the demonstrator, the way the circuit is constructed is correct, but it doesn't work correctly. So if this product was being built for a client, this wouldn't be acceptable. A way to make the circuit work correctly would be to have a mod-32 counter and clear the counter when it reached 10000. This would be more expensive however because a 5-bit counter is more complex than a 4-bit. Also it should work on the mod-15 counter. I am confident that if a mod-32 counter had been used, the motor would function as designed.


5V
74HCT04
Hex Inverter
1A 1Y 2A 2Y 3A 3Y
GO/STOP
74HCT00
Quad 2-Input NAND
4Y 4A 4B
1A 1B 1Y 2A 2B 2Y
74LS86
Quad 2-Input XOR
1A 1B 1Y 2A 2B 2Y
74HCT74
Dual D Flip-flop
2R 2D 2CP 2S 2Q 2Q
1R 1D 1CP 1S 1Q 1Q
UP/DOWN
150kΩ
1.5k
Vcc Dis Th
555
GND Tr Q R
10μF
74HCT393
Mod-16 binary counter
1CP R Q0 Q1 Q2 Q3
74LS20
4 Input NAND
1A 1B NC 1C 1D 1Y
74HCT4538
One-shot
Cx RxCx R A B Q Q
47μF
180kΩ
74HCT112
JK Flip-flop
CP K J S Q Q NC