Sensor-less brushless DC(BLDC) motor control project:

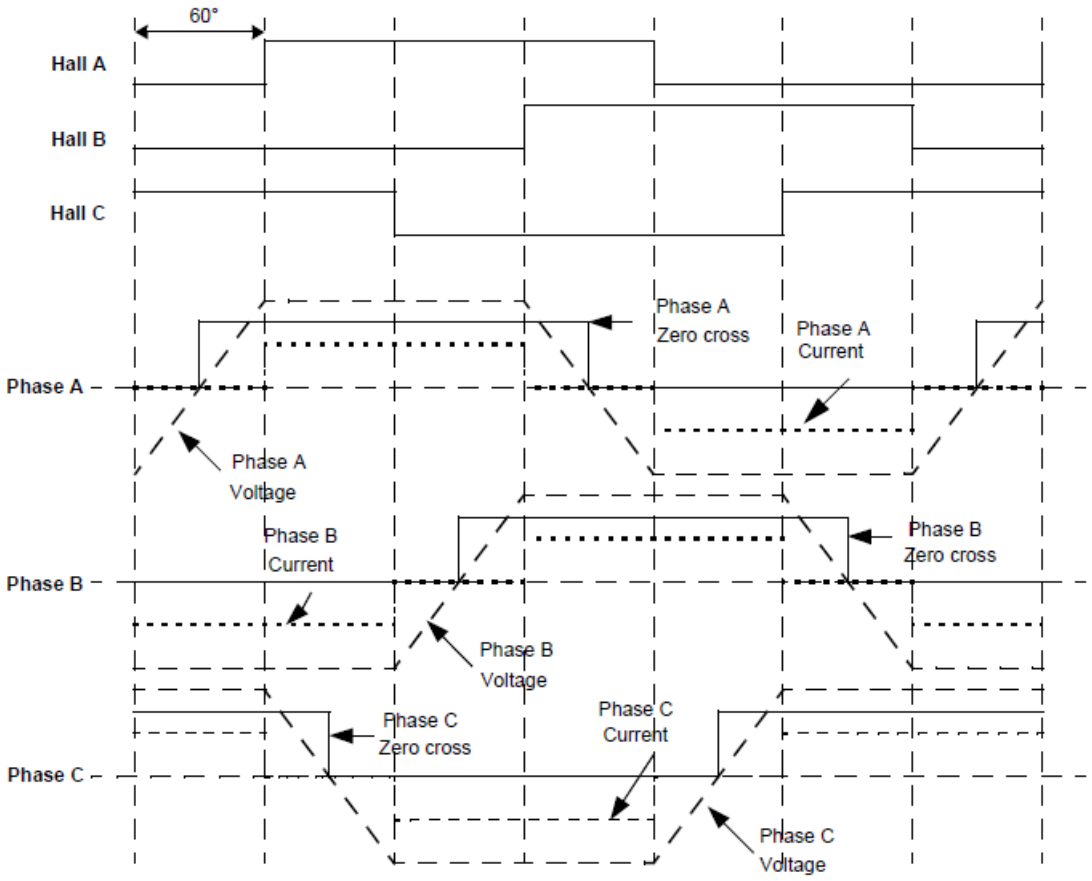
This summer, we worked on a motor control drive with which we could adjust the speed of a DC motor (could be used to run an air conditioner, a scooter wheel, and/or other flying gadgets such as a quadcopter). By using the word “Sensor-less”, we are referring to the fact that no sensors are to be used for detecting its rotor position. Instead we use the Back ElectroMotive Force (Or the BEMF) created by the motor itself to determine the rotor position. This will result in lower cost in mass production, as well as sufficient results, Having generated 3 BEMF signals (each 120° out of phase).

If we were to use sensors, or hall effect sensors, we would have to measure the density of a magnetic field around the device the entire time. These sensors have a present threshold and when the magnetic flux density exceeds this limit, the device is able to detect the magnetic field by generating an output called the hall voltage.

Back EMF signals are NOT synchronized with the hall effect sensor signals (shift of 30°).

For this part of the project we got help from this website presented below:

https://simple-circuit.com/arduino-sensorless-bldc-motor-controller-esc/



In every sequence, 2 windings are energized. One connected to positive and the other connected to negative while the third winding is to be left open (floating). The floating winding is applied for the zero-

crossing detection. The combination of all 3 zero crossing points are used to generate the energizing sequence; Therefore we have 6 events demonstrated below:

Phase A Zero-Crossing : High to Low & Low to High.

Phase B Zero-Crossing : High to Low & Low to High.

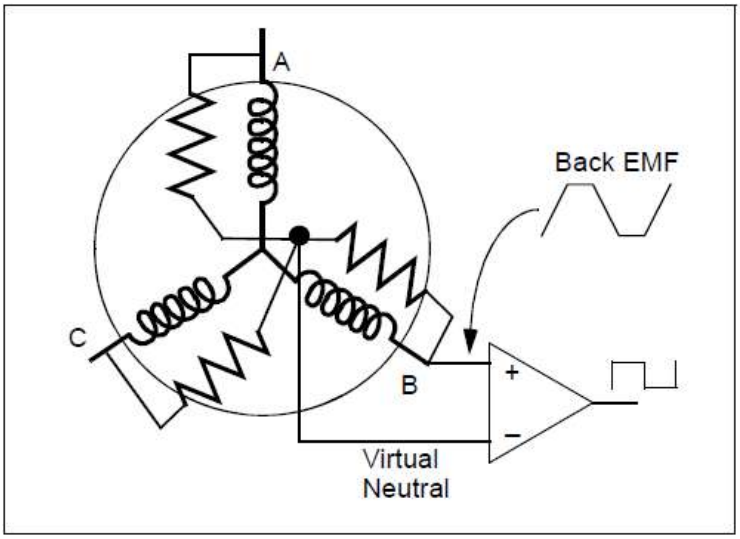
Phase C Zero-Crossing : High to Low & Low to High.

In this part, we use comparators to detect the events. The comparator has 3 main terminals:

2 inputs (Positive and Negative) and one output;

Output is Logic-High if : Positive input > Negative input

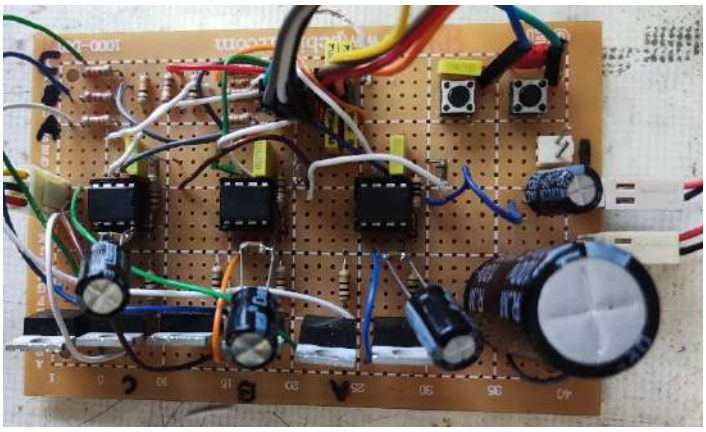
And Logic-Low if : Positive input < Negative input

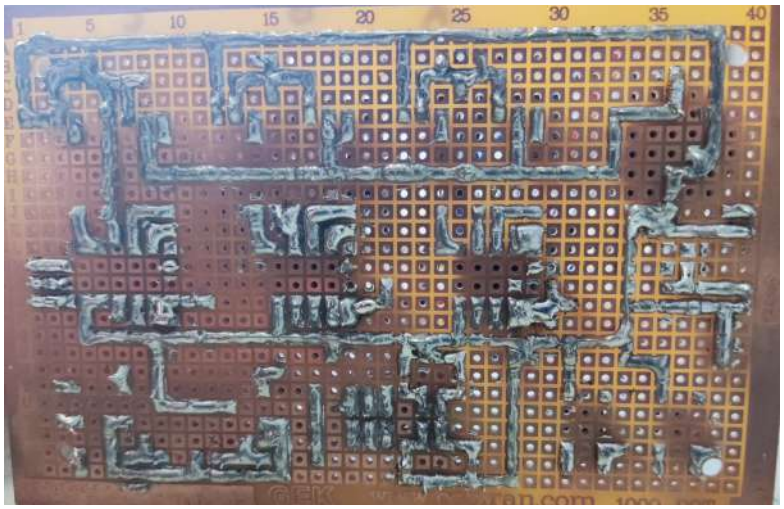


Now based on this website and this schematic below, we assembled the schematic using the Proteus application, applying the Arduino ".hex" code inside Proteus to see if it works.

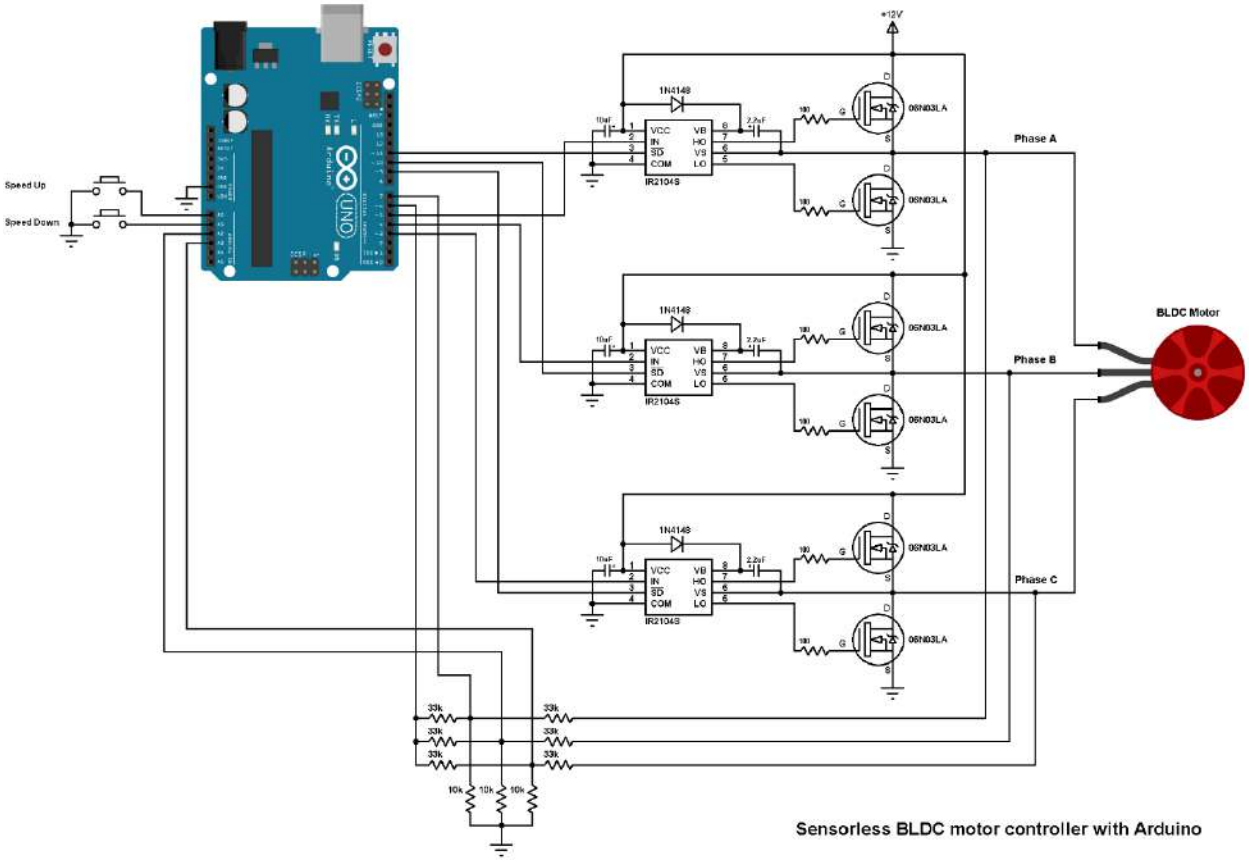
Folder name: "simulation"

This is how the BLDC-DRIVER ended up to be:



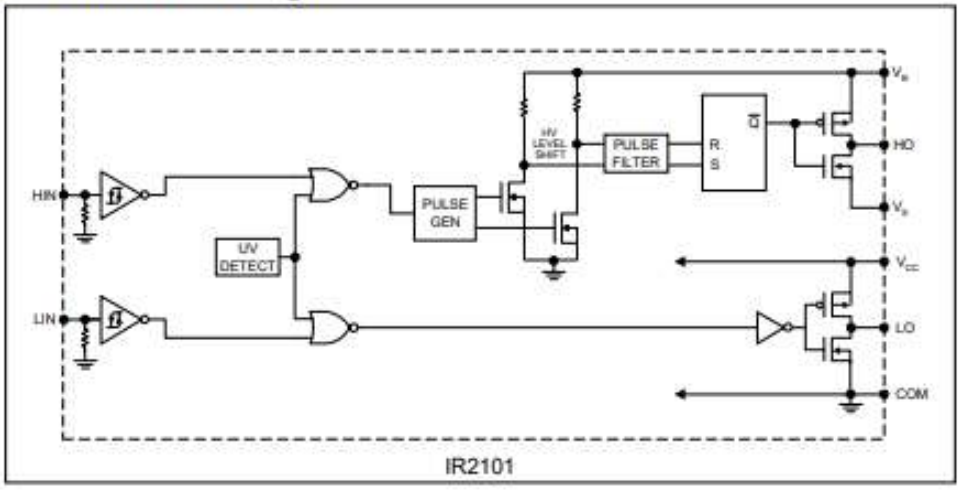


The scooter motor we are using:

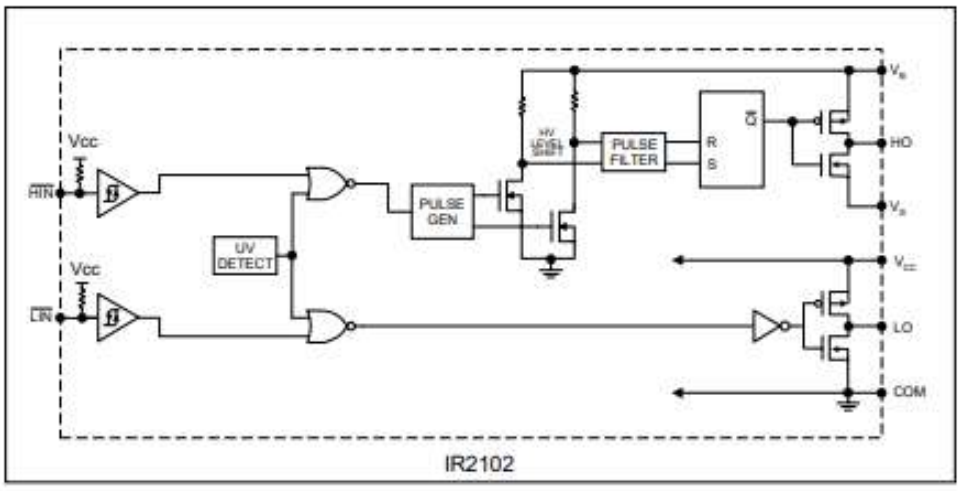


We used 3 IR2104 ICs in this Part of the project. But IR2104 has a down side compared to 2103; Based on the inner schematic and structure of each IC we can see that the 2104 has the High side input connected to DEAD TIME & SHOOT-THROUGH PREVENTION where as in 2103 both High side and Low side inputs are connected to DEAD TIME & SHOOT-THROUGH PREVENTION independently. In IR2101 and IR2102 however, there isn't any DEAD TIME & SHOOT-THROUGH PREVENTION connected to the input:

Functional Block Diagram

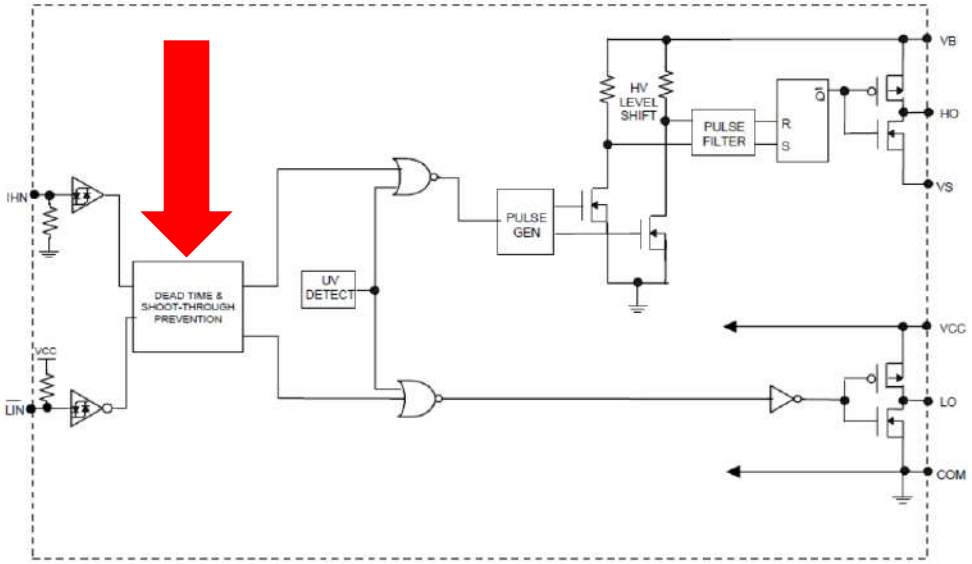


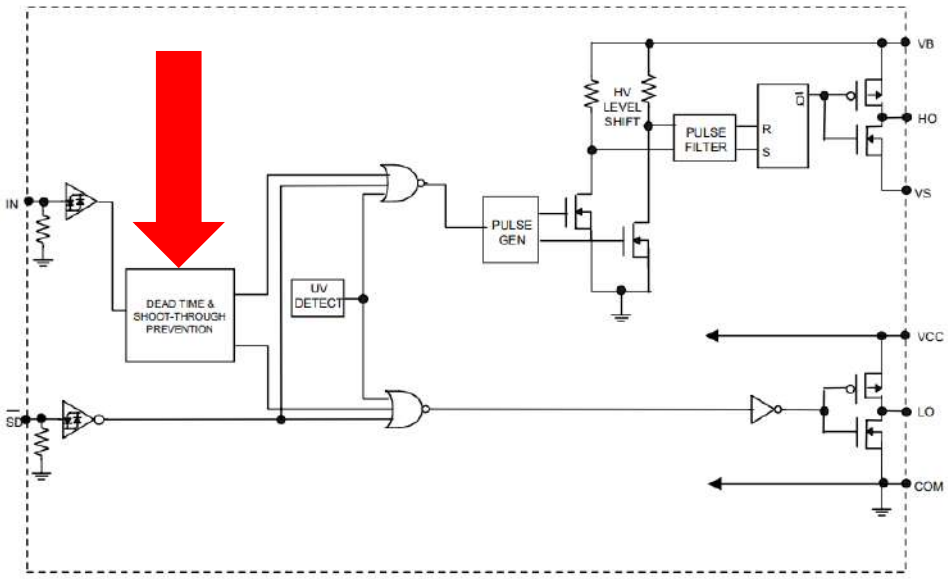
IR2101



IR2102

Functional Block Diagram





(That's why further in the project we will use IR2103.)

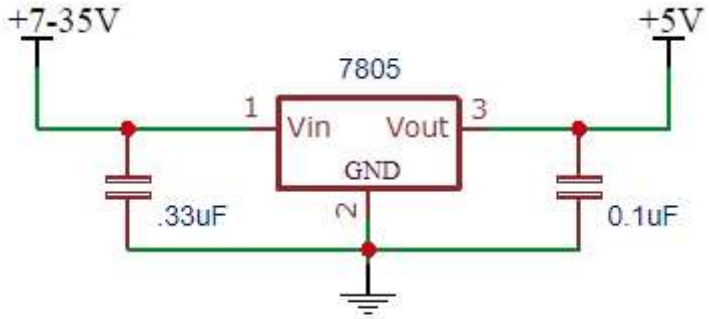
This section's Arduino code can be found in the Arduino code in the folder:

"Arduino_Scooter_Motor_Code".

Every general detail of the code is explained in the comment section.

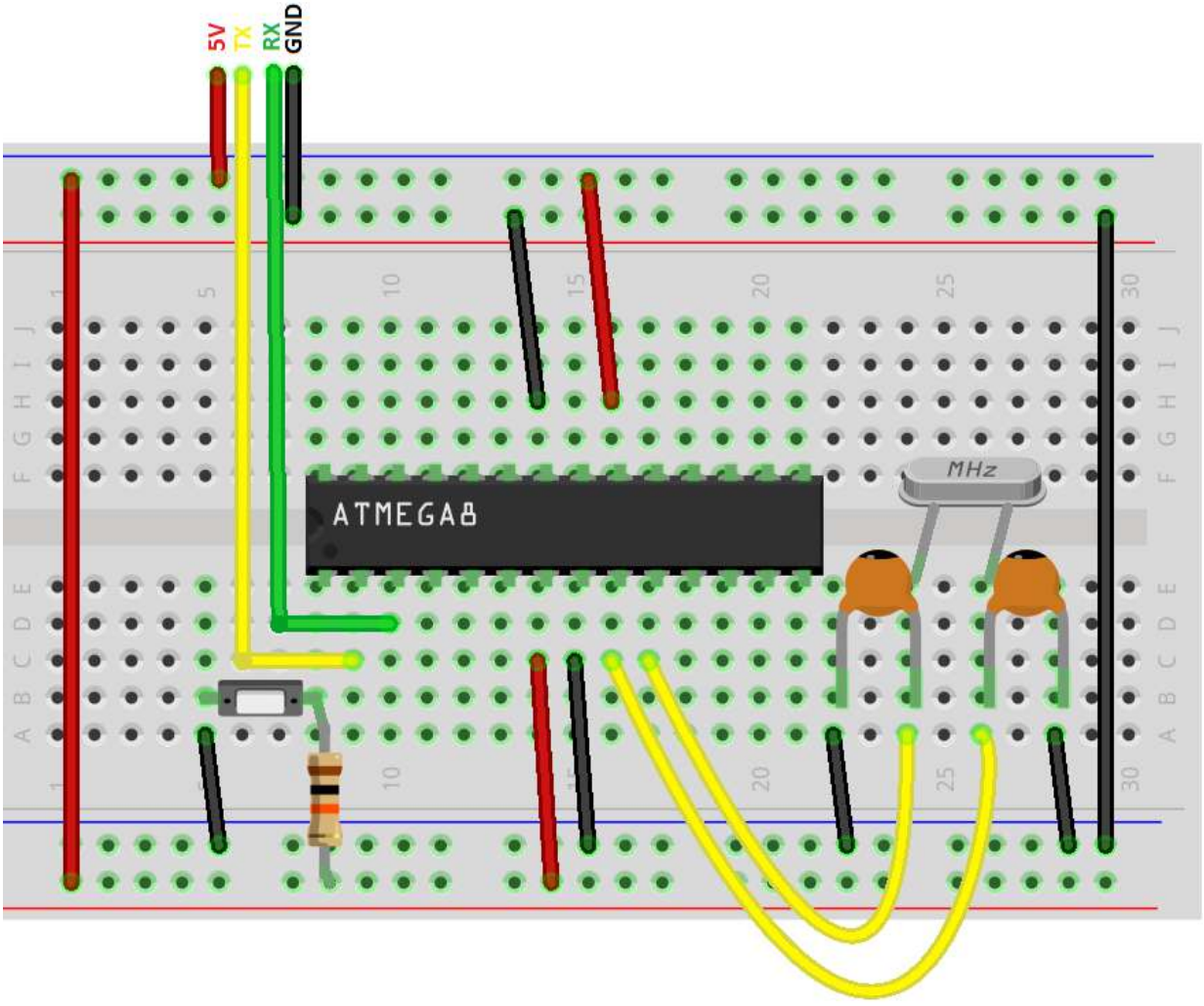
Next off, we transferred this code to the Atmel studio app. Programming the ATmega8a using the

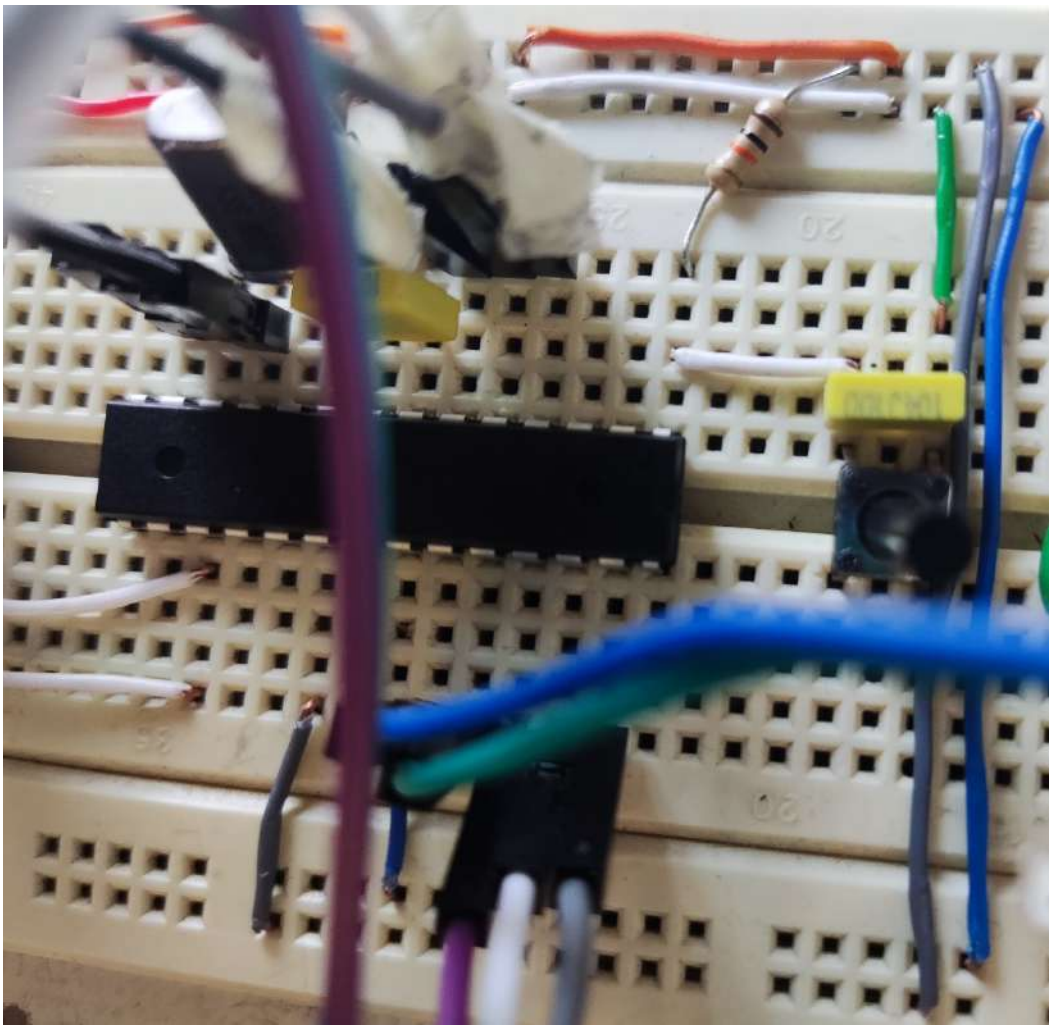
For the V_{cc} = 5V of the bread board, we used a 7805 regulator:



7805 5V Regulator With Bypass Caps

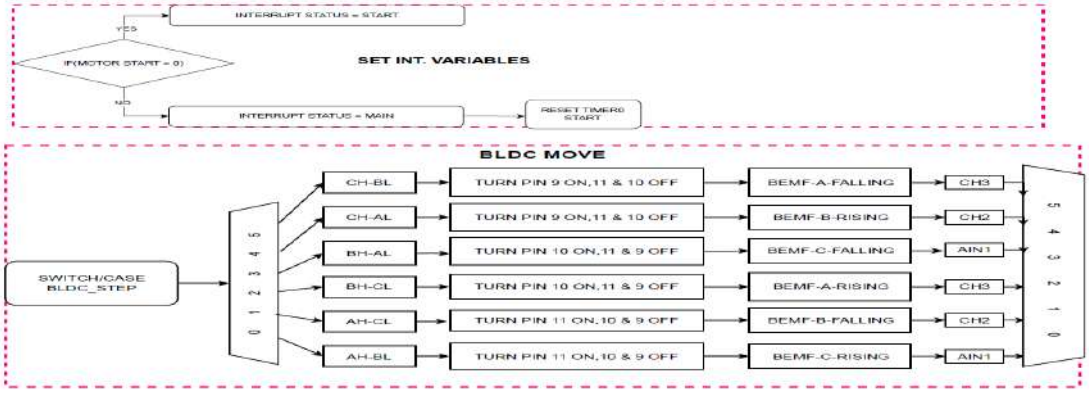
Connecting the other necessary wires based on the schematic below, we concluded the breadboard connections.

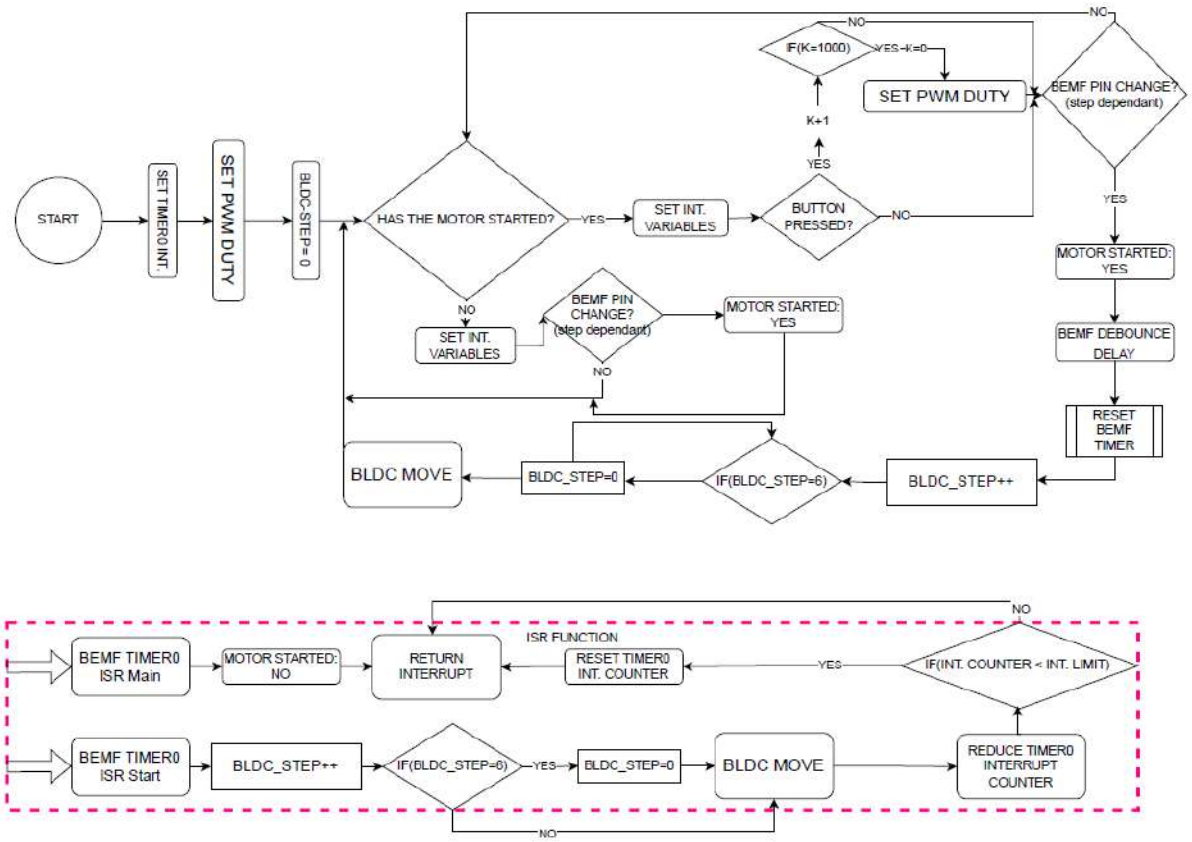




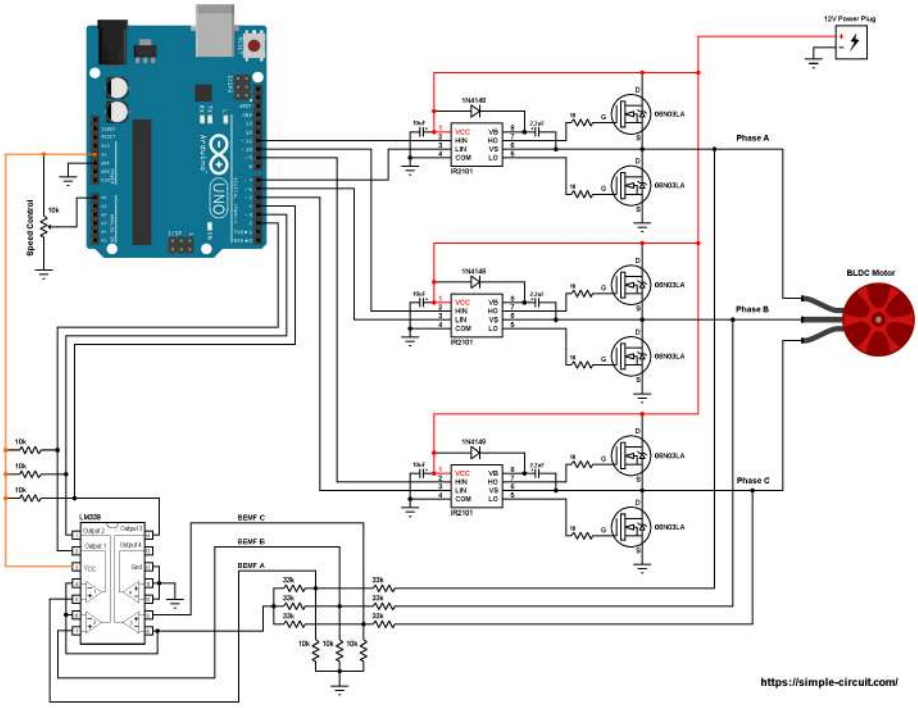
Then we had an Arduino based code for ATmega8a in this file below: “final3”.

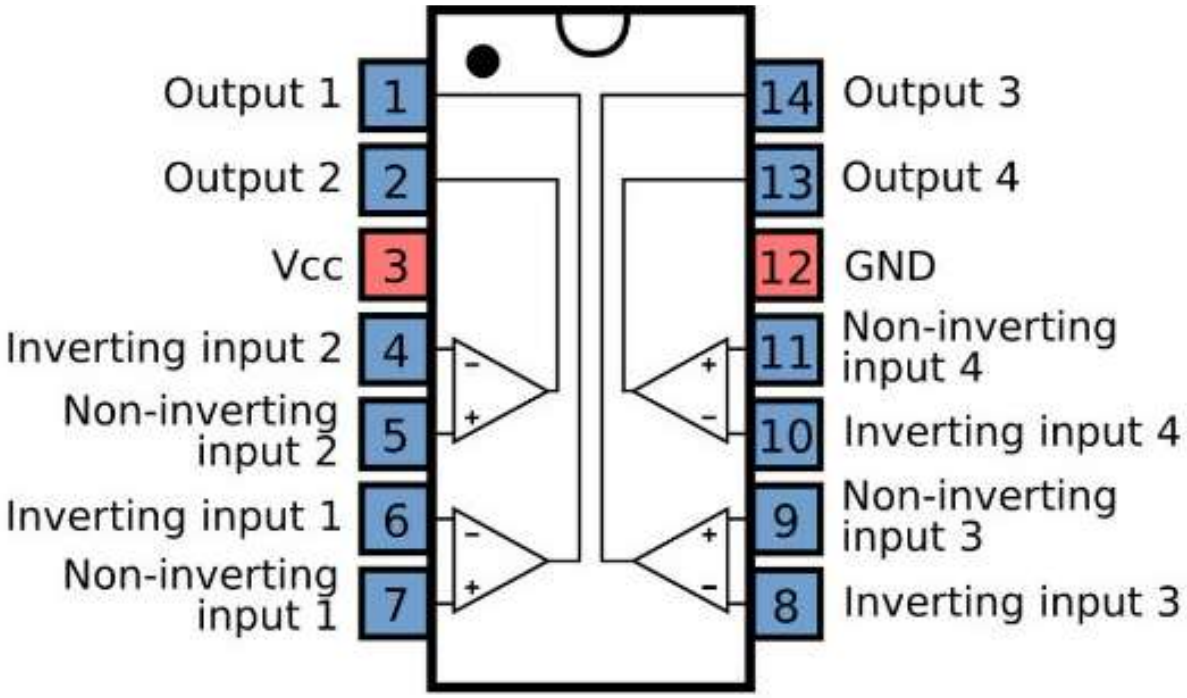
Not being able to run the code without the PWM pins we had to start a new code from scratch based on this flowchart below. Note that a few changes were forced on us to make, and the final code differs from this flowchart: (File name: “FLOWCHART.pdf”)



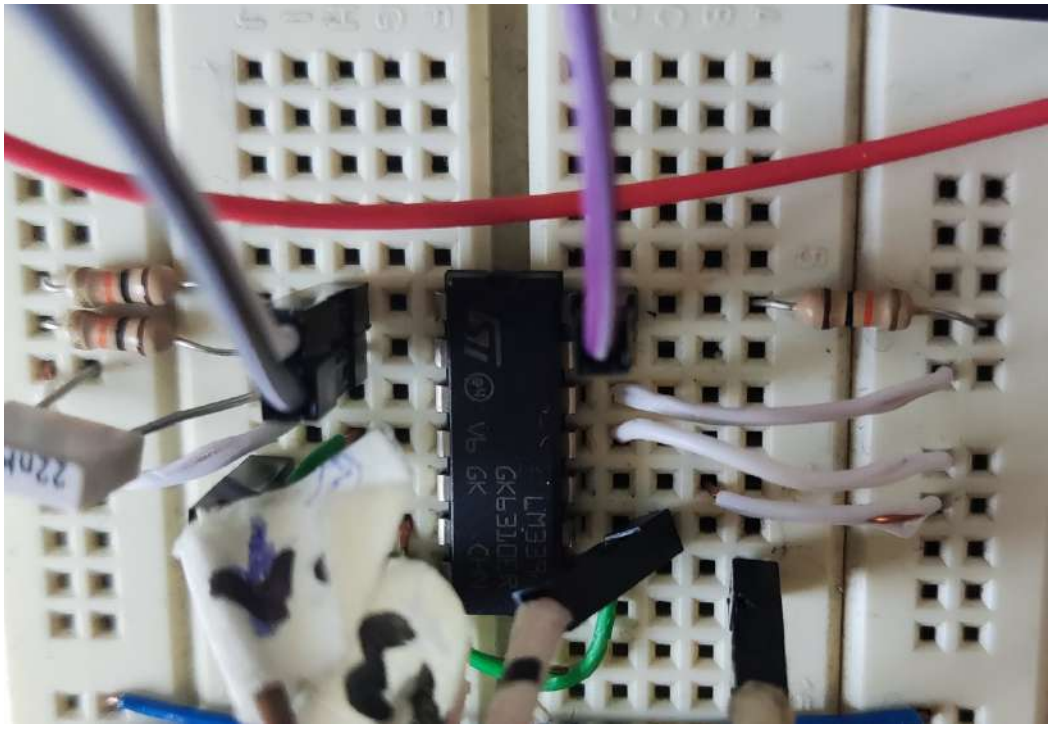


In contrast to the Arduino code, where the ANALOG_COMP_vect ISR was used, this time we used an LM339 to notice the BEMF and determine the next case.

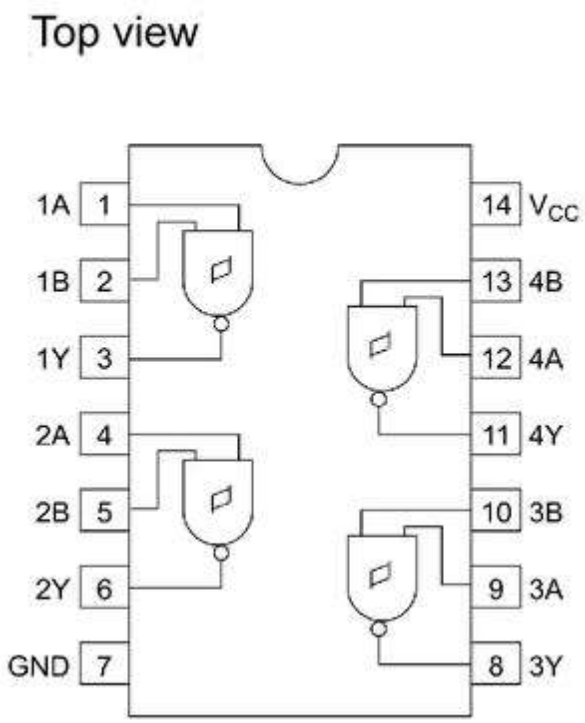


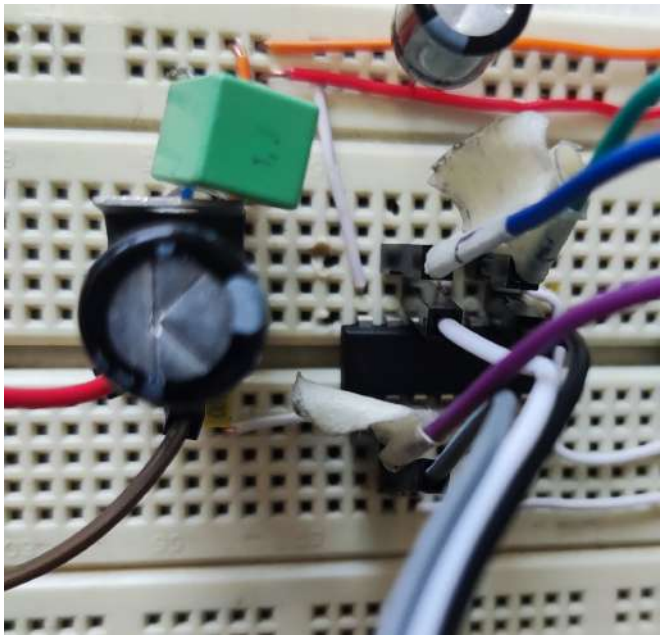


LM339 DIP package



In addition, for simplicity we only set one PWM port using the SN74HC132N NAND IC:





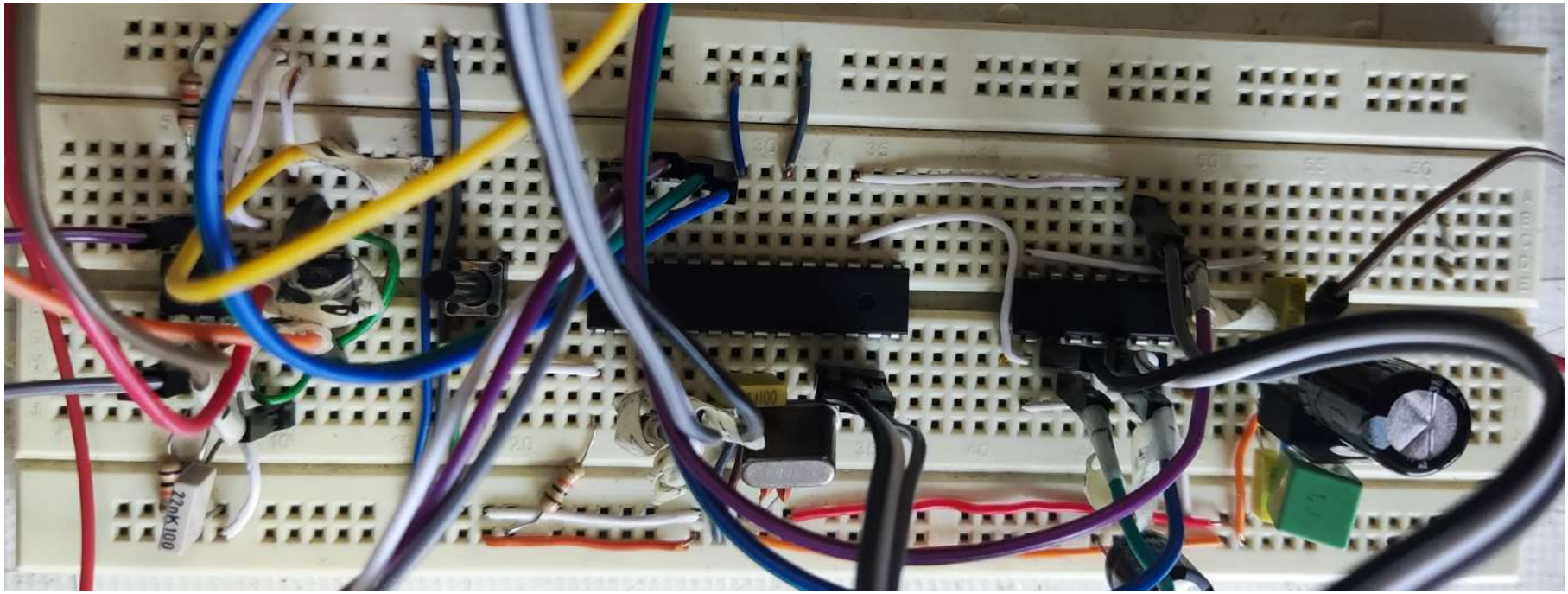
We then wrote a code using the BEMF of each case to predict the next stage of the motor motion. This is the table which helped us write this part:

ورودی ها													خروجی ها												توضیحات
BEMF			سنسورهای اثر هال						سیگنالهای کنترلی				سوییچ های بالا			سوییچ های پایین			های نشان گر LED						
			60°			120°																			
A	B	C	HALL_A	HALL_B	HALL_C	HALL A	HALL_B	HALL_C	F/R	EN	BRAKE	OCP	HIN_A	HIN_B	HIN_C	LIN_A	LIN_B	LIN_C	LED1	LED2	LED3	LED4	LED5	LED6	LED1: EN LED2: FAULT LED3: OCP LED4: SPEED LOOP LED5: SPEED ADJ BLINK LED6: RS232/MANUAL
		0	1	0	0	1	0	0	0	0	1	X	1	0	0	0	0	1	0	1	X	X	X	X	Normal Forward Application F/R=0
	1		1	1	0	1	1	0	0	0	1	X	0	1	0	0	0	1	0	1	X	X	X	X	
0			1	1	1	0	1	0	0	0	1	X	0	1	0	1	0	0	0	1	X	X	X	X	
		1	0	1	1	0	1	1	0	0	1	X	0	0	1	1	0	0	0	1	X	X	X	X	
	0		0	0	1	0	0	1	0	0	1	X	0	0	1	0	1	0	0	1	X	X	X	X	
1			0	0	0	1	0	1	0	0	1	X	1	0	0	0	1	0	0	1	X	X	X	X	
0			1	0	0	1	0	0	1	0	1	X	0	0	1	1	0	0	0	1	X	X	X	X	Normal Reverse Application F/R=1
		1	1	1	0	1	1	0	1	0	1	X	0	0	1	0	1	0	0	1	X	X	X	X	
	0		1	1	1	0	1	0	1	0	1	X	1	0	0	0	1	0	0	1	X	X	X	X	
1			0	1	1	0	1	1	1	0	1	X	1	0	0	0	0	1	0	1	X	X	X	X	
		0	0	0	1	0	0	1	1	0	1	X	0	1	0	0	0	1	0	1	X	X	X	X	
	1		0	0	0	1	0	1	1	0	1	X	0	1	0	1	0	0	0	1	X	X	X	X	
			1	0	1	1	1	1	X	0	1	X	0	0	0	0	0	0	0	0	X	X	X	X	Invalid sensor input data
			0	1	0	0	0	0	X	0	1	X	0	0	0	0	0	0	0	0	X	X	X	X	
			1	0	1	1	1	1	X	X	0	X	0	0	0	1	1	1	X	0	X	X	X	X	Invalid sensor input data with brake
			0	1	0	0	0	0	X	X	0	X	0	0	0	1	1	1	X	0	X	X	X	X	
			V	V	V	V	V	V	X	X	0	X	0	0	0	1	1	1	X	1	X	X	X	X	Valid inputs with brake=0
			V	V	V	V	V	V	X	1	1	X	0	0	0	0	0	0	1	1	X	X	X	X	Valid inputs with EN=1

For the Hall Sensors, we extracted the table from the MC33035 Datasheet; However we derived the BEMF part ourselves.

File name: “ATmega8a_Final”

We explained every part of the code via comments throughout the entire code.


220K 100