



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

MICROPROCESSOR

Report

By Matthew Taylor

Contents

Task One.....	3
Introduction	3
Code	3
Conclusion.....	5
Task Two	6
Introduction	6
Code	6
Conclusion.....	9
Annex	10
Flow chart for task one	10
Flow chart for task two	11
Code for Task one	12
Data from the serial monitor for task one	13
Code for Task two	14

Task One

Introduction

I designed a flow chart for the PWM motor as it gives me a format to follow when writing the code. When I designed The flow diagram found in the annex, I planned on creating a simple loop and using the interrupt service routine for one of the user inputs. During the writing of the code I changed my approach so the code doesn't share much similarity with the flow diagram but, the flow diagram was still a beneficial part of the process to getting to the code I have now.

Code

All the code for task one is within the main loop. The loop starts, as shown below, by debouncing one and reading the three user inputs and saving their state in variables. Switch A is the most complicated switch to read because I didn't want the switch to be registered if the button hasn't been released otherwise the program would read a new button press on every loop despite the user not releasing the button. The way it's written the program remembers the last state so it won't register a press of the button unless the previous state was low. This idea could be improved and I will talk more about it at the end.

```
30. void loop() {
31.     // Debounce and read buttons and save config.
32.
33.     Serial.println("-----");
34.
35.     AState = digitalRead(SwitchA);
36.     if (AState == HIGH && previousA == 0) {
37.         Aconfig = HIGH;
38.         Serial.println("Aconfig HIGH");
39.     };
40.     previousA = AState;
41.
42.     Bconfig = digitalRead(SwitchB);
43.     if (Bconfig == HIGH) { Serial.println("Bconfig HIGH"); };
44.
45.     Cconfig = digitalRead(LatchC);
46.     if (Cconfig == HIGH) { Serial.println("Cconfig HIGH"); };
```

Part two of the code deviates from the flow chart, the original idea was to keep this part very simple and have an interrupt service routine control the enable pin of the motor. I decided to not go with the interrupt method for a few reasons. The main advantage of an interrupt is that they are fast acting but as my loop is short and simple, the latch which the interrupt would be monitoring is checked by the loop at a very high frequency. The disadvantages of the interrupt are that it is only accessible on a few certain pins on the Arduino Uno, this doesn't include pin 3 and that not all functions work inside the interrupt, like delay. I decided that I would change plans, as it would be simpler to incorporate the latch into the same part of the code with buttons A and B. The way I chose to do the second part of the main loop is described in the next paragraph.

Part two of the main loop first tests the state of the latch because if this is open then the break pin needs to be pulled high and the speed set to zero. If the latch is set to closed by the user, then the program jumps to the ELSE line of the IF statement. Within the ELSE there are two configurations possible, there is an IF statement for each. These IF statements adjust the variable that controls the speed. The last lines of this snippet reset the config, ready for the next loop.

```
48.  if (Cconfig == HIGH) {
49.
50.      digitalWrite(breakPin, HIGH);
51.      varDelay = 1500;
52.      Serial.println("Break set");
53.
54.      // reset
55.      Aconfig = LOW;
56.      Bconfig = LOW;
57.      Cconfig = LOW;
58.
59.  } else {
60.
61.      digitalWrite(breakPin, LOW);
62.      Serial.println("Break released");
63.
64.      if (Aconfig == HIGH && Bconfig == LOW) {
65.          varDelay = varDelay + 10;
66.          Serial.println("Speed increase");
67.      };
68.
69.      if (Aconfig == HIGH && Bconfig == HIGH) {
70.          varDelay = varDelay - 10;
71.          Serial.println("Speed decrease");
72.      };
73.      // reset
74.      Aconfig = LOW;
75.      Bconfig = LOW;
76.      Cconfig = LOW;
77.
78.  };
```

Part three of the main loop, shown below, is responsible for creating a square wave. It is the shape of the square wave that indicates the speed of the motor and is determined by the variable called 'varDelay'. The speed is also printed to the monitor.

```
80.  // generate a square wave to the spec of varDelay.
81.  Serial.println(varDelay);
82.  digitalWrite(motorPin, HIGH);
83.  delayMicroseconds(varDelay);
84.  digitalWrite(motorPin, LOW);
85.  Serial.println("-----");
86.  }
```

Below is what is printed to the serial monitor during operation. The dashed lines start and end a loop. When Latch C is high the break is set and the speed is set to 0 which requires a pulse of 1500 milliseconds. When button A is pressed then the speed is increased by 10ms, up to 2000ms where the motor will be at its maximum speed. The program will allow the speed to go beyond 2000ms but as it won't affect the motor, the program could be improved by capping the maximum and minimum speed. The final loop shows a decrease in speed, this works similarly to increasing the speed, with a minimum of 1000ms.

```
1. Setup Finished
2. -----
3. Break released
4. 1500
5. -----
6. -----
7. Cconfig HIGH
8. Break set
9. 1500
10. -----
11. -----
12. Aconfig HIGH
13. Break released
14. Speed increase
15. 1510
16. -----
17. -----
18. Aconfig HIGH
19. Bconfig HIGH
20. Break released
21. Speed decrease
22. 1490
23. -----
```

Conclusion

After testing the program, I had to rework it to make it work as expected. The biggest issue was due to the first design debouncing all user inputs, this has two undesired consequences. One, that the breaks would be disabled after one run of the loop, and two, button A and B would have to be pressed at the same time for them to be registered as a pair, as the loop is running so quickly this was not user friendly.

Due to the small increment of change on every button press, the user is required to press the button 50 times to get the speed from zero to maximum. To make the program more convenient the way button A is handled in part two could be changed so that when the user holds the button then the speed will increment at a faster pace.

Task Two

Introduction

Task two set out the large project of producing a program to manage a carpark. With a program of this size and complexity it was beneficial to create a flow diagram before working on the code. After having written a flow chart I went on to writing the code for the program. Writing the program with the flowchart to the side is very beneficial as the code can often be split up according to process and decision boxes of the flowchart.

Code

The following lines are the first four lines of the program. The barrier to the car park is a stepper motor so I have included the stepper motor library. Lines three and four are part of setting up the stepper motor, they pass the parameters of the motor to the library so that when a function within the library is called upon it can perform as expected.

```
1. #include <Stepper.h>
2.
3. const int stepsPerRevolution = 200;
4. Stepper myStepper(stepsPerRevolution, 8, 11, 12, 13);
```

The following snippet of code is where the three sensors are debounced and their associated function is called. Sensor one is located at the entrance to the car park and is responsible for opening the barrier.

Sensor two is located right after the barrier and is responsible for closing the barrier and counting the car in. It was designed this way so that a car can't reverse through the entrance once the car has been counted.

Sensor three is located at the exit of the car park. It is responsible for counting the cars that leave the car park. S3 is different from the others as it doesn't call a function that is outside of the loop, with only 2 commands it would be a waste of memory. Within the IF statement of S3 there are some commands for printing information to the monitor, printing lines like these were invaluable to debugging the program. Here they simply print that the program has reached this location and the current count of cars in the car park.

```

61.    // Debounce S1:
62.
63.    readingS1 = digitalRead(S1);
64.    if (readingS1 == HIGH && previousS1 == 0){
65.        newCar();
66.    };
67.    previousS1 = readingS1;
68.
69.    // Debounce S2:
70.
71.    readingS2 = digitalRead(S2);
72.    if (readingS2 == HIGH && previousS2 == 0){
73.        closeBarrier();
74.    };
75.    previousS2 = readingS2;
76.
77.    // Debounce S3:
78.
79.
80.    readingS3 = digitalRead(S3);
81.    if (readingS3 == HIGH && previousS3 == 0){
82.
83.        Serial.println("running carLeaving");
84.        carCount = carCount - 1;
85.        Serial.println(carCount);
86.        digitalWrite(Full, LOW);
87.        Serial.println("returning to loop");
88.        Serial.println("-----");
89.
90.    };
91.    previousS3 = readingS3;

```

To provide feedback for as many possibilities, the program creates a visual indication when a car pulls up to the entrance when the car park is already full. This will be done by blinking the car park full sign for 5 seconds. The code below is situated in the main loop but it will only run if a variable is changed during the new car function. Once a car has triggered S1 whilst the car park is full the variable will change and as the program loops through the main loop, the program will access all the code within the IF statement.

Within the IF statement, the full sign is toggled on or off if enough time has elapsed since the last time the state changed. Once the program has run the sequence 11 times, the program will reset the variables, leaving the sequence can be triggered again after the car has left the entrance.

```

98.  if ( fullBlinkTriggered == true ) {
99.      currentMillis = millis();
100.     if ( currentMillis - previousMillis >= interval &&
        fullStateCount <= 10 ) {
101.         previousMillis = currentMillis;
102.
103.         if (fullState == LOW) {
104.             fullState = HIGH;
105.             lastFullState = fullState;
106.         } else {
107.             fullState = LOW;
108.             lastFullState = fullState;
109.         }
110.         digitalWrite(Full, fullState);
111.         Serial.println("Blinking Full sign");
112.         if ( fullStateCount >= 11 ) {
113.             int fullStateCount = 0;
114.             fullBlinkTriggered = false;
115.         };
116.     };
117. };

```

The code below starts with the new car function. This function is called when a car triggers sensor one. Most of the function are prints to the monitor but on line 129 there's an IF statement that tests if there's space for this new car. If there's space then it calls the next function and the car will move forward. If the car park is full then the global variable called '*fullBlinkTriggered*' is set to be true, this variable is checked during the main loop and will cause the 'Car park full sign' to blink.

```

125. void newCar() {
126.     Serial.println("running new car");
127.     Serial.println("carCount at start:");
128.     Serial.println(carCount);
129.     if (carCount < 5) {
130.         Serial.println("about to run openBarrier");
131.         openBarrier();
132.     } else {
133.         Serial.println("fullBlinkTriggered = true");
134.         int fullBlinkTriggered = true;
135.     };
136. }

```

The open barrier function is called by the previous new car function. Open barrier simply instructs the stepper motor via the stepper library to rotate 50 steps which is equivalent to 90° and to change the traffic lights. A design floor of the program around the entry of a car is that it expects a car to enter S1 then pass S2 before a new car enters the car park. The consequence of a car trailing too closely the car in front and triggering S1 before the first car has passed S2 is a malfunctioning barrier.

The close barrier function does a similar thing in the opposite order. This function also counts the car into the car park. If that was the last space, then the program illuminates the car park full sign otherwise the program returns to the main loop to read the sensors again. A design floor with these two functions is that they don't test what position the barrier is in before sending instructions to the stepper motor. This would be problematic if a car tries to leave via the entrance and triggers sensor two, as this will cause the motor to drive the barrier into the ground. In the conclusion I discuss how I would improve my program.

```

131. void openBarrier() {
132.   Serial.println("running openBarrier");
133.   myStepper.step(openSteps);
134.   digitalWrite(Red, LOW);
135.   digitalWrite(Green, HIGH);
136.   Serial.println("Returning to loop");
137.   Serial.println("-----");
138. }
139.
140. void closeBarrier() {
141.   Serial.println("running closeBarrier");
142.   Serial.println("current carCount:");
143.   Serial.println(carCount);
144.
145.   carCount = carCount + 1;
146.   Serial.println("new carCount");
147.   Serial.println(carCount);
148.   if ( carCount >= 20 ) {
149.     Serial.println("Full sign to high");
150.     digitalWrite( Full, HIGH );
151.   };
152.   digitalWrite(Red, HIGH);
153.   digitalWrite(Green, LOW);
154.   myStepper.step(closeSteps);
155.   Serial.println("returning to loop");
156.   Serial.println("-----");

```

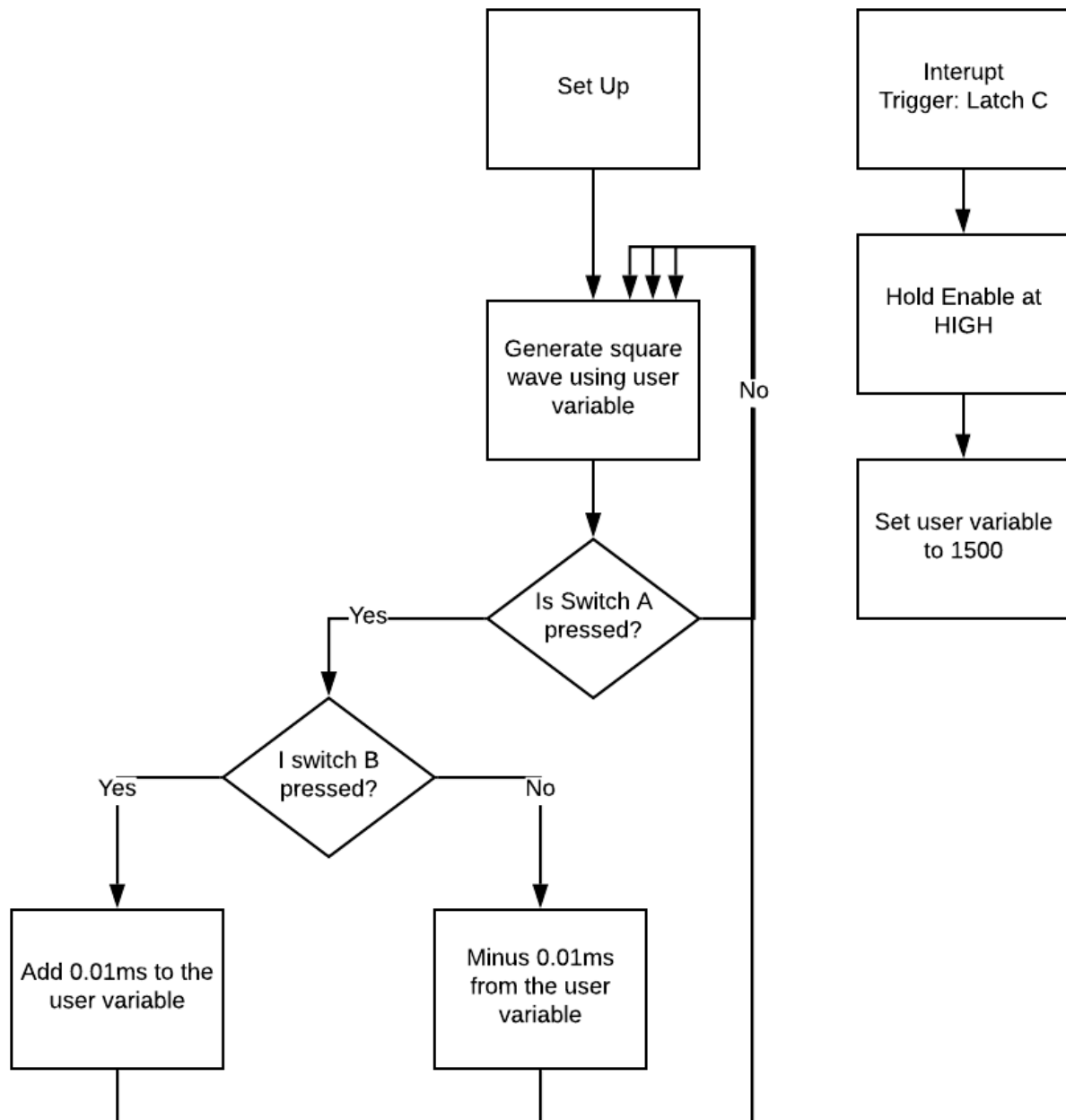
Conclusion

The program that I wrote followed the flow chart I designed at the start of the process quite closely. One part of the flow chart that I dropped from the program is verifying that the car left Sensor one. I decided that if a car was to pass sensor two then it must have left Sensor one.

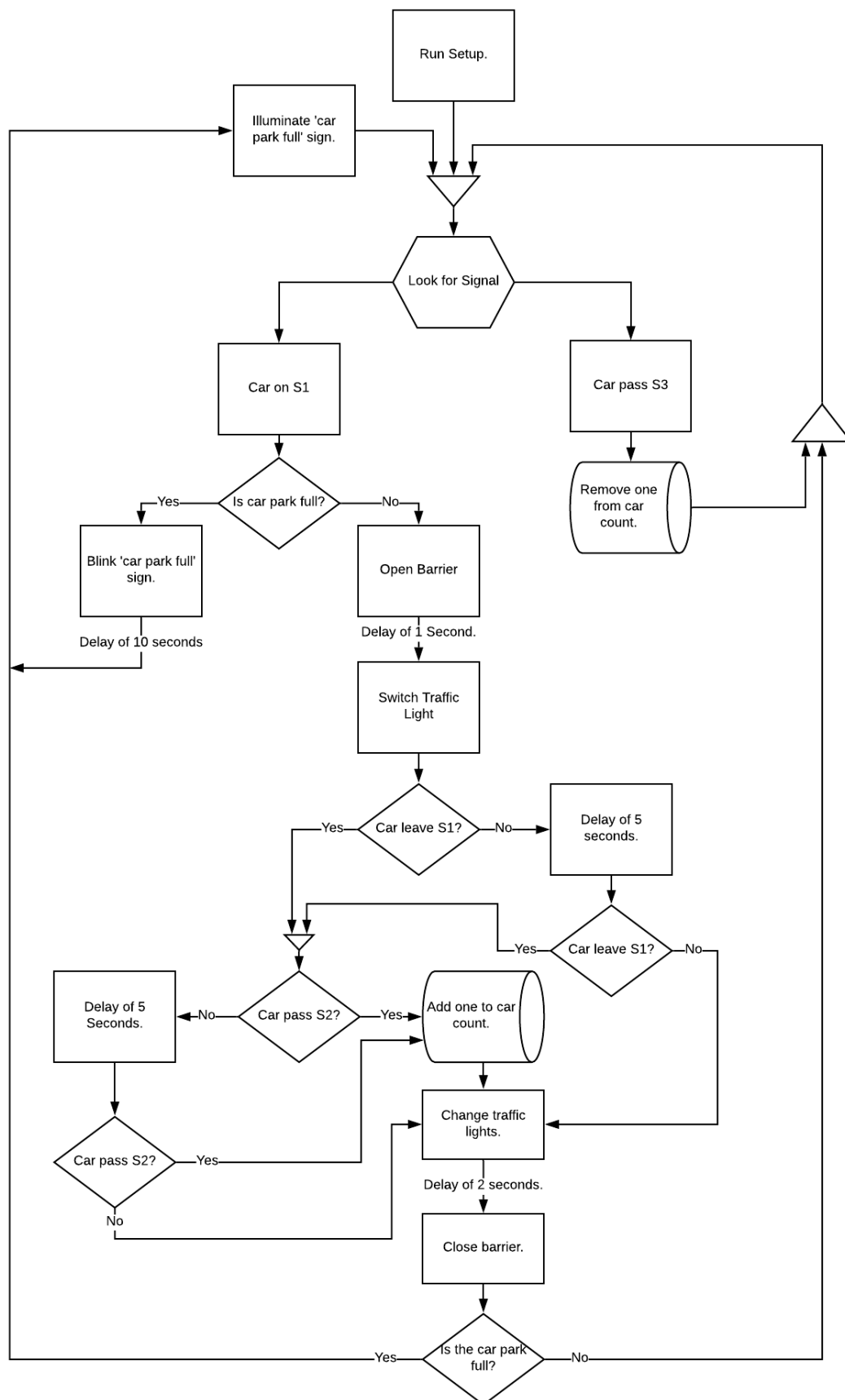
The biggest issue with the program as highlighted earlier is that the barrier can be given instructions to rotate outside of the normal operating range, this will break the barrier. The simplest way of preventing this would be to store the position of the motor in a variable and test this variable before moving the motor.

Annex

Flow chart for task one



Flow chart for task two



Code for Task one

```
87. int SwitchA = 2;           // Pin for switch A.
88. int SwitchB = 3;           // Pin for switch B.
89. int LatchC = 4;            // Pin for switch C.
90. int AState;
91. int BState;
92. int CState;
93. int Aconfig;
94. int Bconfig;
95. int Cconfig;
96. int previousA = 0;
97. int previousB = 0;
98. int previousC = 0;
99. int motorPin = 13;          // Pin with pulse for motor.
100. int breakPin = 12;          // Pin for disc break.
101. int varDelay = 1500;        // the current pulse width
102.
103. void setup() {
104.
105.     pinMode(SwitchA, INPUT);
106.     pinMode(SwitchB, INPUT);
107.     pinMode(LatchC, INPUT);
108.     pinMode(motorPin, OUTPUT);
109.     pinMode(breakPin, OUTPUT);
110.     digitalWrite(motorPin, LOW);
111.     digitalWrite(breakPin, LOW);
112.     Serial.begin(9600);
113.     Serial.println("Setup Finished");
114. }
115.
116. void loop() {
117.     // Debounce and read buttons and save config.
118.
119.     Serial.println("-----");
120.
121.     AState = digitalRead(SwitchA);
122.     if (AState == HIGH && previousA == 0) {
123.         Aconfig = HIGH;
124.         Serial.println("Aconfig HIGH");
125.     };
126.     previousA = AState;
127.
128.     Bconfig = digitalRead(SwitchB);
129.     if (Bconfig == HIGH) { Serial.println("Bconfig HIGH"); };
130.
131.     Cconfig = digitalRead(LatchC);
132.     if (Cconfig == HIGH) { Serial.println("Cconfig HIGH"); };
133.
134.     if (Cconfig == HIGH) {
135.
136.         digitalWrite(breakPin, HIGH);
137.         varDelay = 1500;
138.         Serial.println("Break set");
139.
140.         // reset
141.         Aconfig = LOW;
142.         Bconfig = LOW;
143.         Cconfig = LOW;
144.     }
```

```

145. } else {
146.
147.     digitalWrite(breakPin, LOW);
148.     Serial.println("Break released");
149.
150.     if (Aconfig == HIGH && Bconfig == LOW) {
151.         varDelay = varDelay + 10;
152.         Serial.println("Speed increase");
153.     };
154.
155.     if (Aconfig == HIGH && Bconfig == HIGH) {
156.         varDelay = varDelay - 10;
157.         Serial.println("Speed decrease");
158.     };
159.     // reset
160.     Aconfig = LOW;
161.     Bconfig = LOW;
162.     Cconfig = LOW;
163.
164. };
165.
166. // generate a square wave to the spec of varDelay.
167. Serial.println(varDelay);
168. digitalWrite(motorPin, HIGH);
169. delayMicroseconds(varDelay);
170. digitalWrite(motorPin, LOW);
171. Serial.println("-----");
172. }
173.

```

Data from the serial monitor for task one

```

24. Setup Finished
25. -----
26. Break released
27. 1500
28. -----
29. -----
30. Cconfig HIGH
31. Break set
32. 1500
33. -----
34. -----
35. Aconfig HIGH
36. Break released
37. Speed increase
38. 1510
39. -----
40. -----
41. Aconfig HIGH
42. Bconfig HIGH
43. Break released
44. Speed decrease
45. 1490
46. -----

```

Code for Task two

```
1. #include <Stepper.h>
2.
3. const int stepsPerRevolution = 200;
4. Stepper myStepper(stepsPerRevolution, 8, 11, 12, 13);
5.
6.
7. // Constants:
8.
9. const int S1 = 7;
10. const int S2 = 2;
11. const int S3 = 3;
12. const int Red = 4;
13. const int Green = 5;
14. const int Full = 6;
15. const int interval = 500;
16. const int debounceDelay = 10;
17. const int openSteps = 50;
18. const int closeSteps = -50;
19.
20. // Variables;
21.
22. int i = 0;
23. int redState = HIGH;
24. int greenState = LOW;
25. int fullState = LOW;
26. int lastFullState = LOW;
27. int carCount = 0;
28. int fullBlinkTriggered = false; // used to tell the loop to blink
    full sign for 5 seconds.
29. int fullStateCount = 0;          // used to count how many time the
    sign has blinked.
30. int readingS1 = LOW;
31. int readingS2 = LOW;
32. int readingS3 = LOW;
33. int previousS1 = LOW;
34. int previousS2 = LOW;
35. int previousS3 = LOW;
36. unsigned long previousMillis = 0;
37. unsigned long currentMillis = millis();
38.
39.
40. void setup() {
41.   pinMode(S1, INPUT);
42.   pinMode(S2, INPUT);
43.   pinMode(S3, INPUT);
44.   pinMode(Green, OUTPUT);
45.   pinMode(Red, OUTPUT);
46.   pinMode(Full, OUTPUT);
47.   pinMode(9, OUTPUT);
48.   pinMode(10, OUTPUT);
49.   digitalWrite(9, HIGH);
50.   digitalWrite(10, HIGH);
51.   digitalWrite(Green, greenState);
52.   digitalWrite(Full, fullState);
53.   digitalWrite(Red, redState);
54.   myStepper.setSpeed(30);
55.   Serial.begin(9600);
56.   Serial.println("Setup Finished");
```

```

57. }
58.
59. void loop() {
60.
61.     // Debounce S1:
62.
63.     readingS1 = digitalRead(S1);
64.     if (readingS1 == HIGH && previousS1 == 0){
65.         newCar();
66.     };
67.     previousS1 = readingS1;
68.
69.     // Debounce S2:
70.
71.     readingS2 = digitalRead(S2);
72.     if (readingS2 == HIGH && previousS2 == 0){
73.         closeBarrier();
74.     };
75.     previousS2 = readingS2;
76.
77.     // Debounce S3:
78.
79.
80.     readingS3 = digitalRead(S3);
81.     if (readingS3 == HIGH && previousS3 == 0){
82.
83.         Serial.println("running carLeaving");
84.         carCount = carCount - 1;
85.         Serial.println(carCount);
86.         digitalWrite(Full, LOW);
87.         Serial.println("returning to loop");
88.         Serial.println("-----");
89.
90.     };
91.     previousS3 = readingS3;
92.
93.
94.
95.
96.     // Blink the Car Park Full Sign:
97.
98.     if ( fullBlinkTriggered == true ) {
99.         currentMillis = millis();
100.        if ( currentMillis - previousMillis >= interval &&
fullStateCount <= 10 ) {
101.            previousMillis = currentMillis;
102.
103.            if (fullState == LOW) {
104.                fullState = HIGH;
105.                lastFullState = fullState;
106.            } else {
107.                fullState = LOW;
108.                lastFullState = fullState;
109.            }
110.            digitalWrite(Full, fullState);
111.            Serial.println("Blinking Full sign");
112.            if ( fullStateCount >= 11 ) {
113.                int fullStateCount = 0;
114.                fullBlinkTriggered = false;
115.            };
116.        };

```

```

117. };
118.
119. }
120.
121. void newCar() {
122.     Serial.println("running new car");
123.     Serial.println("carCount at start:");
124.     Serial.println(carCount);
125.     if (carCount < 5) {
126.         Serial.println("about to run openBarrier");
127.         openBarrier();
128.     } else {
129.         Serial.println("fullBlinkTriggered = true");
130.         int fullBlinkTriggered = true;
131.     };
132. }
133.
134.
135. void openBarrier() {
136.     Serial.println("running openBarrier");
137.     myStepper.step(openSteps);
138.     digitalWrite(Red, LOW);
139.     digitalWrite(Green, HIGH);
140.     Serial.println("Returning to loop");
141.     Serial.println("-----");
142. }
143.
144. void closeBarrier() {
145.     Serial.println("running closeBarrier");
146.     Serial.println("current carCount:");
147.     Serial.println(carCount);
148.
149.     carCount = carCount + 1;
150.     Serial.println("new carCount");
151.     Serial.println(carCount);
152.     if ( carCount >= 20 ) {
153.         Serial.println("Full sign to high");
154.         digitalWrite( Full, HIGH );
155.     };
156.     digitalWrite(Red, HIGH);
157.     digitalWrite(Green, LOW);
158.     myStepper.step(closeSteps);
159.     Serial.println("returning to loop");
160.     Serial.println("-----");
161. }
162.
163. void carLeaving() {
164.     Serial.println("running carLeaving");
165.     carCount = carCount - 1;
166.     Serial.println(carCount);
167.     digitalWrite(Full, LOW);
168.     Serial.println("returning to loop");
169.     Serial.println("-----");
170. }

```