AIM:

The aim for this assignment was to create a project that projects our learning to a Microcontroller Systems.

UNIT GOALS ASSESSED:

- Demonstrate an understanding of microcontrollers and interfacing electronic hardware as it relates to a Microcontroller system
- Explore and examine the computer programming to effectively control Microcontroller systems
- Demonstrate an understanding of the Programming Design Cycle as it relates to Microcontroller Systems

Our Project

For this assessment, my group and I were working to create a functioning alarm system to be added to a miniature prison replica. The system is based on Arduino and simply consists of three individual components which are an LED light (originally an RGB LED), piezo buzzer and a servo motor. Each member of the group was tasked to research and successfully build a functional component, for us to collaboratively work together to bring the individual aspects together to create the main body of work.

METHOD:

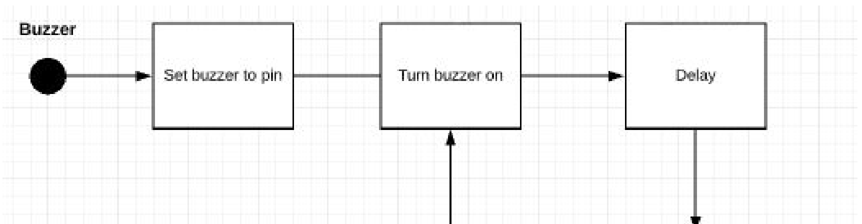
Section 1: Knowledge and Acquisition

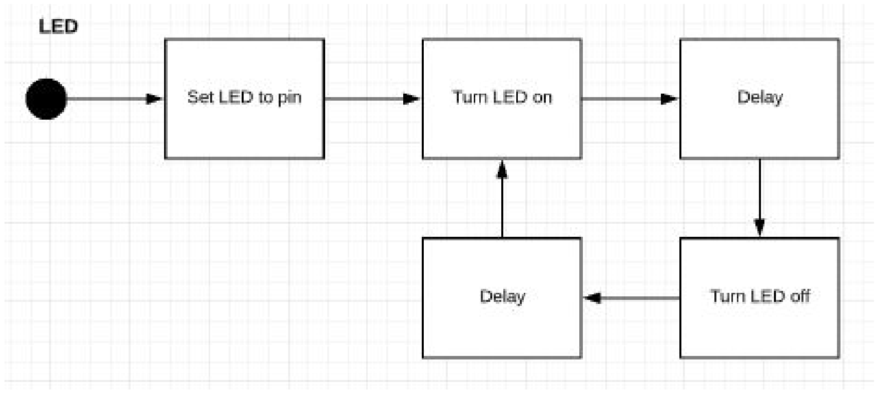
Everything is built into a servo motor: a motor, a feedback circuit and most importantly, a motor driver. Resulting in it having one power line, one ground, and one control pin

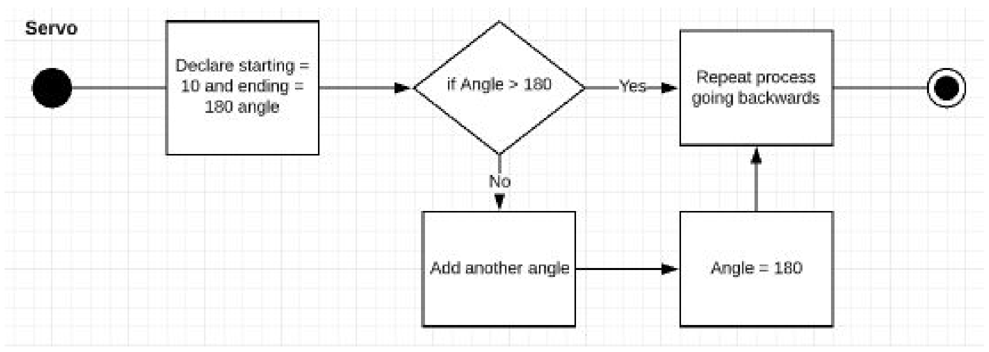
Servo motors are high torque motors that enable precise angular position, velocity and acceleration control, which is commonly used in robotics and several other applications because their rotation can be easily controlled. Servo motors have a geared output shaft to turn one degree at a time, which can be controlled electrically.

For the first time in the world of Remote Control (RC), servo motors were used to control the steering of RC cars or the flaps on an RC aircraft. They found their uses in robotics, automation, and the Arduino world, of course, with time.

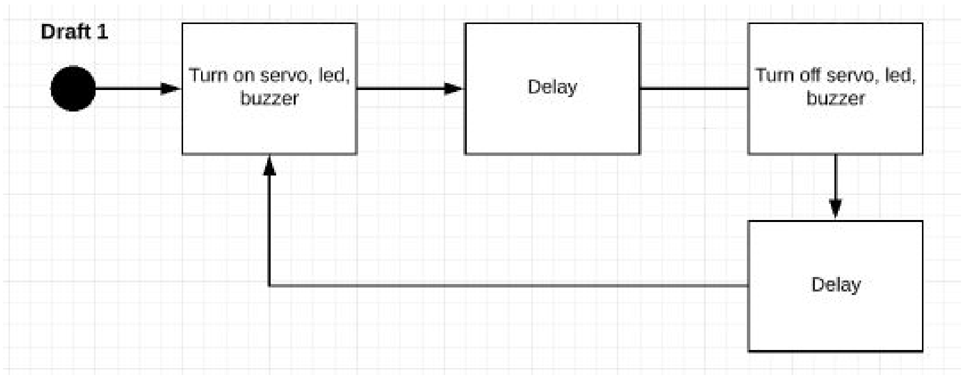
Section 2: Evidence Guide



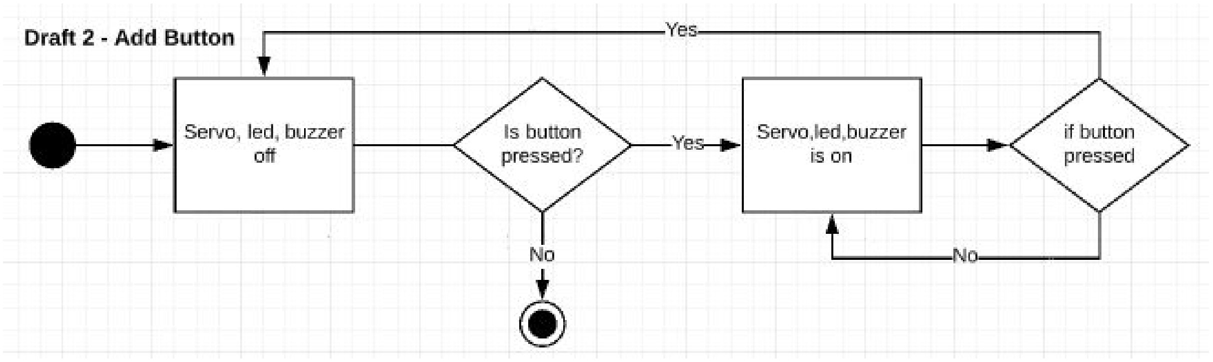


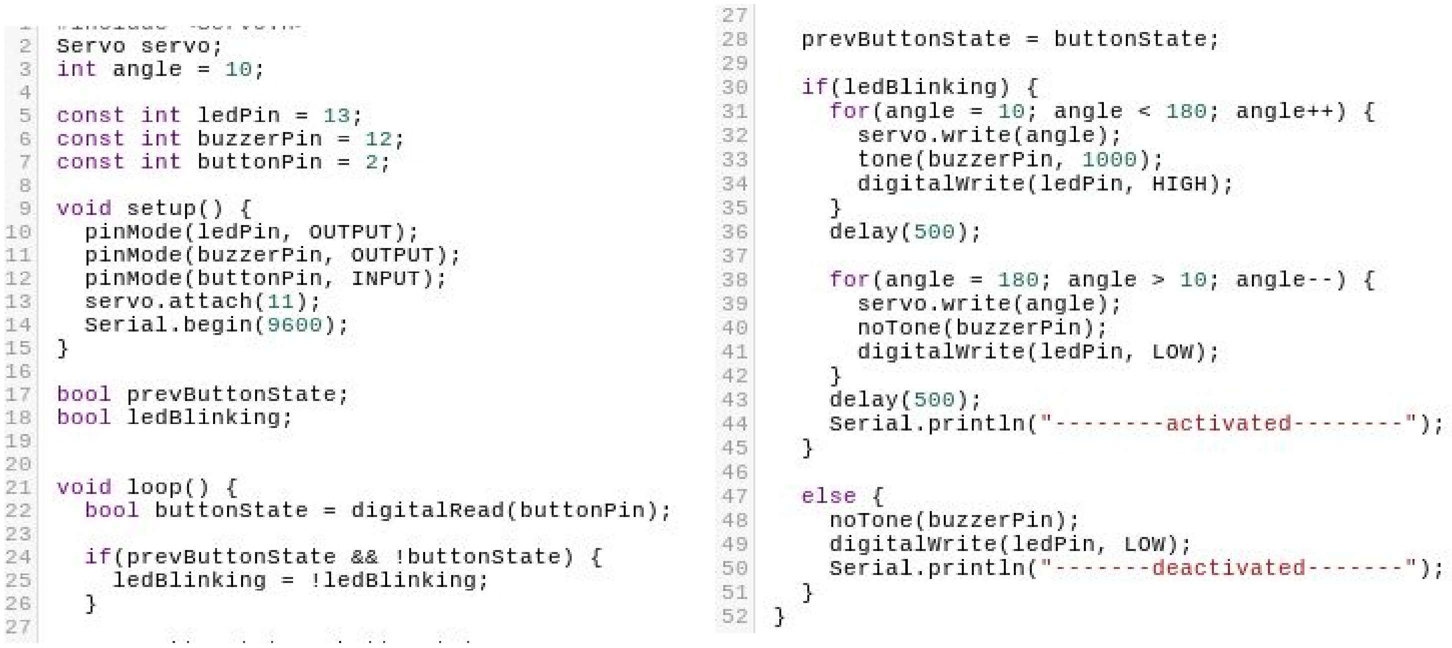

Servo
Declare starting = 10 and ending = 180 angle
if Angle > 180
Yes
No
Repeat process going backwards
Add another angle
Angle = 180

```
#include <Servo.h>
Servo servo;
int angle = 10;
void setup() {
    servo.attach(8);
    servo.write(angle);
}
void loop() {
   // scan from 0 to 180 degrees
  for(angle = 10; angle < 180; angle++)
  {
    servo.write(angle);
    delay(15);
  }
  // now scan back from 180 to 0 degrees
  for(angle = 180; angle > 10; angle--)
  {
    servo.write(angle);
    delay(15);
  }
}
```



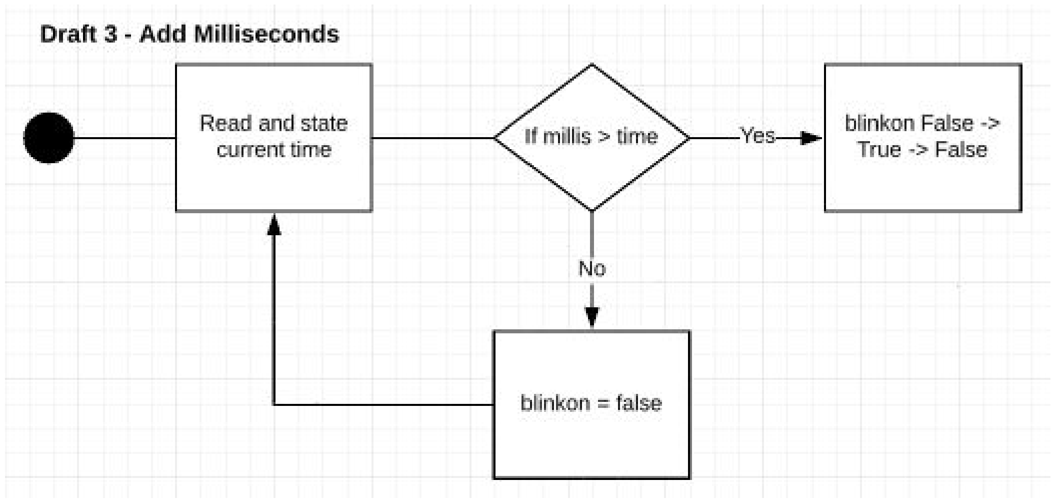
Our initial idea was basic and it was just for us to find a way to bind the codes together and to branch out from there.

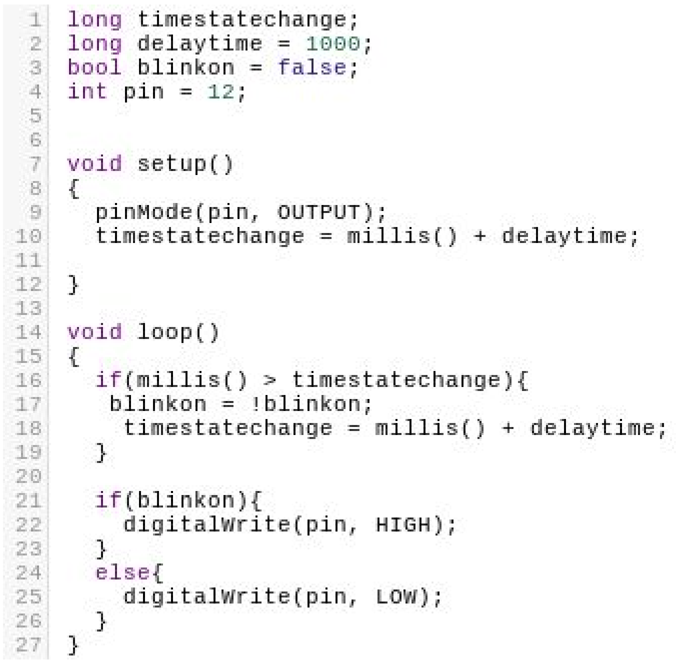




```
Servo servo;
int angle = 10;

const int ledPin = 13;
const int buzzerPin = 12;
const int buttonPin = 2;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
  pinMode(buttonPin, INPUT);
  servo.attach(11);
  Serial.begin(9600);
}

bool prevButtonState;
bool ledBlinking;


void loop() {
  bool buttonState = digitalRead(buttonPin);

  if(prevButtonState && !buttonState) {
    ledBlinking = !ledBlinking;
  }

  prevButtonState = buttonState;

  if(ledBlinking) {
    for(angle = 10; angle < 180; angle++) {
      servo.write(angle);
      tone(buzzerPin, 1000);
      digitalWrite(ledPin, HIGH);
    }
    delay(500);

    for(angle = 180; angle > 10; angle--) {
      servo.write(angle);
      noTone(buzzerPin);
      digitalWrite(ledPin, LOW);
    }
    delay(500);
    Serial.println("--------activated--------");
  }

  else {
    noTone(buzzerPin);
    digitalWrite(ledPin, LOW);
    Serial.println("-------deactivated-------");
  }
}
```

We wanted a way to turn our individual components on and off together, we used our previous knowledge of pushbuttons to create a way for the components only coming on once the button was pushed. We found flaws in this code regarding out servo motor as it seemed that it never turned off but the motor sped up.


Draft 3 - Add Milliseconds
Read and state current time
If millis > time
Yes
blinkon False -> True -> False
No
blinkon = false



```
long timestatechange;
long delaytime = 1000;
bool blinkon = false;
int pin = 12;


void setup()
{
  pinMode(pin, OUTPUT);
  timestatechange = millis() + delaytime;

}

void loop()
{
  if(millis() > timestatechange){
   blinkon = !blinkon;
    timestatechange = millis() + delaytime;
  }

  if(blinkon){
    digitalWrite(pin, HIGH);
  }
  else{
    digitalWrite(pin, LOW);
  }
}
```

We planned to add mills() after our unsuccessful button plan. For this function, when the Arduino's time catches up to the current time, it's original bool will switch to it's opposite state. Resulting with our components being able to properly fuction together and turn it self on and off.

FINAL:

https://www.tinkercad.com/things/luA8fGsABuC-programming-creative-prison-alarm/editel?sharecode=5-FrZ96wV5tse9-F4PFmCdYicWSXpVVyo49AkdCGzKA=

Research:

https://www.allaboutcircuits.com/projects/servo-motor-control-with-an-arduino/
https://www.robotshop.com/community/tutorials/show/arduino-5-minute-tutorials-lesson-5-servo-motors
https://www.instructables.com/id/Arduino-Servo-Motors/
https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/
http://www.electronics-lab.com/project/using-sg90-servo-motor-arduino/