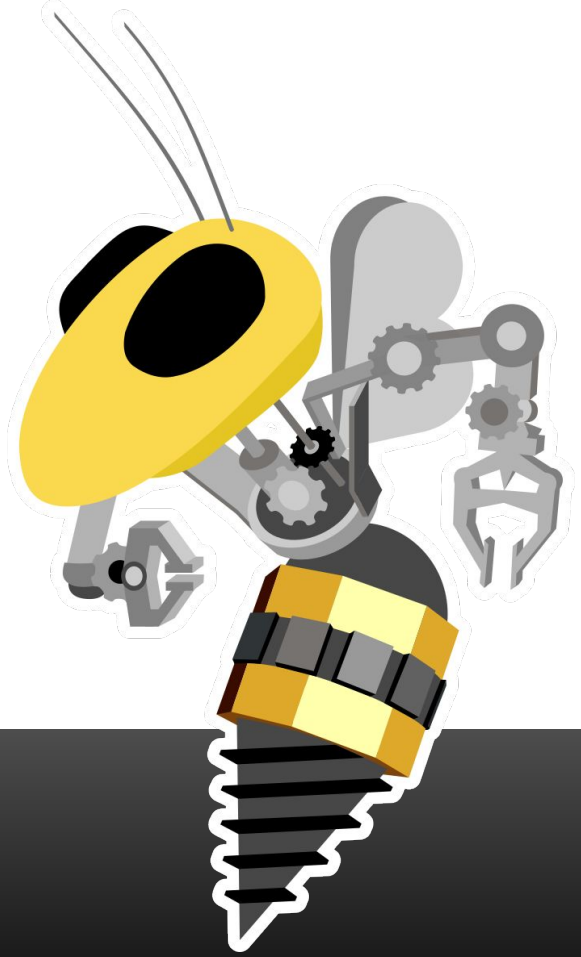
# Welcome!

## Firmware Training
## Week 2



RoboJackets
Competitive Robotics at Georgia Tech

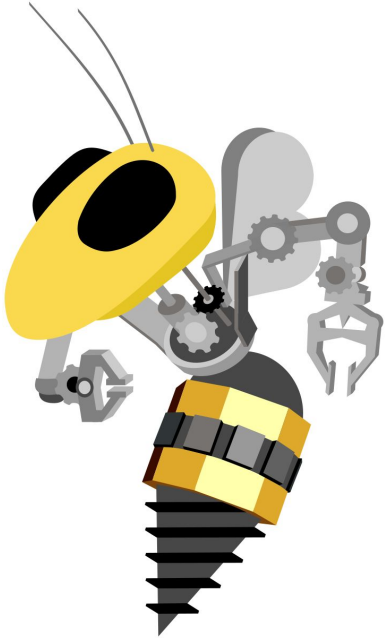
*www.robojackets.org*

# Last Week!

- Microcontrollers
- Intro to C++
- Prototyping

# This Week!

- C++ Continued
- Interrupts
- State Machines
- Lab



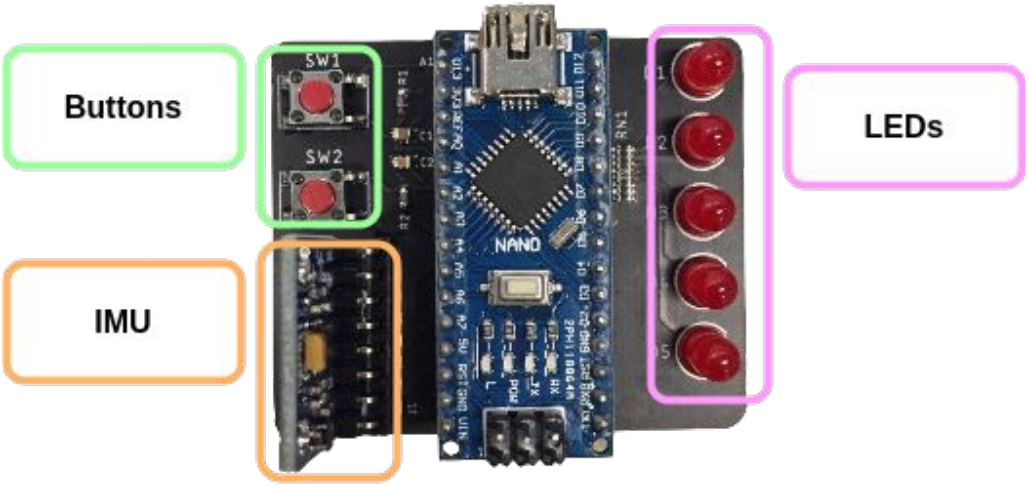
# Firmware Training Board

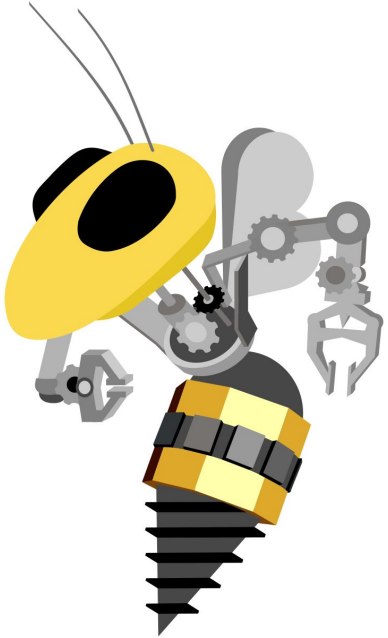
What is this thing?

RoboJackets
Competitive Robotics at Georgia Tech

# Firmware Training Board

*No more breadboards*



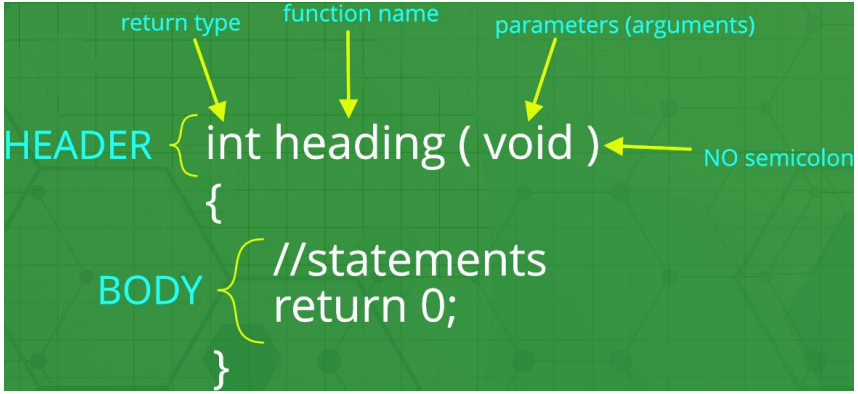


# C++ Continued

See Plus Plus

# Functions

- Helps to organize code into chunks
- Makes it easier to read and prevents duplicated code
- Define before using it in your code




```
#include<iostream>

int add(int a, int b) {
    return (a + b);
}

int main() {
    int sum;

    sum = add(100, 78);
    ... ...
}
```

function call

# Variable Scope

- Local variables
  - Within functions
  - Can’t be accessed elsewhere
- Global variables
  - What you used before (before setup)
  - Accessible everywhere in the file
- Volatile
  - Variables used in interrupts
  - Compilers check for variables that are not either assigned to or read from (dead code)
  - Since ISRs are things called by hardware and not the code the compiler doesn't understand it's still affecting the value of a variable
  - Marking the variable as volatile tells the compiler not to remove operations to a variable as it's changing in ways you can't see

```
#include<iostream>
using namespace std;
```
Global Variable
```
// global variable
int global = 5;

// main function
int main()
```
Local variable
```
{
    // local variable with same
    // name as that of global variable
    int global = 2;

    cout << global << endl;
}
```

# For Loops

- Designed to repeat code a fixed number of times
- Three part syntax
  - Initialize counter
  - Bounds check
  - Increment counter

```
// Adds a bunch of numbers
int sum = 0;
for(int i = 0; i <= 5; i++) {
    sum += i;
}
```

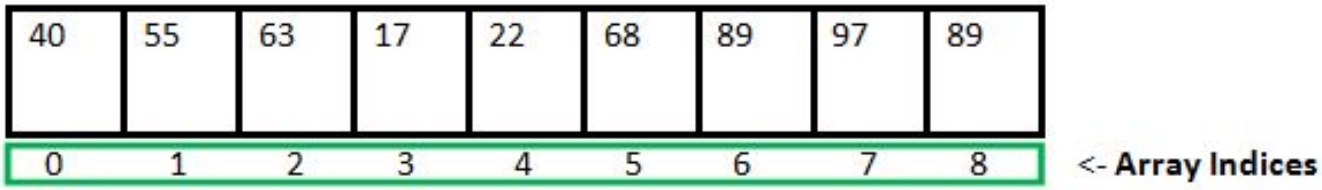
# While Loops

- Designed to repeat code until condition is met
- Three part syntax
  - Initialize counter
  - Condition check
  - Increment counter

```
// Adds a bunch of numbers
int sum = 0;
int i = 0;
while(i <= 5) {
    sum += i;
    i++;
}
```

# Arrays

- Way to organize data (collection of same data type)
- Has a fixed size at creation
- Access using an index




**Array Length = 9**
**First Index = 0**
**Last Index = 8**

# Using Arrays

- Create with *type name[size]* syntax
- Read and set data with *name[index]* syntax
  - You should always set before you read

```
int array[10];
// Loop through array and sets value
for(int i = 0; i <= 9; i++) {
    array[i] = i;
}
```

# 2D Arrays

- Make an array of arrays (multiple dimensions)
- Created by adding another dimension *[size1][size2]*

| | Column 0 | Column 1 | Column 2 |
|---|---|---|---|
| Row 0 | x[0][0] | x[0][1] | x[0][2] |
| Row 1 | x[1][0] | x[1][1] | x[1][2] |
| Row 2 | x[2][0] | x[2][1] | x[2][2] |

# Structs

- Ways to organize data of same or different types
  - A vector has multiple different quantities
- Commonly used to package related data

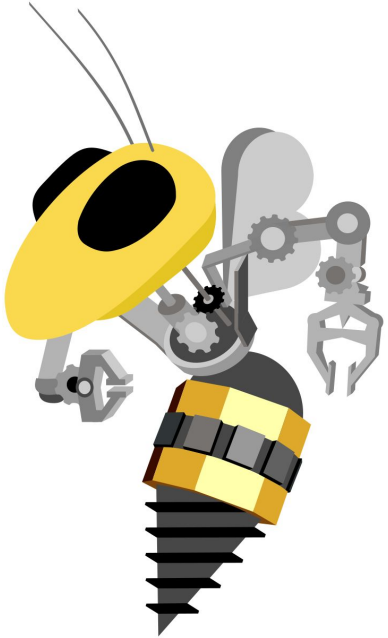
```
typedef struct
{
    float x;
    float y;
    float z;
} Vector3D;
```

# Using Structs

- Create as you would a variable, *type name* syntax
- Read and set using *name.field* syntax
  - You should always set before you read

```
// Creates vector
Vector3D vector;
vector.x = 1.0;
vector.y = 0.5;
```

```
typedef struct
{
  string title;
  int pages;
  float price;

} Book;

Book book1;
book1.title = "Fahrenheit 451";
book1.pages = 158;
book1.price = 15.99;

Book book2;
book2.title = "Catch-22";
book2.pages = 453;
book2.price = 18.00;

Book bookshelf[2] = {book1, book2};
printf("%f", bookshelf[1].price);
```



# Interrupts

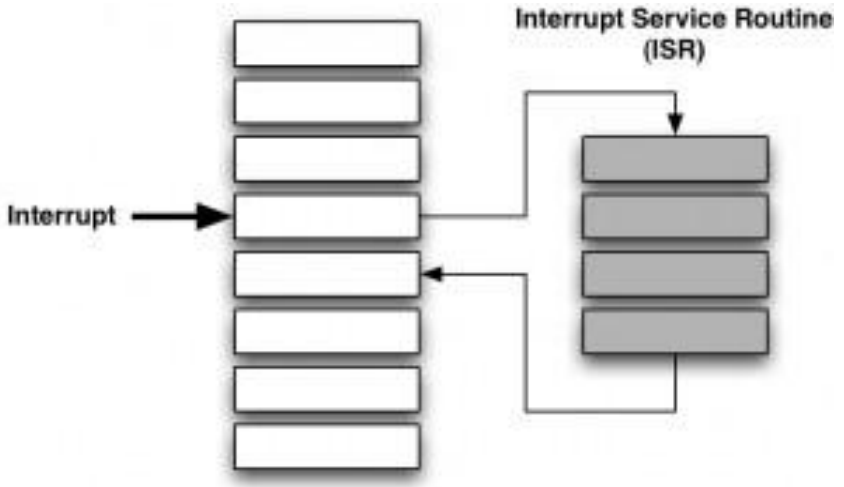
Hey Mom. Mom. Mom. Mom.
Mo- WHAT!

# What are Interrupts

- Mechanism built into processors to run a function when an event occurs
  - Can be hardware (a pin) or software (a timer)
- The function that gets called is known as an ISR (interrupt service routine)
- It stops (interrupts) the main code before returning back to the main code

# Using Interrupts

- Allows us to not waste time checking if a device is ready (polling)
- Instead the device just tells us it is ready (interrupts)
- Returns to our normal code right after

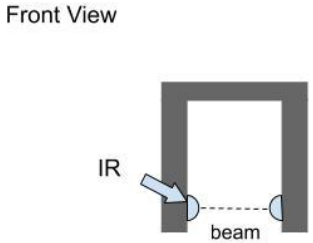


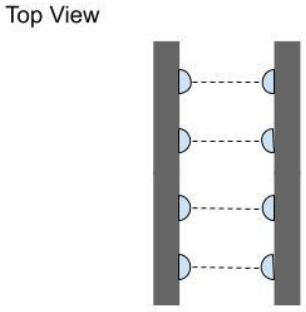
# Arduino and Interrupts

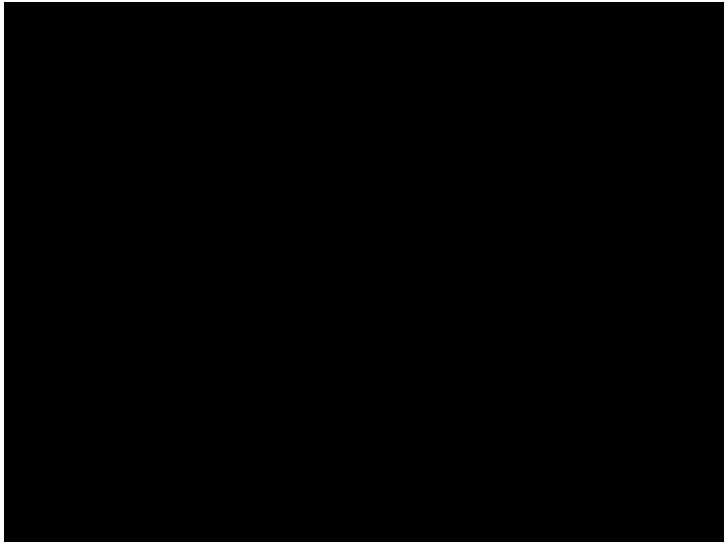
- Arduino provides the attach interrupt method for this
  - Triggers an interrupt when the signal to an input pin changes
  - *attachInterrupt(pin, ISR, trigger mode)*
- Mode:
  - Rising/Falling/Change Edge trigger
  - Calls ISR on 0 to 1 change, 1 to 0 change, or both

# RoboCup Ball Speed Example

- Want to calculate the speed of the ball after the robot kicks ball
- IR sensors determine when the ball has passed
  - Using the time this occurs and distance between sensors can calculate average speed



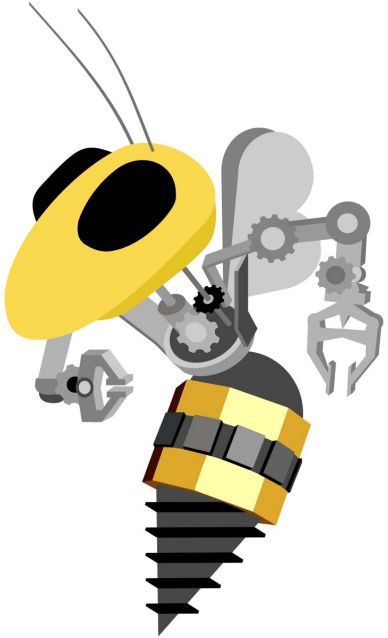



# Code: Interrupt Setup

```
const double r_sensors = 0.1905; //distance between sensors (m)

#define sensor1 3 //TX -> sensor 1
#define sensor2 2 //RX -> sensor 2
#define sensor3 0 //SDA -> sensor 3
#define sensor4 1 //SCL -> sensor 4

unsigned long time_sensor[4]; //time sensor was triggered (µs)
int j;
double mean_velocity;

void setup() {
  Serial.begin(115200);

  pinMode(sensor1, INPUT);
  pinMode(sensor2, INPUT);
  pinMode(sensor3, INPUT);
  pinMode(sensor4, INPUT);

  attachInterrupt(digitalPinToInterrupt(sensor1), interrupt1, FALLING);
  attachInterrupt(digitalPinToInterrupt(sensor2), interrupt2, FALLING);
  attachInterrupt(digitalPinToInterrupt(sensor3), interrupt3, FALLING);
  attachInterrupt(digitalPinToInterrupt(sensor4), interrupt4, FALLING);
}
```

# Code: Creating the ISR

```
// an interrupt for each sensor
void interrupt1 () {
  noInterrupts();
  time_sensor[0] = micros();
  interrupts();
}
void interrupt2 () {
  noInterrupts();
  time_sensor[1] = micros();
  interrupts();
}
void interrupt3 () {
  noInterrupts();
  time_sensor[2] = micros();
  interrupts();
}
void interrupt4 () {
  noInterrupts();
  time_sensor[3] = micros();
  interrupts();
}
```

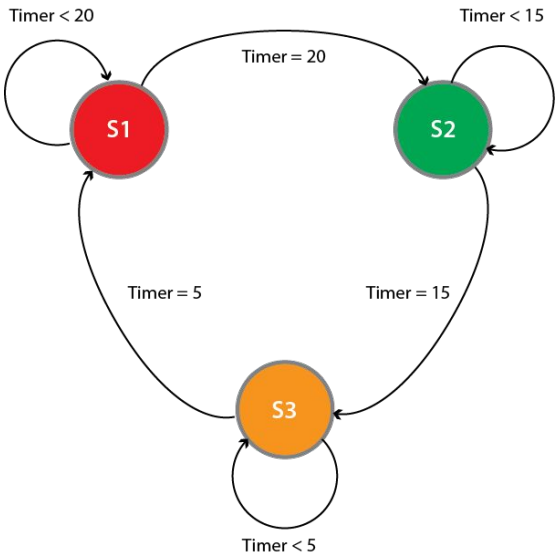


# State Machines

"Ferst Driiive, walk sign is on to cross, Ferst Driiiive"

# What is a State Machine?

- Stores a status, or state, at any given time
- Takes in inputs and gives outputs to interact with other devices
- Switches states based on current states and inputs




Traffic Light State Machine

# Purpose of a State Machine

- Provide a way to use a sequence or history of inputs, not just current input values
- Restrict behavior of a system to certain actions based on a variety of inputs
- Provide a sort of memory - tied to history of inputs
  - Combination lock
  - Traffic signals && crosswalks
  - Robots!

# Usages of State Machines

- Making sure things happen in the right order (startup)
- Controlling behavior of robots better - output based off of states (stable), not directly by inputs
- A CPU (and a Microcontroller)!

# Types of State Machines

- **Moore state machine** - outputs are based only on current state
- Mealy state machine - outputs are based on the current state and the input (transitions)
- We typically use a Moore State Machine

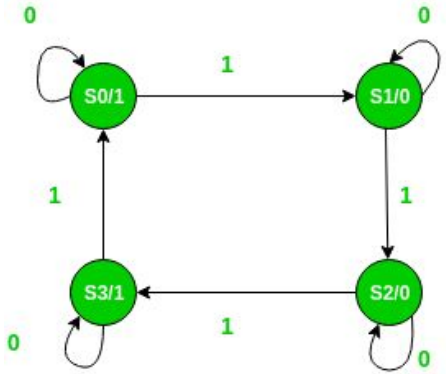



**Figure -** Moore machine

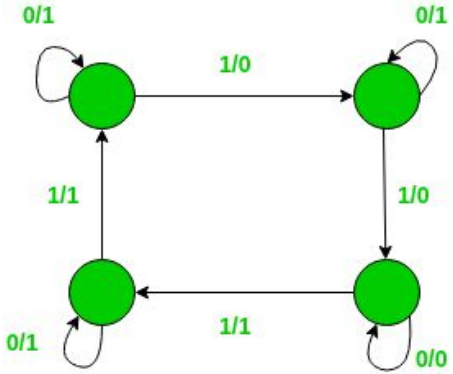
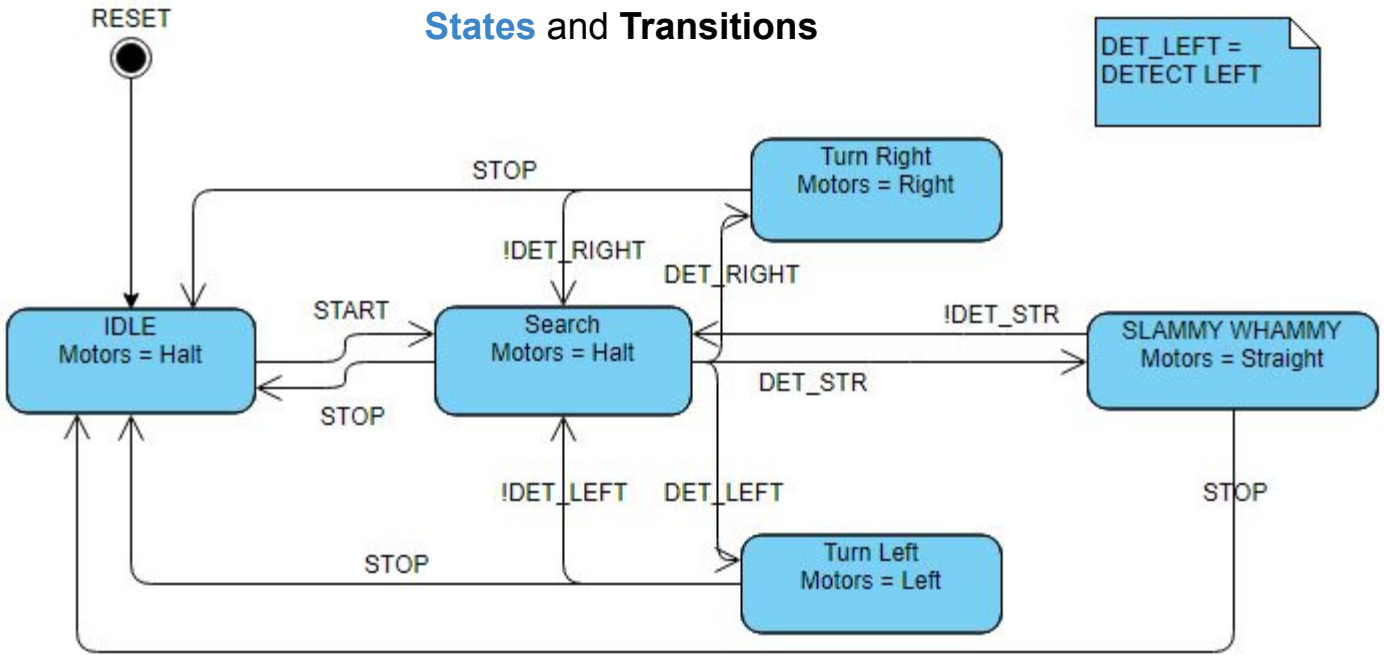


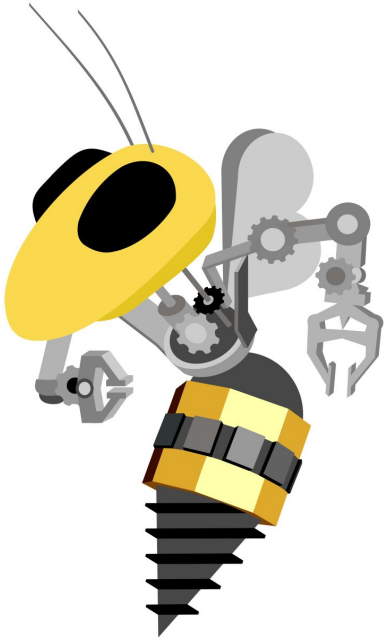

**Figure -** Mealy machine

# Inputs and Outputs

- Inputs frequently include sensors or buttons
  - Prefer boolean state transition
- Outputs can be motors or LEDs
  - Behaviors (running motors, running code) of the robot

# Example Diagram (RoboWrestling)





# Lab Time

# Lab Info

- Create a counter state machine
- Write interrupts for each button
  - One to count up
  - One to count down
- Implement state machine
- Display state using board LEDs