EECS 201 - System Design Fundamentals

Final Project - Fall 2019

OBSTACLE DETECTOR DEVICE

Project Report

Supervisor: Prof. Dr. Mehmet Serkan Apaydın

CONTENT

TEAM MEMBERS

FULL NAME	NUMBER	RESPONSIBILITY
FURKAN GÜNEŞ	217421896	Circuitry and Report
ALI REZA IBRAHIMZADA	218012083	Code and Report
BASIL ELKHALIFA	218191285	Code and Circuitry
OUMAIMA HNIKI	218640797	Circuitry and Report

1. INTRODUCTION

In this project, we implement an Obstacle Detector Device using the state machine paradigm. There are certain number of states to this state machine which can be activated in a specific input range. Based on the limitations set in the project manual, states can change to each other based on their input. As an important rule in state machines, the values of the next state and output directly depends on input. The materials used in this project are very simple, an ultrasonic sensor which detects the distance from nearest obstacles, LEDs to display the distances and a buzzer as a warner when obstacles gets too close. This is only an initial start to this device, and it can be further developed to improve its performance.

Moreover, this project is to practically use and understand the idea of PCAP (Primitives, Combinations, Abstractions and Patterns). Understanding the main idea of PCAP and to be able to implement it in complex systems is for sure the most important thing engineers do. Being able to connect those primitives, make bigger abstractions, implement the same functionality in a hierarchy and ... are examples that relies in PCAP.

The detailed description of the different parts are available below.

2. GOAL

The aim of this project is to apply State Machines on CircuitPython METRO M0 Express board, Ultrasonic Sensors, a Buzzer, control LEDs (green-yellow-red) using Python. This project combines these equipment in order to create an Obstacle Detector Device. Such an Obstacle Detector Device using a state machine paradigm is a good practical of PCAP (Primitives, Combinations, Abstractions and Patterns). It helps us to understand how simple primitives which perform single functions can be combined to perform more complex tasks. As mentioned earlier, dividing the functionalities into different states actually makes the design process much more easier and it does not matter how the system can get more complex, a good design and paradigm can always take care of complexity. In this project, based on the input values, we change states and use the hardware to show certain outputs for the user.

3. MATERIALS

3.1. Metro M0 Board

Adafruit's beginner-oriented flavor of MicroPython, a small but full-featured version of the popular Python programming language created specifically for use with board level circuitry and electronics.

3.2. Control LEDs

LEDs are light sources that are used for showing some output values according to the purpose of the project. For this project, red, green and yellow control LEDs are used to display the distance of the obstacle.

3.3. Resistors

Each LED needs to have a resistor in series due to the fact that the unexpected or unrestricted current will burn it out immediately. Resistors restrict the flowing current in order to protect the LEDs.

3.4. Buzzer

Buzzer is used to show the distance between obstacle and object thanks to the sound it makes. If the distance is closer than the given value, it starts working and it creates sound like "beep".

3.5. Ultrasonic Sensor

Ultrasonic Sensors are equipment that measures the distance through using ultrasonic sound waves. They are also a transducer which sends ultrasonic impulses and receive them.

3.6. Bread Board

Breadboard is fundamental material that is used for building circuit. For building a prototype, testing out new parts and trying to test something into the circuit breadboard is one of the magical tools for these steps.

4. Hardware Design (Schematic) - Device Building Stage

4.1. In the first stage, we attached the LEDs and the resistors to the breadboard. We then connected four wires. Three of the four wires connects the three LEDs' negative polarities together. The last wire connects all negative polarities to the Metro M0 ground pin (GND): as seen in Figure 2.

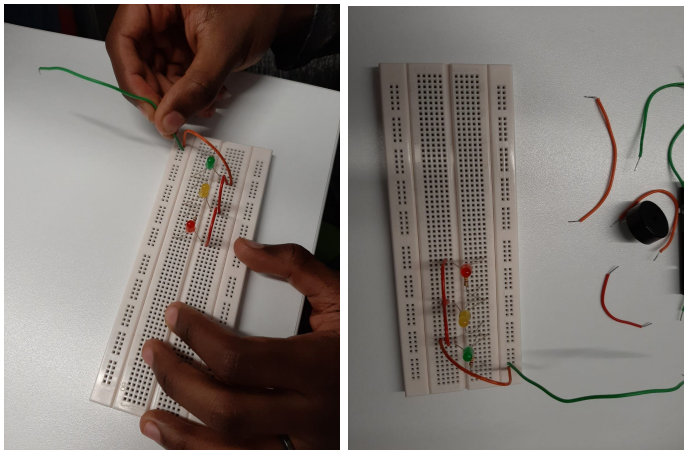

Figure 1: Attaching LED pins and resistors on the bread board

4.2. In the second stage, we connected the LEDs' negative polarities to the ground pin on the Metro M0 board.

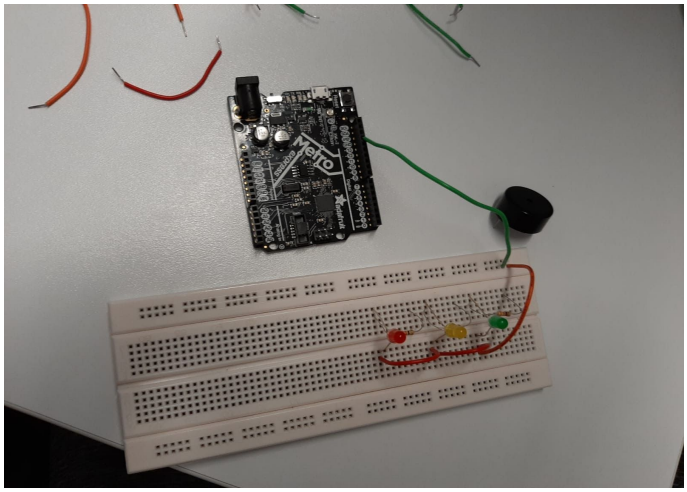

Figure 2: Connecting the bread board to the Metro Board

4.3. In the third stage, we connect the LEDs to pre-defined pins on Metro Board

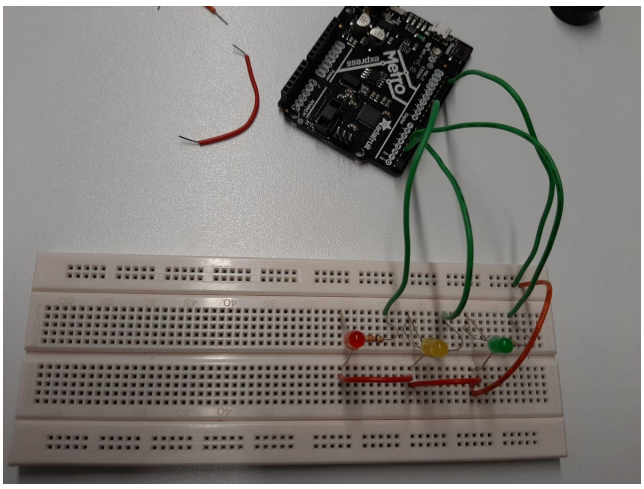

Figure 3: Connecting LEDs to specific pins on Metro Board

4.4. In the fourth stage, we attached the buzzer and its resistor to the breadboard; the negative side of the buzzer is connected to the ground. We then connected the buzzer to its pre-defined pin.

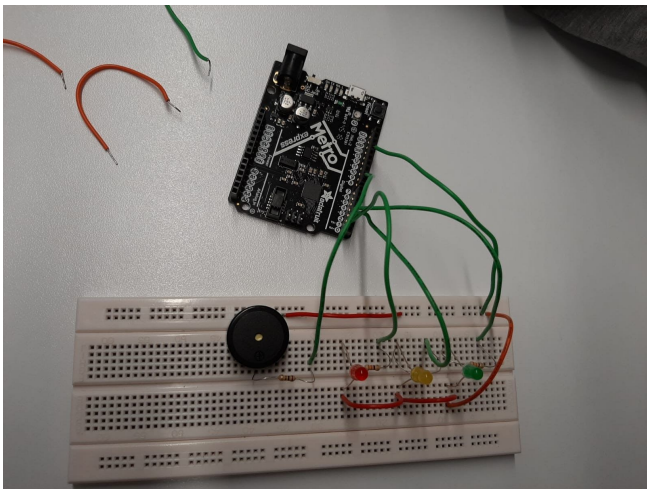

Figure 4: Connecting buzzer and ground pins

4.5. In the final stage, we attached the ultrasonic sensor to the breadboard. We then connected each ultrasonic sensor's pins to the appropriate pins on the board. We also powered the upper five blocks with 5 volts of power.

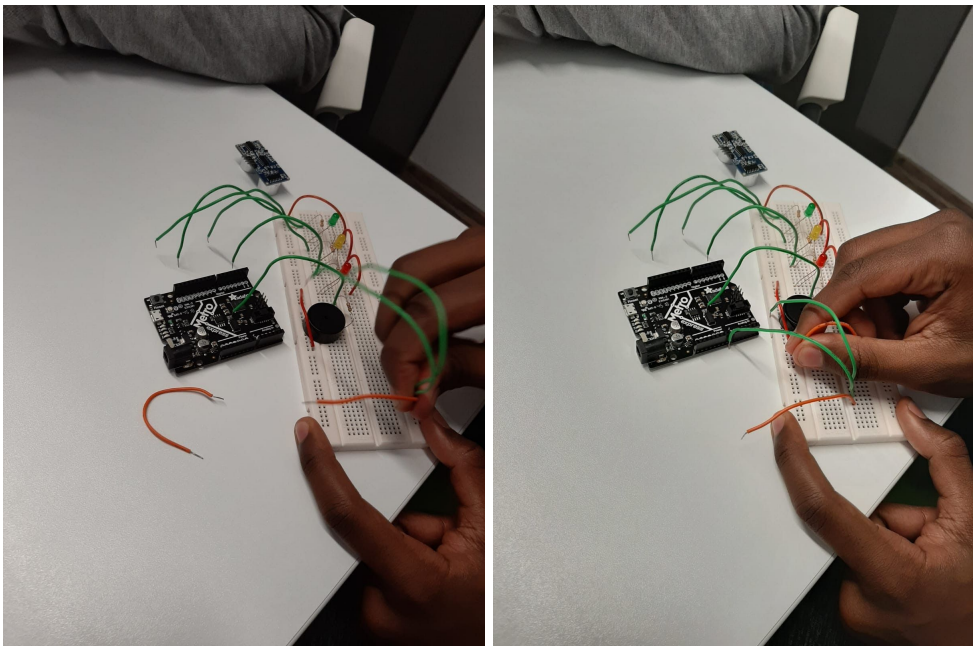

Figure 5: Connecting ultrasonic sensor to the metro board

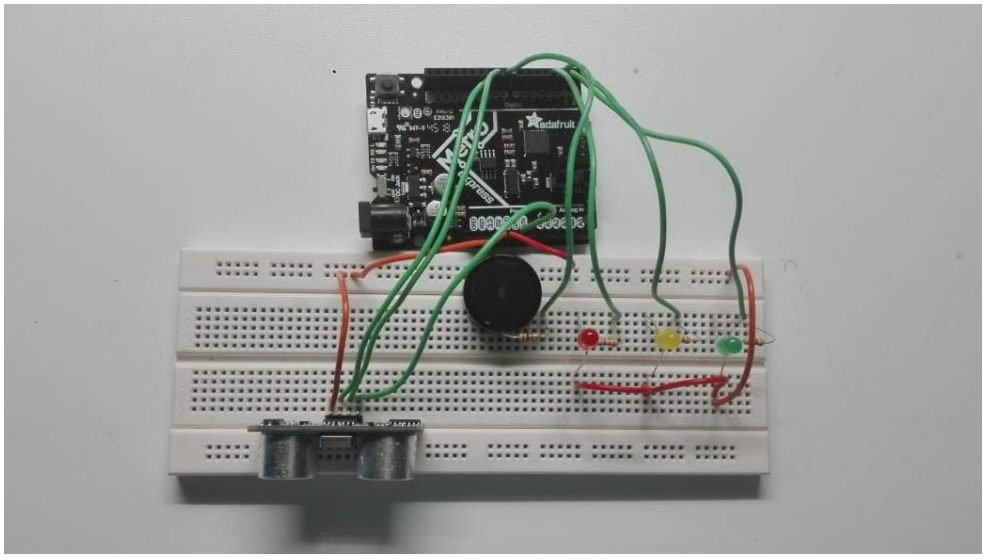

Figure 6: Final connections of the Obstacle Detector Device

5. Software

The software which makes the circuit functional uses the abstraction of a state machine. In a state machine, there is an input at each state, this input is used to determine the output and the next state. Everything starts by importing the necessary modules and using those to read the drivers for hardware we used in the project such as, buzzer, sonar, LEDs, ... Then we have a class *SM* which implements the fundamental operations of a state machine. The start method is responsible for setting the state at the very beginning of state machine to the initial state. The step method takes care of new state and output. By calling the getNextValues() method from the child class, first it considers if the state machine is in closeState (a state in which the distance is <= 25). If so, it starts a timer and does not change the state, if the state machine stays in the closeState for more than 5 seconds, then it changes the state to finalState and the program ends. However, if the distance is anything except less than 25, then the step sets the new state and returns an output based on the values returned by getNextValues(). self.isFinalState is a class level boolean attribute which checks if state machine is in closeState, if so, then the timer will be started using the time module. self.getTimeDifference() method is used to calculate the time difference between the current time and since the most recent timer started. self.transduce() is a method which deals with the new input values and calls self.step(). It uses a class level boolean attribute from child class to check if it is initial state, if so, then it will call self.start(). It returns the new state and output after each step.

The code further continues to *ObstacleDetector(SM)* class which inherits from SM class. It's attributes and their purposes are already described in above. The self.initialize() method encapsulates the functionality of initialState. Each different LED calls this method at the initialState and it performs a functionality described in project manual. The self.getNextValues(state, input) method gets the current state and input as parameters and returns the next state and output. As described in project manual, there are 4 states that can be repeated based on input values. Each state has been implemented inside this method and the conditions helps the state machine to return an appropriate next state and output based on the value of input.

Outside classes, by creating an object of ObstacleDetector(), we use appropriate methods to start the state machine and make the circuit functional. When the triggered waves of sonar cannot return back (when it does not reflect on an obstacle), the circuit gives a timeout error (RuntimeError). We handle this using an except case and later we set the distance to 200 (because the interval of fifthState starts from 200 to infinity). Afterwards, by calling the transduce method, we give the distance value from sonar as a

parameter to this method and the operation of state machine starts. It calls step() and getNextValues() and returns an appropriate output and changes the state based on input. Finally, if the state machine stays in the closeState for more than 5 seconds, the state gets changed to finalState and the program ends. The condition at the very end of the code checks this, prints a specific output and turns on the red LED and buzzer for 5 seconds. The complete python script is below.

```
# importing necessary modules for project
import board
import digitalio
import time
import adafruit_hcsr04
import pulseio

# reading drivers for hardware and introducing them to computer
buzzer = pulseio.PWMOut(board.D8, variable_frequency=True)
sonar = adafruit_hcsr04.HCSR04(trigger_pin=board.D10, echo_pin=board.D9)
red_led = digitalio.DigitalInOut(board.D3)
yellow_led= digitalio.DigitalInOut(board.D2)
green_led = digitalio.DigitalInOut(board.D1)
red_led.direction = digitalio.Direction.OUTPUT
yellow_led.direction = digitalio.Direction.OUTPUT
green_led.direction = digitalio.Direction.OUTPUT

class SM:
    def start(self):
        self.state = self.startState

    def step(self, inp):
        (s, o) = self.getNextValues(self.state, inp)
        if s != 'closedState':
            self.state = s
            self.isFinalState = False
            self.start = 0
        elif s == 'closedState' and not self.isFinalState:
            self.start = time.time()
            self.isFinalState = True
```

```
        elif s == 'closedState' and self.getTimeDiffernce(self.start) > 5:
            self.state = 'finalState'
        return o

    def transduce(self, input):
        if not self.isInitialState:
            self.start()
            self.isInitialState = True
        o = self.step(input)
        return (self.state, o)

    def getTimeDiffernce(self, old_time):
        return time.time() - old_time


class ObstacleDetector(SM):
    startState = 'initialState'
    isInitialState = False
    isFinalState = False

    def initialize(self, led):
        led.value = True
        time.sleep(2)
        led.value = False
        time.sleep(1)

    def getNextValues(self, state, inp):
        if state == 'initialState':
            self.initialize(green_led)
            self.initialize(yellow_led)
            self.initialize(red_led)
            buzzer.duty_cycle = 2**15
            time.sleep(2)
            buzzer.duty_cycle = 0
            return ('Initial State', 'Obstacle Detector Device is Working!')
```

```
        elif 120 < inp < 200:
            red_led.value = False
            yellow_led.value = False
            green_led.value = True
            return ('secondState', 'You are safe')

        elif 50 < inp <= 120:
            green_led.value = False
            red_led.value = False
            yellow_led.value = True
            return ('thirdState', 'Slow Down Your Speed')

        elif 25 < inp <= 50:
            yellow_led.value = False
            green_led.value = False
            red_led.value = True
            buzzer.duty_cycle = 0
            return ('fourthState', 'DANGER TOO CLOSE')

        elif inp >= 200:
            red_led.value = False
            green_led.value = True
            yellow_led.value = True
            buzzer.duty_cycle = 0
            return ('fifthState', 'Distance Error')

        elif inp <= 25:
            return ('closedState', '')

obj = ObstacleDetector()
while True:
    try:
        sonar_distance = sonar.distance
    except RuntimeError:
```

```
        sonar_distance = 200
    (s, o) = obj.transduce(sonar_distance)
    print(o)

    if s == 'finalState':
        print('The Device Stopped! Calling Emergency Services')
        green_led.value = False
        yellow_led.value = False
        red_led.value = True
        buzzer.duty_cycle = 2**15
        time.sleep(5)
        break
```

6. State Diagram

State Diagrams are a best way to visualize how a state machine works. They show all possible states and how they are connected between each other. In our project, we have 6 different states. The Obstacle Detector circuit starts from the initial state, and then based on new inputs, the state can change to 5 other states as shown in below. 4 states can change between themselves in this state machine based on inputs during implementation. Initial and final states cannot be repeated as mentioned in the project manual. Figure 7 describes all possible interactions between states.

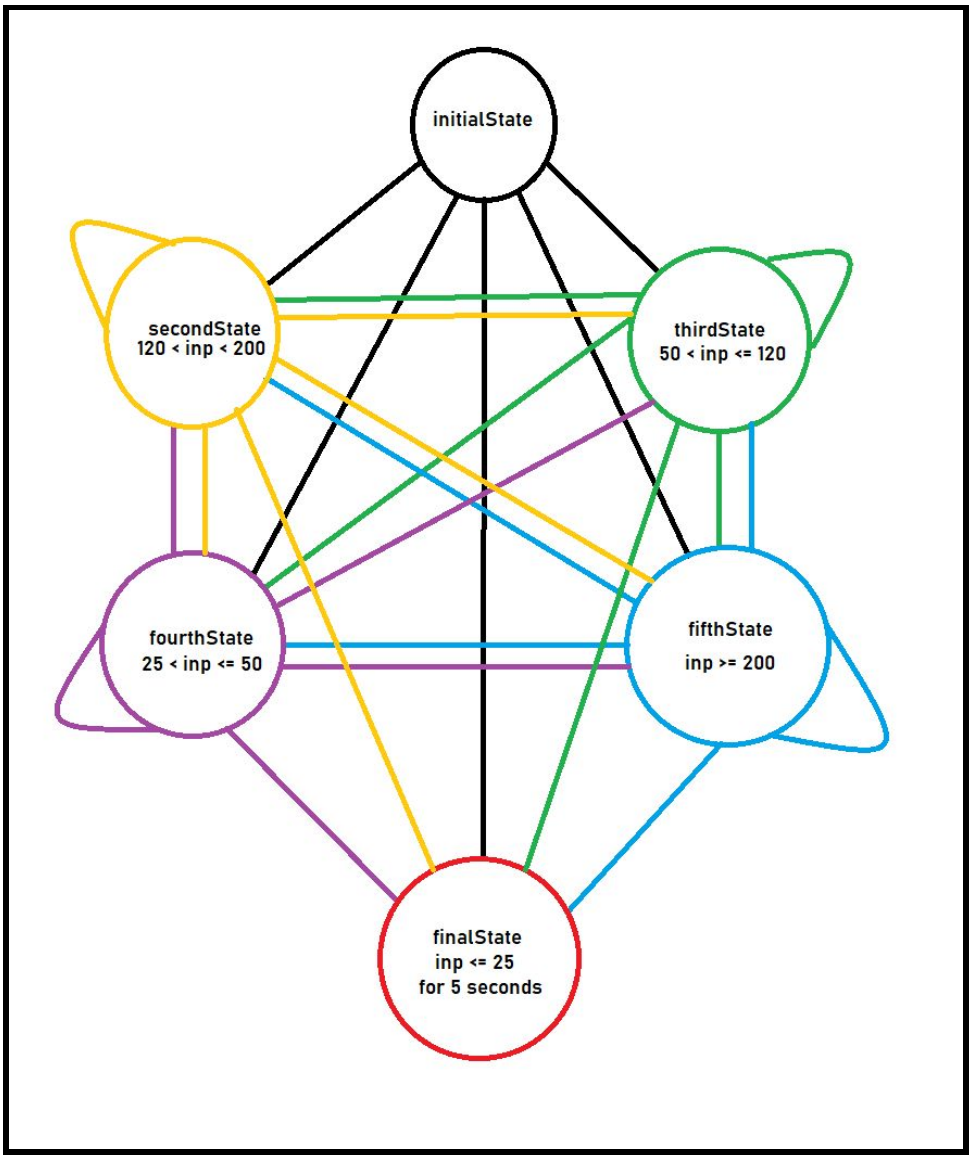

Figure 7: State Diagram of Obstacle Detector Device

7. Challenges

As this implementation is a basic and fundamental version of Obstacle Detector Device, there are a lot of challenges we can think of. The biggest challenge is the sensor can only sense obstacles which are positioned in front of it. A good example of this can be observed in cars. Whenever parking a car with smart system, have you ever thought why a warning alarm rings when the car gets closer and closer to other cars? Well there is a developed version of Obstacle Detector Device available in those cars which can detect nearby objects not only in one direction, but in all directions.

One other challenge can be its working environment. Our version of Obstacle Detector Device works only in dry and plain area which is quite a comfortable environment. However, more developed devices function in wet and unsuitable places. Our implementation of this device is also breakable and making a good hard covering for it can assure its long term functionality.

8. Improvements

The project made is only a basic implementation of Obstacle Detector Device. It can be improved by introducing new features such as its ability to spin each second in order to catch the obstacles in all surroundings. Making it to observe 3D environment could be another nice feature. On the other hand, it has been witnessed in practice that Ultrasonic Sensor does not always gives an output. For example when it starts for the first time and if there is no obstacle in front of it, the triggered waves cannot reflect and return. Using more standard and high quality hardware can improve the functionality of the device a lot.

Making the device a bit smarter can be another improvement we can make. Some obstacles are not going to damage us if we come into contact with them. This data is collect-able from previous experiences and it can be used to train some statistical learning algorithms in order to predict whether the obstacle is harmful or not. One may want to set a camera for the device and use it to train the models, others may want to use the distance pattern received at each interval of time.

9. Applications

Having the ability to detect obstacles and measure the distance between a detector and an obstacle accurately comes with many applications. For example, modern cars use the same concept of detecting obstacles to aid drivers in parking; since, it is difficult to see obstacles or/and objects behind a car, obstacle detectors are usually implemented in the back of the car to warn the driver about the obstacles. They use similar systems to the system that was explained above: state machines.

The Obstacle Detector, explained in the introduction, can be implemented in a tank as a liquid level control. Changing the outputs to something more relative to the

tank, like “Tank is half way full” or “Tank is full”, allows us to develop a liquid control system. Let us suppose that the tank is empty. First, the initial state that tests if all components work should start. The sensor then can find the distance between itself and the liquid used in the tank and goes to the appropriate state (assuming that the sensor is placed on top of the tank) . If the liquid reaches a level where leaks can occur, the obstacle detector should run the final state, which is to wait for 5 seconds for the liquid level to decrease, if the liquid does not decrease, the sensor should shut off and inform the user or call the emergency.

10. Conclusion

All in all, from what we have discussed so far, the conclusion to be drawn is that the obstacle detector is a device used to detect and measure how far an object or/and obstacle accurately is. It uses 5 main components: Adafruit METRO M0 board, control LEDs, resistors, a buzzer, and an ultrasonic sensor. It uses a state machine implementation to organize the states that should run based on a given input. The device can be used in various applications such as a car’s obstacle sensor. The device can be improved by adding more components to make the device more accurate at detecting obstacle or/and object around it.

11. Reference(s)

- Massachusetts Institute of Technology OpenCourseWare Program
    - https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-01sc-introduction-to-electrical-engineering-and-computer-science-i-spring-2011/resource-index/
- Circuit Basics
    - https://www.youtube.com/channel/UC1efD8NIf7viH9Q_EWP6L5A
    - http://www.circuitbasics.com/
- AdaFruit Institution
    - https://learn.adafruit.com/
    - https://www.youtube.com/channel/UCpOlOeQjj7EsVnDh3zuCgsA