Implementation of Traffic Light System Using Raspberry Pi and PCA 9685

Prepared by:

Shubham Agarwal (11013345)

Dept. of Information Technology

Under Guidance of

Prof. Alfred Moos

***Abstract* – The purpose of this project is to provide a safe traffic light system which can be used to control the traffic on one lane. This systems consists of two traffic lights which controls the traffic over a one narrow lane. In order to achieve this the LED’s in form of traffic lights are connected with PCA 9685. Further PCA 9685 is connected with Raspberry Pi through an I2C bus. PCA 9685 is controlled by Raspberry Pi using a python program which controls both the traffic lights.**

Keywords: Raspberry Pi, PCA 9685, traffic light, LED, Python

I. Aim

To control the traffic lights and implement smooth flow of 2 – way traffic over a bridge of one lane.

II. Introduction

With the increase in number of vehicles on roads it has become very important to have a good and reliable traffic light system. In this project we will be implementing such traffic light system which can be used to control 2 – way traffic on a narrow one lane bridge.

This system will be consisting of two traffic lights which will be controlled in four phases. Between each phases there will be a time delay in order to allow the traffic to behave accordingly. In phase 1, traffic light 1 will be Red stopping the traffic whereas traffic light 2 will be green allowing the traffic to go. In phase 2, traffic light 1 will glow both Red and Yellow signalling vehicles to be ready to go while traffic light 2 will become Yellow signalling vehicles to slow down and stop. In phase 3, traffic light 1 will become Green signalling traffic to go whereas traffic light 2 will become Red stopping the traffic. In phase 4, traffic light 1 will become Yellow signalling traffic to slow down to stop whereas traffic light 2 will glow both Red and Yellow signalling traffic to get ready to go.

These phases will be repeating in order to control the flow of traffic on one lane.

Architecture

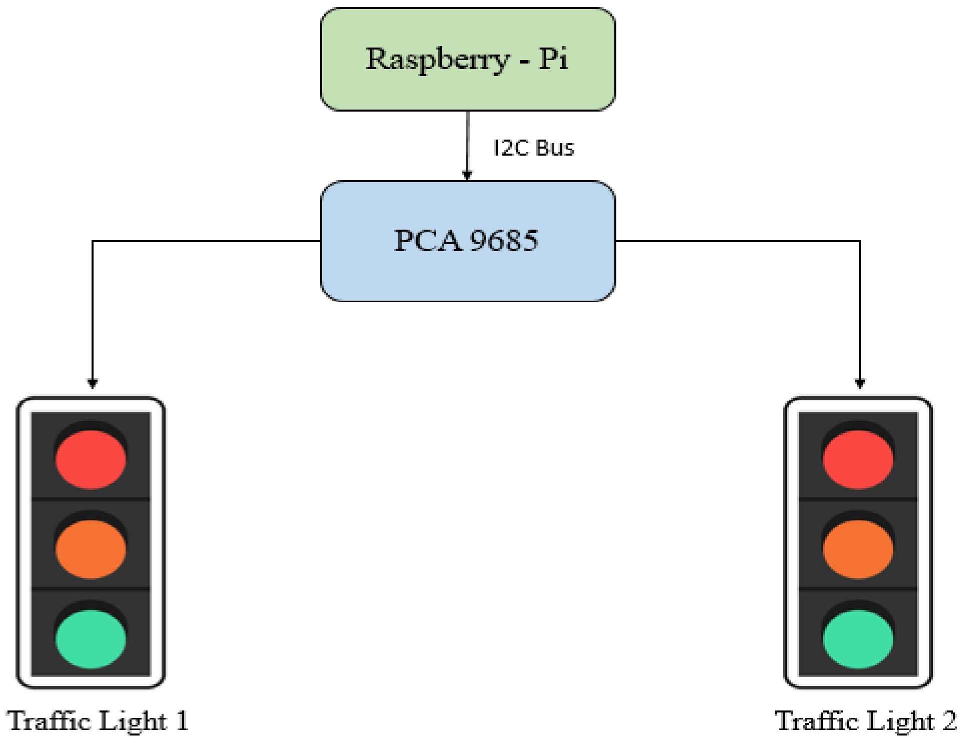


Fig 1. Architecture for Traffic Light System

The complete model consists of the following components:

i. Raspberry Pi 3
ii. PCA 9685
iii. I2C bus
iv. LED's representing traffic lights

PCA 9685 is connected to Raspberry Pi through an I2C bus. Traffic light 1 consists of 3 LED's of colour Red, Yellow and Green representing traffic signal for traffic light 1. Traffic light 2 also consists of 3 LED's of colour Red, Yellow and Green representing traffic signal for traffic light 2. All the LED's of both the Traffic lights are connected to different channels of PCA 9685. A python program is run on Raspberry Pi which controls both the traffic lights in 4 phases.

III. Hardware and Software Used

Hardware:

A. Raspberry Pi 3:

Raspberry Pi is a series of small single board computers developed in United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries [1]. In this project we have used Raspberry Pi 3 Model B which is the third generation of Raspberry Pi boards.

Specification for Raspberry Pi 3 are as follows [2]:

i. Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
ii. 1GB RAM
iii. BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
iv. 100 Base Ethernet
v. 40-pin extended GPIO
vi. 4 USB 2 ports
vii. 4 Pole stereo output and composite video port
viii. Full size HDMI
ix. CSI camera port for connecting a Raspberry Pi camera
x. DSI display port for connecting a Raspberry Pi touchscreen display
xi. Micro SD port for loading your operating system and storing data
xii. Upgraded switched Micro USB power source up to 2.5A

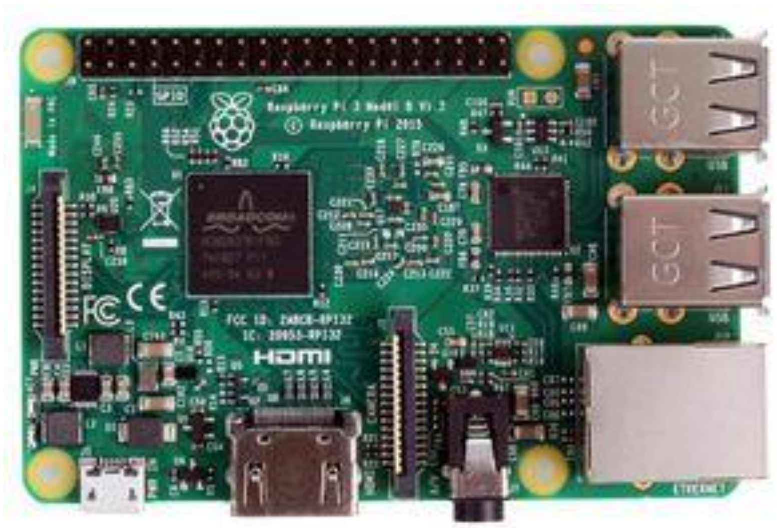


Fig 2. Raspberry Pi 3 [3]

B. PCA 9685:

The PCA9685 is an I²C-bus controlled 16-channel LED controller optimized for Red/Green/Blue/Amber (RGBA) colour backlighting applications. Each LED output has its own 12-bit resolution (4096 steps) fixed frequency individual PWM controller that operates at a programmable frequency from a typical of 24 Hz to 1526 Hz with a duty cycle that is adjustable from 0 % to 100 % to allow the LED to be set to a specific brightness value. All outputs are set to the same PWM frequency. Each LED output can be off or on (no PWM control) or set at its individual PWM controller value. The LED output driver is programmed to be either open-drain with a 25 mA current sink capability at 5 V or totem pole with a 25 mA sink, 10 mA source capability at 5 V. The PCA9685 operates with a supply voltage range of 2.3 V to 5.5 V and the inputs and outputs are 5.5 V tolerant. LEDs can be directly connected to the LED output (up to 25 mA, 5.5 V) or controlled with external drivers and a minimum amount of discrete components for larger current or higher voltage LEDs [4].

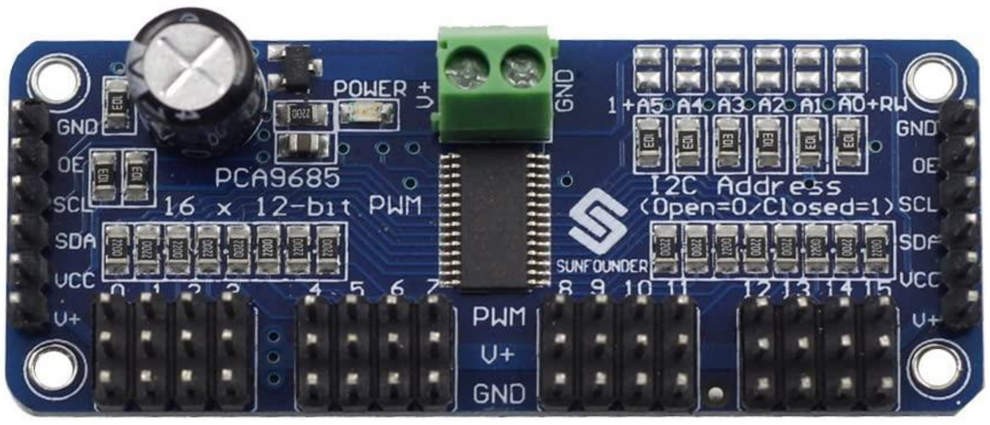


Fig 3. PCA 9685 [5]

C. I2C bus:

The I2C bus was designed by Philips in the early ’80s to allow easy communication between components which reside on the same circuit board. The name I2C translates into “Inter IC”. Sometimes the bus is called IIC or I²C bus [6]. I2C combines the best features of SPI and UARTs. With I2C, you can connect multiple slaves to a single master and you can have multiple masters controlling single, or multiple slaves. I2C only uses two wires to transmit data between devices:

- **SDA (Serial Data)** – The line for the master and slave to send and receive data.
- **SCL (Serial Clock)** – The line that carries the clock signal.

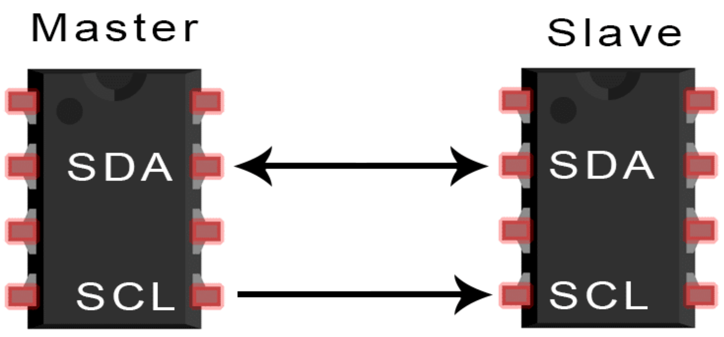


Fig 4. I2C bus [7]

With I2C, data is transferred in messages. Messages are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted. The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:

- **Start Condition:** The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.
- **Stop Condition:** The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.

- **Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
- **Read/Write Bit:** A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
- **ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device [7].

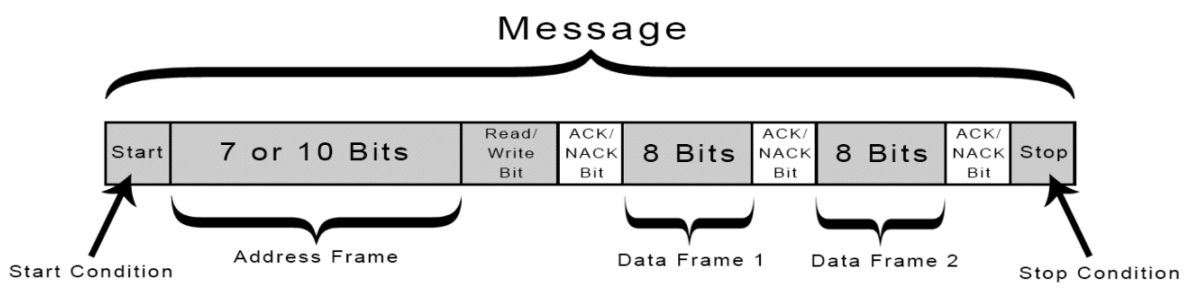
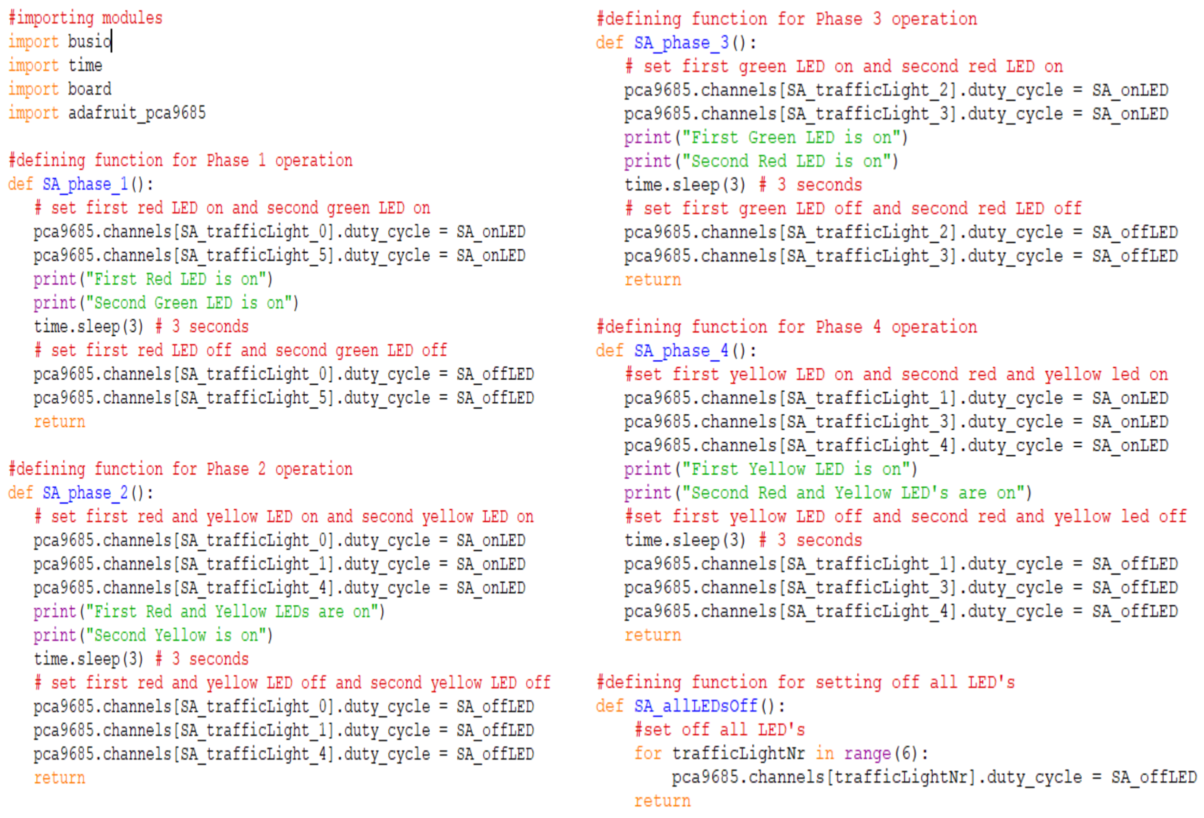


Fig 5. I2C working [7]

Software:

We have used IDLE Python 3.8.3 which is Integrated Development Environment provided by Python itself to develop the code.

The code is as follows:



```
#importing modules
import busio
import time
import board
import adafruit_pca9685

#defining function for Phase 1 operation
def SA_phase_1():
   # set first red LED on and second green LED on
   pca9685.channels[SA_trafficLight_0].duty_cycle = SA_onLED
   pca9685.channels[SA_trafficLight_5].duty_cycle = SA_onLED
   print("First Red LED is on")
   print("Second Green LED is on")
   time.sleep(3) # 3 seconds
   # set first red LED off and second green LED off
   pca9685.channels[SA_trafficLight_0].duty_cycle = SA_offLED
   pca9685.channels[SA_trafficLight_5].duty_cycle = SA_offLED
   return

#defining function for Phase 2 operation
def SA_phase_2():
   # set first red and yellow LED on and second yellow LED on
   pca9685.channels[SA_trafficLight_0].duty_cycle = SA_onLED
   pca9685.channels[SA_trafficLight_1].duty_cycle = SA_onLED
   pca9685.channels[SA_trafficLight_4].duty_cycle = SA_onLED
   print("First Red and Yellow LEDs are on")
   print("Second Yellow is on")
   time.sleep(3) # 3 seconds
   # set first red and yellow LED off and second yellow LED off
   pca9685.channels[SA_trafficLight_0].duty_cycle = SA_offLED
   pca9685.channels[SA_trafficLight_1].duty_cycle = SA_offLED
   pca9685.channels[SA_trafficLight_4].duty_cycle = SA_offLED
   return

#defining function for Phase 3 operation
def SA_phase_3():
   # set first green LED on and second red LED on
   pca9685.channels[SA_trafficLight_2].duty_cycle = SA_onLED
   pca9685.channels[SA_trafficLight_3].duty_cycle = SA_onLED
   print("First Green LED is on")
   print("Second Red LED is on")
   time.sleep(3) # 3 seconds
   # set first green LED off and second red LED off
   pca9685.channels[SA_trafficLight_2].duty_cycle = SA_offLED
   pca9685.channels[SA_trafficLight_3].duty_cycle = SA_offLED
   return

#defining function for Phase 4 operation
def SA_phase_4():
   #set first yellow LED on and second red and yellow led on
   pca9685.channels[SA_trafficLight_1].duty_cycle = SA_onLED
   pca9685.channels[SA_trafficLight_3].duty_cycle = SA_onLED
   pca9685.channels[SA_trafficLight_4].duty_cycle = SA_onLED
   print("First Yellow LED is on")
   print("Second Red and Yellow LED's are on")
   #set first yellow LED off and second red and yellow led off
   time.sleep(3) # 3 seconds
   pca9685.channels[SA_trafficLight_1].duty_cycle = SA_offLED
   pca9685.channels[SA_trafficLight_3].duty_cycle = SA_offLED
   pca9685.channels[SA_trafficLight_4].duty_cycle = SA_offLED
   return

#defining function for setting off all LED's
def SA_allLEDsOff():
    #set off all LED's
    for trafficLightNr in range(6):
        pca9685.channels[trafficLightNr].duty_cycle = SA_offLED
    return
```

```
# Create the I2C bus interface.
i2c_bus = busio.I2C(board.SCL, board.SDA)

# Create a simple PCA9685 class instance.
pca9685 = adafruit_pca9685.PCA9685(i2c_bus)

# Set the PWM frequency to 60 Hz.
pca9685.frequency =    60

#defining required variables
SA_onLED              = 65535 # PWM duty cycle 100%
SA_offLED             =     0 # PWM duty cycle   0%
SA_trafficLight_0     =     0 # First traffic light Red
SA_trafficLight_1     =     1 # First traffic light Yellow
SA_trafficLight_2     =     2 # First traffic light Green
SA_trafficLight_3     =     3 # Second traffic light Red
SA_trafficLight_4     =     4 # Second traffic light Yellow
SA_trafficLight_5     =     5 # Second traffic light Green

#calling function to terminate program and set off all LED's
SA_allLEDsOff()
print("To finish press Ctrl C")

#calling all 4 phases on an infinite loop
try:
   while True:
       print()
       SA_phase_1()
       SA_phase_2()
       SA_phase_3()
       SA_phase_4()
except:
    print("finish")
    SA_allLEDsOff()
```

IV. Hardware Connections

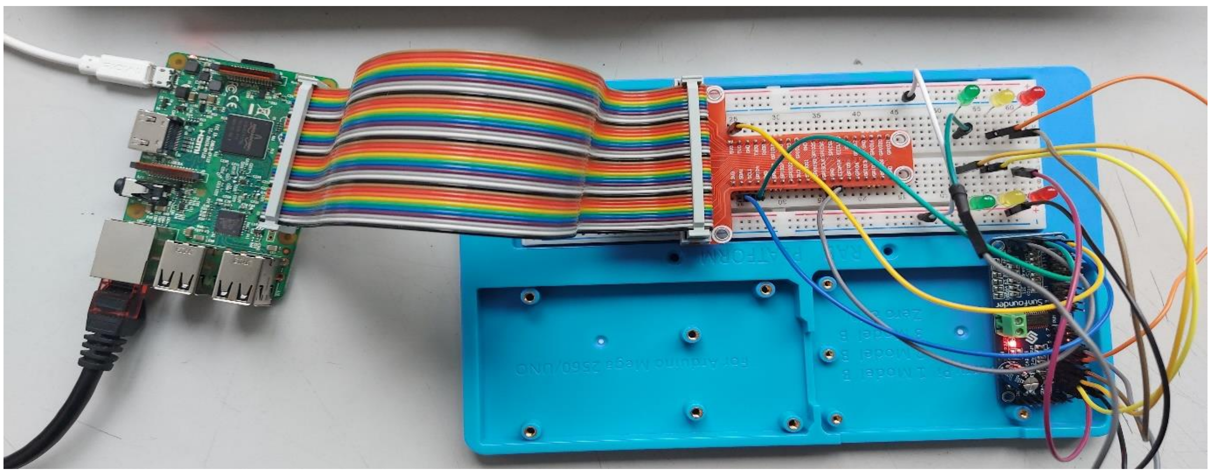


Fig. 6. Hardware Connection of Prototype

V. Result

The required 4 phases are realised as shown from Fig 7 to Fig 10. In Fig 7

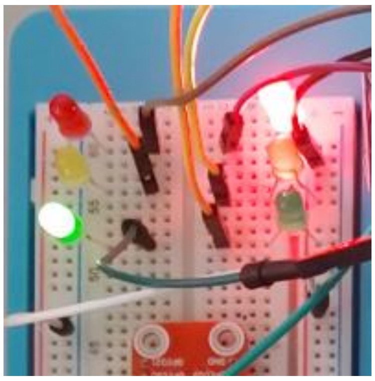


Fig 7. Phase 1

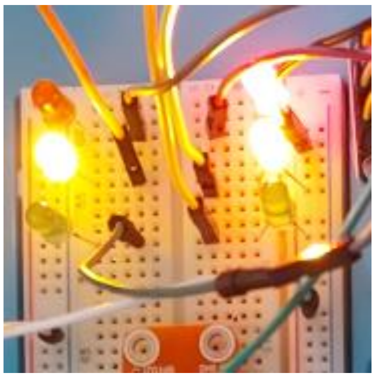


Fig 8. Phase 2

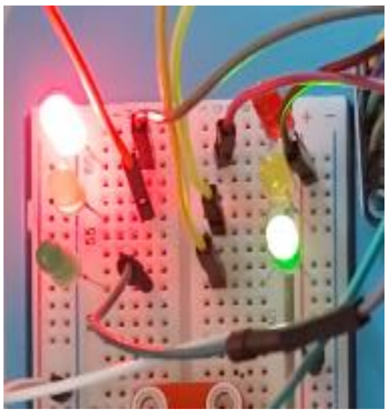


Fig 9. Phase 3

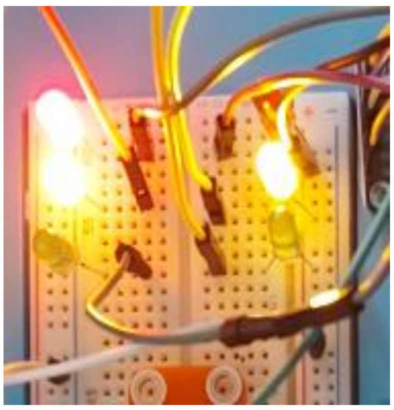


Fig 10. Phase 4

VI. Conclusion

This project successfully represents a system for managing 2 way traffic over a narrow lane. Raspberry PI 3 and PCA9685 acts as the suitable hardware for the implementation of this design as they come with all the required functionalities to run it and consist of the required number of ports and channels. Python proves to be an efficient language as it helps to define the working of all the LEDs separately, thus avoiding confusion and enabling the traffic light system to work successfully.

VII. References

1. https://en.wikipedia.org/wiki/Raspberry_Pi
2. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
3. https://de.farnell.com/raspberry-pi/raspberrypi3-modb-1gb/sbc-raspberry-pi-3-model-b-1gb/dp/2525226
4. https://www.nxp.com/products/power-management/lighting-driver-and-controller-ics/ic-led-controllers/16-channel-12-bit-pwm-fm-plus-ic-bus-led-controller:PCA9685
5. https://www.amazon.de/SunFounder-PCA9685-Channel-Arduino-Raspberry/dp/B014KTSMLA
6. https://www.i2c-bus.org/
7. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/