The electronics and the software implementation to traffic light system

Prepared By: **NIKHIL GOWDA SHIVASWAMY**
Matriculation number: 11013382
E-Mail**:** **nikhil.gowda1696@gmail.com**

Dept. of Information Technology
SRH University, Heidelberg, Germany

Under the guidance of Prof. Alfred Moos

Abstract – **With the technological advancement nowadays embedded has become a hot field for research. Embedded system is now used in almost all areas like security, transportation, medical equipment's, military equipment's etc. The major focus of this project is on the use of microcontroller I2C protocol and PCA9685 servo driver to implement a traffic light system. Here RPI GPIO2 and GPIO3 are used for I2C protocol with PCA9685 to control individual LED. With this feature we can control the level of dim and bright with PWM output. In this paper, the electronics and the software implementation to a traffic light system contains of two traffic lights that control the traffic on a narrow river bridge with only one line.**

***Keywords:** Raspberry Pi, LED, PWM Servo driver*

I. Aim

The project has been designed for developing a traffic system that control the traffic on a narrow river bridge with only one line. Basic objectives that we should consider while doing this project is noted as follows:

1) The traffic lights are controlled in four phases.
2) The two traffic lights must be realized with LEDs on the breadboard.
3) The LEDs must be controlled with the PCA9685 module in PWM.
4) The connection between the Raspberry Pi and the PCA9685 must be made through the I2C bus.

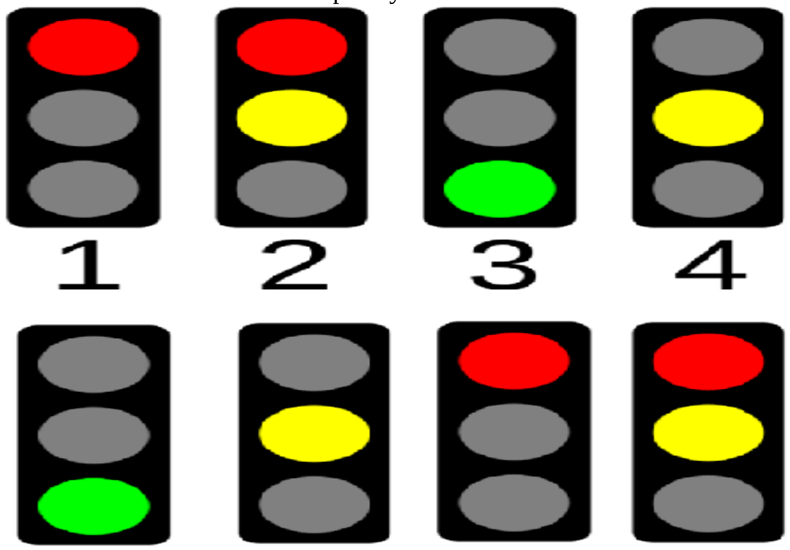


Fig 1. 4 phases of two traffic light

II. Introduction

Science has brought wonderful and amazing technologies to easy out the human life. Embedded System is one of the branches of it which has made human life easier and reduced the workload. It has also enabled us to reduce the participation of human in risky works. An embedded system, which mainly controls physical operations of the machine that it is embedded within the real time computation. Therefore, embedded applications are being used for various purpose in industries, labs, Space and in battlefield.

“An **embedded system** is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electrical system”.[1] It is dedicated to perform specific task by considering the size, cost, power consumption , reliability and performance. It ranges from portable device like kid toy and mp3/mp4 player, to large system like avionics and military tools.

This project’s main functionality is to ensure the co-ordination between two line on narrow river to move as smoothly, and safely as possible. This system is implementing using I2C protocol and the traffic lights are controlled by PWM signal which was driven by the RPI and PCA9686 servo driver. Using I2C protocol we can send data bits, read, or write bits with synchronous clock model. SDA and SCA carries data and clock, respectively. This whole module is controlled by RPI where software is written and dumped to control the traffic lights. The adjustment of brightness and dimness of LED is done via PCA9685.

III. PROJECT PLANNING

3.1 Stages

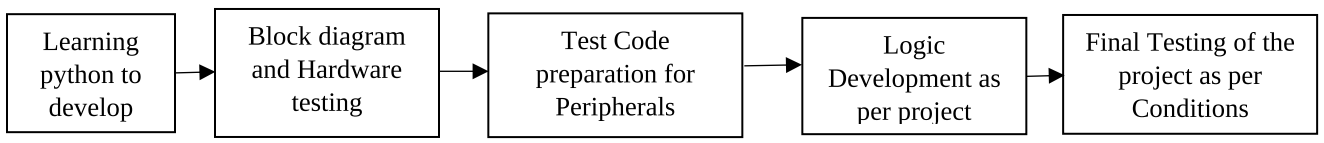

Learning python to develop
Block diagram and Hardware testing
Test Code preparation for Peripherals
Logic Development as per project
Final Testing of the project as per Conditions

Fig 2. Design and development flow

1) Learning Python:

   To develop a software for this project, Python coding knowledge is desirable. Hence, we learnt Python language effectively from the embedded system class.

2) Block diagram and Hardware testing:

   To understand the project and to implement properly we need proper block diagram which ease us to connect hardware properly. After collecting all electronic components, we need to test the working behavior of working.

3) Test code preparations for peripherals:

   To check the GPIO's working and LED's, we need to prepare test code and cross check the communication between the peripherals and GPIO.

4) Logic Development:

   After testing the components, we developed the software as per project requirements and this is written in Python.

5) Final testing:

   After dumping program into microcontroller, we need to verify with all possibilities outcomes.

3.2 Block diagram

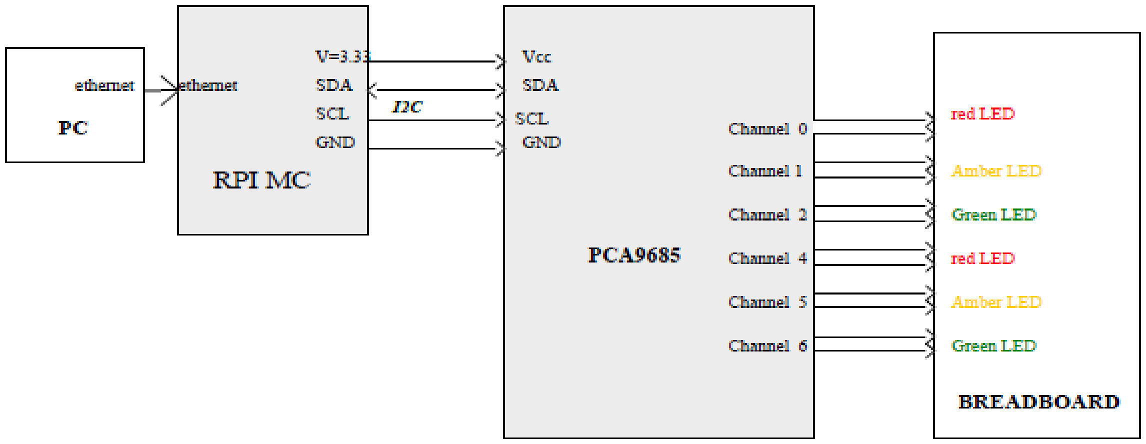


Fig 3. Block diagram of the entire system and subsystem

Here the block diagram representation of the entire traffic control system is depicted from where its operating procedure can be described. The first block represents the PC, and which is connected to Raspberry pi via ethernet. This ethernet connection allowing RPI to access the Remote Desktop. The remote block (RPI MC) represents the operator controlling section from where instructions are encoded and transmitted from Data signal which is received by the receiver PCA9685 SDA line and it is then decoded and delivered through the PWM and GND pin to the cathode and anode of LED respectively. This microcontroller is then programmed according to the instructor's demand

3.3 I2C Protocol

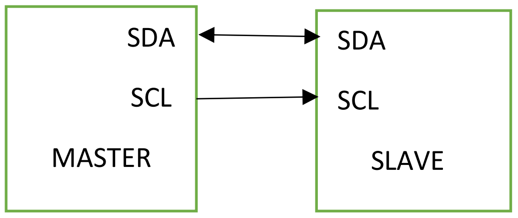


Fig 4. I2C Protocol overview

3.3.1 I2C Overview

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, sensors, A/D and D/A converters, I/O interfaces, and other similar peripherals in embedded systems. It was invented by Philips and now it is used by almost all major IC manufacturers. Each I2C slave device needs an address. I2C uses only two wires: SCL (serial clock) and SDA (serial data). Both need to be pulled up with a resistor.

3.3.2 I2C Operations

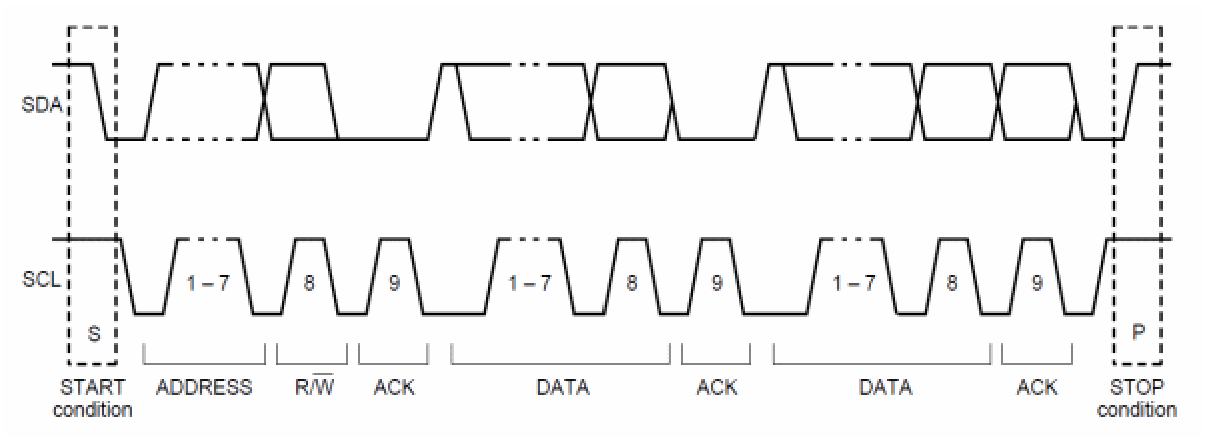


Fig 5. I2C Protocol operation [2]

In general, both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the Start condition (S) followed by the address of the slave device. This communication is using transfers of 8 bits or multiple bytes. Each I2C slave device has a 7-bit address that is unique. 7-bit address represents bits 7 to 1 while bit 1 is used to signal to read from and 0 to write to the device.
If 8th bit of address byte 0, the master device will write to the slave device. Or else, the next byte will read from the slave device. Once all bytes are read or written the master device gets the acknowledgment (ACK) and at the end it generates Stop condition (P).

IV Project Setup

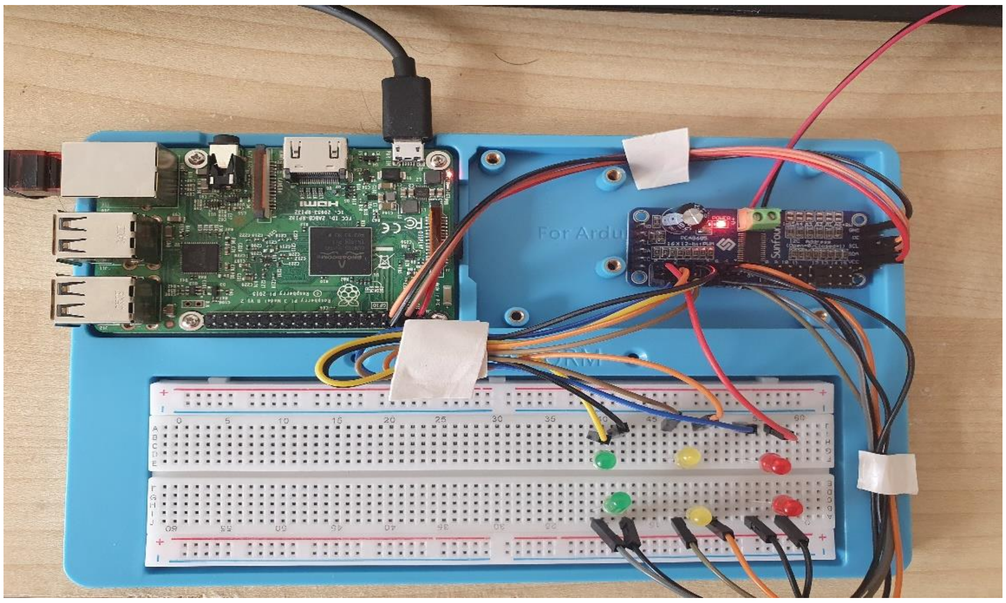


Fig 6. Project Setup

4.1 Project Setup and Procedure

4.1.1 Project Setup for RPI

Here Raspberry Pi 3b controller is used as master and configured with slave PCA9685 PWM servo driver. Both are communicated with I2C protocol. RPI's GPIO 2 and GPIO 3 acts as I2C's SDA and SCL respectively. We need to configure by installing the below two function in terminal window of RPI

1. `sudo apt-get install -y python-smbus` and
2. `sudo apt-get install -y i2c-tools`

Further, we just need to enable I2C at RPI configuration in preferences.

4.1.2 Project Setup for PCA9685

Here PCA9685 PWM servo driver is used as slave and take a command from master RPI. This controls the LED brightness and dimness via its PWM signal. It has control input pins for I2C on either side. Where SDA and SCL are configured with RPI to communicate synchronously. To enable this protocol with RPI, we need to configure by installing two adafruit libraries in terminal window of RPI

1. `pip3 install adafruit-circuitpython-pca9685`
2. `sudo pip3 install adafruit-circuitpython-servokit`

4.1.3 PIN and LED connection

After setup all above modules, we need to build a connection with RPI and PCA9685 by connecting with female to female jumper wires.

Connections are shown as below:

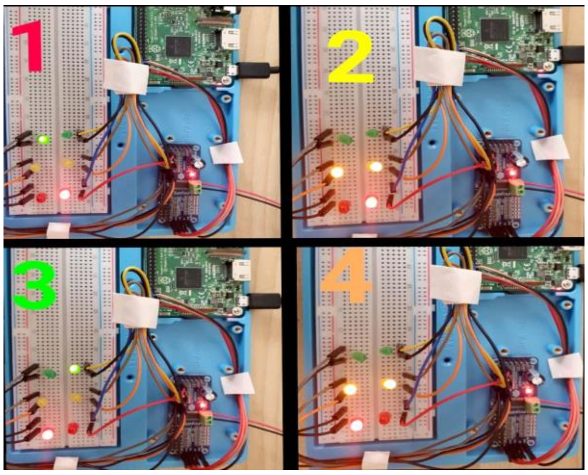
RPI → PCA9685

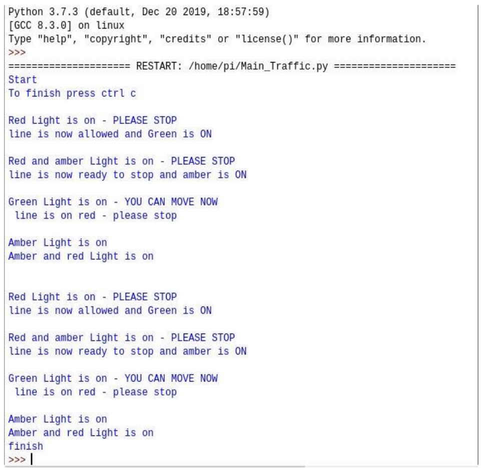
GPIO 2 (SDA)	→	SDA
GPIO 3 (SCL)	→	SCL
3V3 Power	→	VCC
GND	→	GND

Further, we need to connect the LED as per the project requirements on the bread board and it should control by the PWM control output signal. PCA 9685 has 16 control output pins. Here, we are using 6 control outputs to the LED. These connections are shown as below:

PCA9685	→	LED
Channel 0	→	Red
Channel 1	→	Amber
Channel 2	→	Green
Channel 4	→	Red
Channel 5	→	Amber
Channel 6	→	Green

V Project working and Result





```
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==================== RESTART: /home/pi/Main_Traffic.py ====================
Start
To finish press ctrl c

Red Light is on - PLEASE STOP
line is now allowed and Green is ON

Red and amber Light is on - PLEASE STOP
line is now ready to stop and amber is ON

Green Light is on - YOU CAN MOVE NOW
 line is on red - please stop

Amber Light is on
Amber and red Light is on


Red Light is on - PLEASE STOP
line is now allowed and Green is ON

Red and amber Light is on - PLEASE STOP
line is now ready to stop and amber is ON

Green Light is on - YOU CAN MOVE NOW
 line is on red - please stop

Amber Light is on
Amber and red Light is on
finish
>>> |
```

Fig 7. Project Final result and all 4 phases images

5.1.1 Experimental procedure

As the above shown fig(7), all 4 phases are implemented and controlled by PWM signal. At narrow bridge above the river, one side should stop and another side should move concurrently. This is controlled by writing concurrent program in python and flashing into RPI after above project setup. The results of the project explained in 4 phases are as shown below:

Phase 1: One side, vehicles should stop so Red LED is On and another side Vehicles are allow to move hence Green is ON.

Phase 2: One side, vehicles should get ready to move but should not move till the bridge get clear so Red and Amber LED is On and another side, should stop their vehicle hence Amber is ON.

Phase 3: One side, vehicles allowed to move so Green LED is On and another side, vehicles should stop so Red LED is On.

Phase 4: One side, should stop their vehicle hence Amber is ON and another side, vehicles should get ready to move but should not move till the bridge get clear so Red and Amber LED.

These phases are in loop and repeat continuously with given delay and it can be stop by pressing control + C

5.1.2 Results

All four phases implemented successfully and got a proper expected outcome without any error.

VI Software Design of the System

I have used Python IDLE 3 which is Integrated Development Environment provided by the Python.org company itself for developing software. It also supports open-source libraries and no extra piece of hardware is required for flashing.

Code to control the Traffic:

```
######  NAME : Nikhil Gowda Shivaswamy
##      Matricualtion number: 11013382
## File Name: Main_Traffic.py

##Description: Embedded main project
## Designed for developing a traffic system
##I2C is confiqured

##Board: Rasberry pi 3b

## I2C confiqured with: PCA9685

## IMPORTANT NOTE:All identifiers of variables are started with NG
## NG: Nikhil Gowda is an identical identifier which i used to write
#python software.

    # Main program starts from here for Two traffic lights
    # on the narrow line of river bridge

from board import  SCL , SDA
import busio
import time
from adafruit_pca9685 import PCA9685

  # FUNCTION DEFINITIONS FOR ALL 4 PHASES OF TRAFFIC SYSTEM
      # set RED is on
     #opposite side should move so Green LED is ON
def NG_phase_1():
    NG_pca9685.channels[NG_trafficLight_0].duty_cycle = NG_onLED
    NG_pca9685.channels[NG_trafficLight_OPP_2].duty_cycle = NG_onLED
    print("Red Light is on - PLEASE STOP")
    print("line is now allowed and Green is ON")
    time.sleep(3)
    NG_pca9685.channels[NG_trafficLight_0 ].duty_cycle = NG_offLED
    NG_pca9685.channels[NG_trafficLight_OPP_2].duty_cycle = NG_offLED
    return

      # set red and amber is on
     #opposite side should get ready to
      # -- to stop the vehicle so amber LED is ON

def NG_phase_2():
    NG_pca9685.channels[NG_trafficLight_0].duty_cycle = NG_onLED
    NG_pca9685.channels[NG_trafficLight_1].duty_cycle = NG_onLED
    NG_pca9685.channels[NG_trafficLight_OPP_1].duty_cycle = NG_onLED
    print("Red and amber Light is on - PLEASE STOP")
    print("line is now ready to stop and amber is ON")
    time.sleep(3)
    NG_pca9685.channels[NG_trafficLight_0].duty_cycle = NG_offLED
    NG_pca9685.channels[NG_trafficLight_1].duty_cycle = NG_offLED
    NG_pca9685.channels[NG_trafficLight_OPP_1].duty_cycle = NG_offLED
    return

     # set Green is on
     #opposite side should stop red LED is ON


def NG_phase_3():
    NG_pca9685.channels[NG_trafficLight_2].duty_cycle = NG_onLED
    NG_pca9685.channels[NG_trafficLight_OPP_0].duty_cycle = NG_onLED
    print("Green Light is on - YOU CAN MOVE NOW")
    print(" line is on red - please stop ")
    time.sleep(3)
    NG_pca9685.channels[NG_trafficLight_2].duty_cycle = NG_offLED
    NG_pca9685.channels[NG_trafficLight_OPP_0].duty_cycle = NG_offLED
    return

     # set amber is on
     #opposite side should break and stop amber and red LED is ON

def NG_phase_4():
    NG_pca9685.channels[NG_trafficLight_1].duty_cycle = NG_onLED
    NG_pca9685.channels[NG_trafficLight_OPP_0].duty_cycle = NG_onLED
    NG_pca9685.channels[NG_trafficLight_OPP_1].duty_cycle = NG_onLED
    print("Amber Light is on")
    print("Amber and red Light is on")
    time.sleep(3)
    NG_pca9685.channels[NG_trafficLight_1].duty_cycle = NG_offLED
    NG_pca9685.channels[NG_trafficLight_OPP_0].duty_cycle = NG_offLED
    NG_pca9685.channels[NG_trafficLight_OPP_1].duty_cycle = NG_offLED
    return

     # set the all LEDs off

def NG_allLEDsoff():
    for NG_trafficLightNr in range (10):
        NG_pca9685.channels[NG_trafficLightNr].duty_cycle = NG_offLED
    return

   # Creating the I2C bus interface from 'busio and board'.

NG_i2c_bus  = busio.I2C(SCL,SDA)
```

```
  # Creating a simple PCA9685 class instance.

NG_pca9685  = PCA9685(NG_i2c_bus)


#set frequency for pwm
NG_pca9685.frequency = 50  #hz
NG_onLED              = 65535  # PWM duty cycle 100%(full bright)
NG_offLED             =     0   # PWM duty cycle 0%(complete dim)
  ##Creating channels for all led
NG_trafficLight_0      =    0  # Red LED
NG_trafficLight_1      =    1  # AMBER LED
NG_trafficLight_2      =    2  #Green LED
NG_trafficLight_OPP_0  =    4  #Red LED at another side(opposite way)
NG_trafficLight_OPP_1  =    5  #Amber LED at another side(opposite way)
NG_trafficLight_OPP_2  =    6  #Green LED at another side(opposite way)

print ("Start")
NG_allLEDsoff()
print("To finish press ctrl c")
try:
    while True:

        print()
        ## calling function as per the requidred phases
        NG_phase_1()
        print()
        NG_phase_2()
        print()
        NG_phase_3()
        print()
        NG_phase_4()

except:
    NG_allLEDsoff()
    print("pressed ctrl c")
    print("finish")
```

VII Conclusion

This project brief about the working of I2C protocol and PCA9685. Here, we successfully able to learn and implemented all four phases which we need to show in LED. Also, the brightness of LED is controlled by PWM signal.
Conventionally, LED lights are directly implemented through MC GPIO but here we control the brightness of LED and we can save power as well. Moreover, LED's s direct connection to GPIO is risky hence, we need resistance for every LED to avoid the damage of MC.

VIII Appendix

7.1 Hardware description of RPI

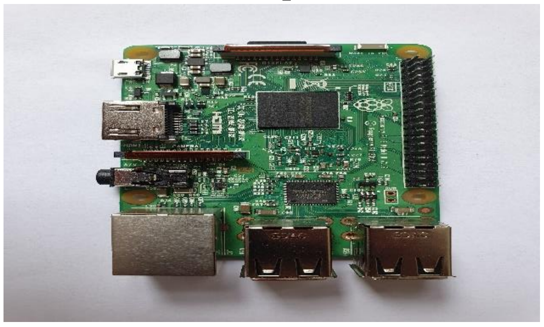


Fig 8. Raspberry pi 3b

Raspberry Pi 3 with a 1.2 GHz 64-bit ARM cortex quad core processor, on-board 802.11n Wi-Fi, Bluetooth and 4 USB ports. This has Base T ethernet socket, 3.5mm audio jack and Video output HDMI port. It is 40 pin Microcontroller. Among 40 pin, 26 pin acts as input/output pin(I/O). It has communication interface (UART) with GPIO15 and GPIO14. It also has 2 SPI, where SPI-1 interface with GPIO10, GPIO, GPIO8, GPIO11 and SPI-2 interface with GPIO20, GPIO19, GPIO21, GPIO17. Also, it has I2C interface peripherals GPIO2 and GPIO3.

7.2 Hardware description of PCA9685

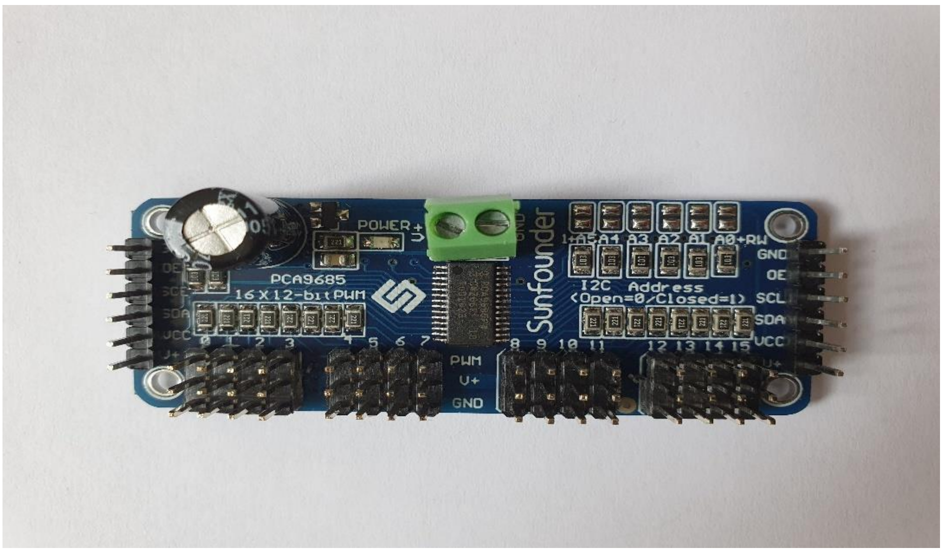


Fig 9. PCA9685 Servo controller

There are two sets of control input pins on either side which as SDA, SCL, VCC, GND and OE. We can use whichever side we like since both sides are identical. we can also easily chain by connecting two side-by-side. There are 16 output ports. Each port has 3 pins: PWM, V+ and GND. PWM is for output e.g. LED or motor. Each PWM runs completely independently but they must all have the same PWM frequency.

IX REFERENCES

[1] *https://en.wikipedia.org/wiki/Embedded_system*

[2] *Image source: http://fastbitlab.com/stm32-i2c-lecture-5-i2c-ack-and-nack-and-i2c-data-validity/*

[3] *Most of the source and knowledge gain from Prof. Alfred Moos ppt slides.*