Extra Credit Project

RGB LED Display Panel

Victor Mendoza

ID: 6348908

EML480 Introduction to Mechatronics

Date of Submission: 05-12-2025

Fall, 2025

Introduction:

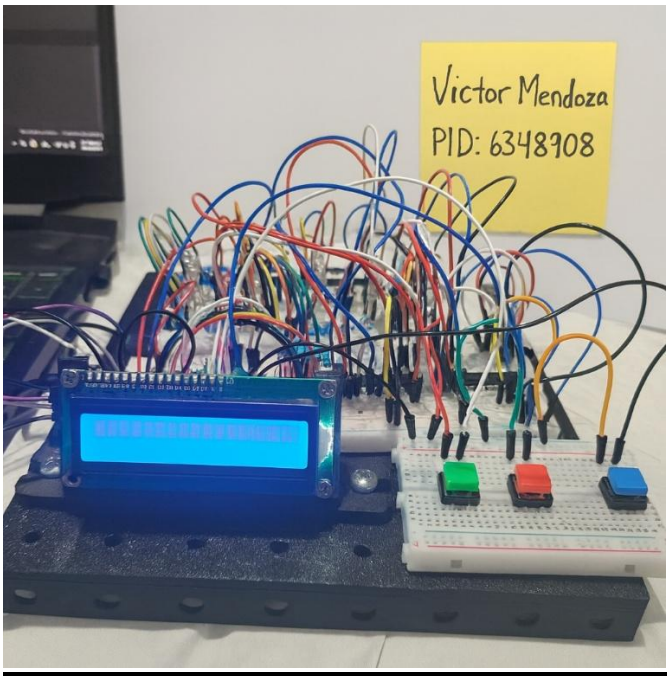
The RGB LEDs Display Panel project showcases the design, integration, and programming of a multifunctional embedded electronic system that combines digital logic expansion, LED color control, human–machine interfacing, and microcontroller-based automation. This project demonstrates engineering principles across electronics, microcontroller architecture, circuit design, and software-hardware synchronization. The system was developed using a Raspberry Pi Pico W as the central processing unit, two MCP23017 I/O expanders for digital output scaling, a 1602 I²C alphanumeric LCD for visual user feedback, and an array of RGB LEDs arranged in a structured matrix. The device integrates multiple subsystems into a cohesive unit capable of executing precise illumination sequences and user-driven interactions.

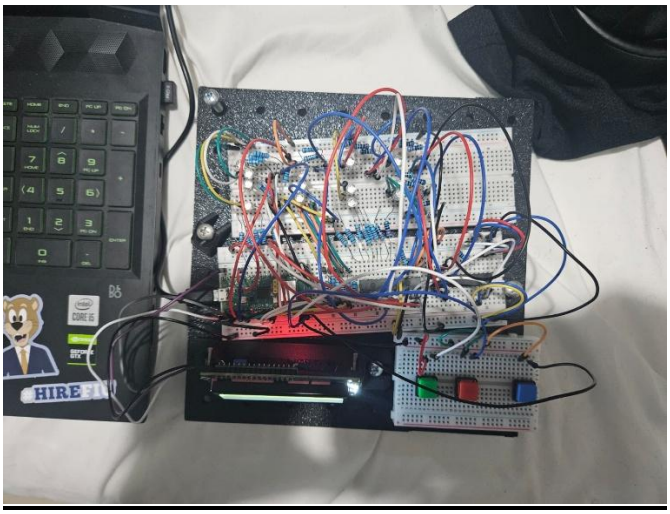
To accommodate the large number of output channels required for the LED matrix, the Raspberry Pi Pico is paired with dual MCP23017 chips connected over the I²C bus. Each MCP23017 provides 16 individually addressable pins, significantly increasing the number of digital outputs available to the controller. A custom Python class (MCP23017Pin) was implemented to manage these expanders efficiently through shadow-register logic, preventing unintentional overwriting of bit states and enabling high-speed, synchronized LED control. This modular approach to pin management ensures scalable expansion, consistent timing, and reliable performance across all output nodes.

Materials used:

1. PICO board
2. Breadboard
3. 13x RGB LEDs
4. 39x 220 Ohm resistor
5. 2x 10k Ohm resistor
6. 2x mcp23017
7. LCD Display Panel
8. Wires
9. 3x Mechanical Buttons

Picture:

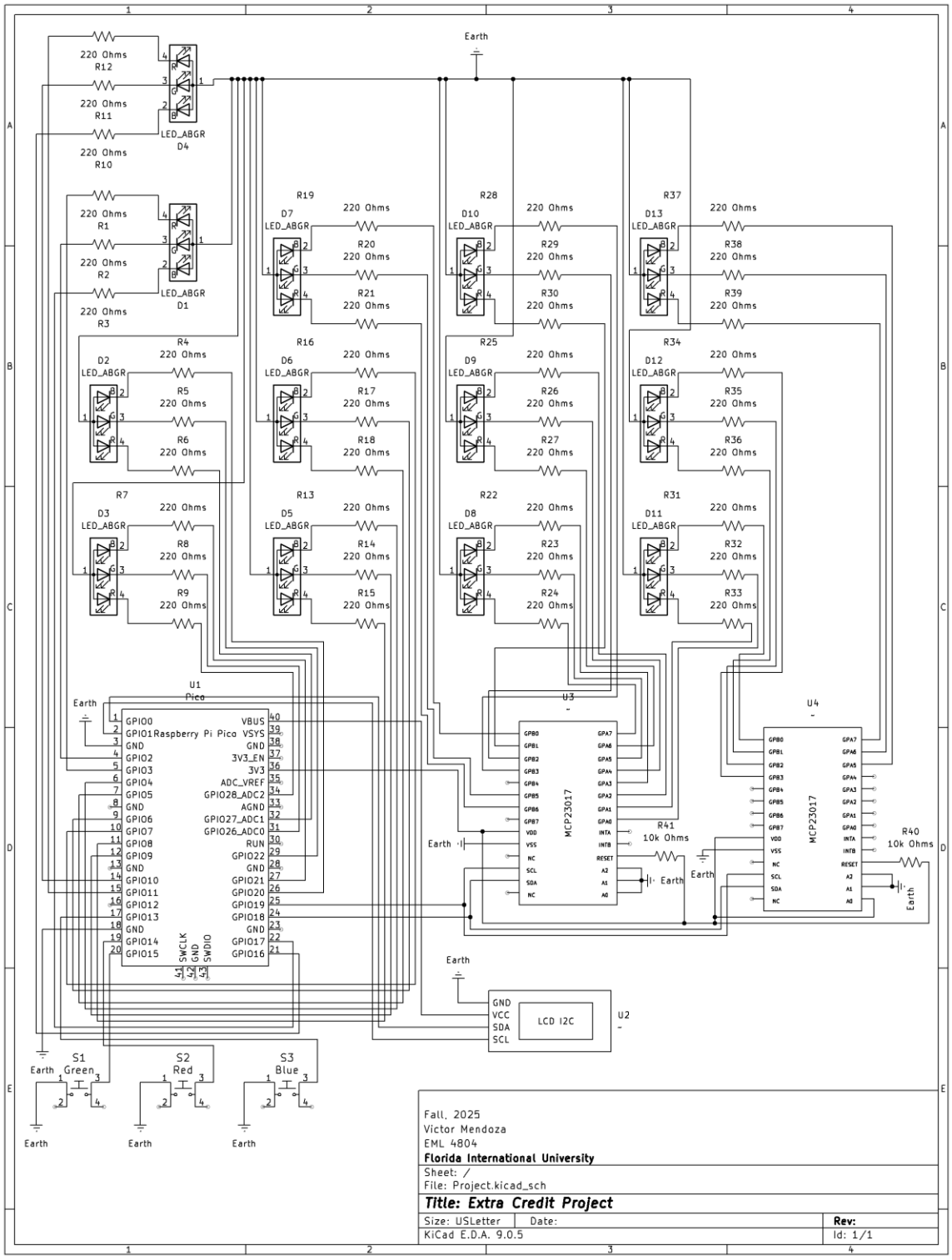




Video link:

https://youtu.be/MGPuzDlBf44

Diagram:


Earth
220 Ohms
R12
R11
R10
LED_ABGR
D4
R1
R2
R3
D1
R19
R20
R21
D7
R28
R29
R30
D10
R37
R38
R39
D13
R4
R5
R6
D2
R16
R17
R18
D6
R25
R26
R27
D9
R34
R35
R36
D12
R7
R8
R9
D3
R13
R14
R15
D5
R22
R23
R24
D8
R31
R32
R33
D11
U1
Pico
Raspberry Pi Pico
GPIO0
GPIO1
GND
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
GPIO8
GPIO9
GPIO10
GPIO11
GPIO12
GPIO13
GPIO14
GPIO15
VBUS
VSYS
3V3_EN
3V3
ADC_VREF
GPIO28_ADC2
AGND
GPIO27_ADC1
GPIO26_ADC0
RUN
GPIO22
GPIO21
GPIO20
GPIO19
GPIO18
GPIO17
GPIO16
SWCLK
SWDIO
U3
MCP23017
U4
R41
10k Ohms
R40
U2
LCD I2C
GND
VCC
SDA
SCL
S1
Green
S2
Red
S3
Blue
Fall, 2025
Victor Mendoza
EML 4804
Florida International University
Sheet: /
File: Project.kicad_sch
Title: Extra Credit Project
Size: USLetter
Date:
Rev:
KiCad E.D.A. 9.0.5
Id: 1/1

Code:

```
from machine import Pin, I2C
import time
import utime
from lcd1602 import LCD
from picozero import Button

#---------- MCP23017 SETUP ----------
i2c = I2C(1, scl=Pin(19), sda=Pin(18), freq=400000)  # GP1=SCL, GP0=SDA

MCP_ADDR1 = 0x20   # first chip (A0=A1=A2=GND)
MCP_ADDR2 = 0x21   # second chip (A0=VCC, A1=A2=GND)

IODIRA = 0x00
IODIRB = 0x01
GPIOA  = 0x12
GPIOB  = 0x13

# Shadow state for each port on each chip
_mcp_state = {
    MCP_ADDR1: {'A': 0x00, 'B': 0x00},
    MCP_ADDR2: {'A': 0x00, 'B': 0x00},
}

def init_mcp(addr):
    # All A/B pins as outputs
    i2c.writeto_mem(addr, IODIRA, b'\x00')
    i2c.writeto_mem(addr, IODIRB, b'\x00')
    # Initialize outputs to 0
    i2c.writeto_mem(addr, GPIOA, bytes([_mcp_state[addr]['A']]))
    i2c.writeto_mem(addr, GPIOB, bytes([_mcp_state[addr]['B']]))

# init both chips
init_mcp(MCP_ADDR1)
init_mcp(MCP_ADDR2)

#Defines MCP
class MCP23017Pin:
    def __init__(self, port, bit, addr=MCP_ADDR1):
        # port: 'A' or 'B'; bit: 0–7; addr: 0x20, 0x21, ...
```

```
        self.port = port
        self.bit = bit
        self.addr = addr

    def _write_back(self):
        reg = GPIOA if self.port == 'A' else GPIOB
        i2c.writeto_mem(self.addr, reg, bytes([_mcp_state[self.addr][self.port]]))

    def on(self):
        _mcp_state[self.addr][self.port] |= (1 << self.bit)
        self._write_back()

    def off(self):
        _mcp_state[self.addr][self.port] &= ~(1 << self.bit)
        self._write_back()

    def toggle(self):
        _mcp_state[self.addr][self.port] ^= (1 << self.bit)
        self._write_back()

 #---------- LED Setup ----------
pin = [28,26,21,27,22,20,3,2,17,11,10,16,8,9,4,5,6,7]
led= []
for i in range(18):
    led.append(machine.Pin(pin[i], machine.Pin.OUT))
#MCP23017
led.append(MCP23017Pin('B', 6, addr=MCP_ADDR1))
led.append(MCP23017Pin('B', 5, addr=MCP_ADDR1))
led.append(MCP23017Pin('B', 0, addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 7,addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 5,addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 3, addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 6,addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 4,addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 2,addr=MCP_ADDR1))
led.append(MCP23017Pin('B', 1,addr=MCP_ADDR1))
led.append(MCP23017Pin('B', 2,addr=MCP_ADDR1))
led.append(MCP23017Pin('B', 3,addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 1,addr=MCP_ADDR1))
led.append(MCP23017Pin('A', 0,addr=MCP_ADDR1))
```

```
led.append(MCP23017Pin('B', 0,addr=MCP_ADDR2))
led.append(MCP23017Pin('B', 1,addr=MCP_ADDR2))
led.append(MCP23017Pin('B', 2,addr=MCP_ADDR2))
led.append(MCP23017Pin('B', 3,addr=MCP_ADDR2))
led.append(MCP23017Pin('A', 7, addr=MCP_ADDR2))
led.append(MCP23017Pin('A', 6, addr=MCP_ADDR2))
led.append(MCP23017Pin('A', 5, addr=MCP_ADDR2))

RGB = [
   #ROW 1
   (0, 1, 2),  # LED 1
   (3, 4, 5),  # LED 2
   (6, 7, 8),  # LED 3
   (9, 10, 11),  # LED 4

   #ROW 2
   (12, 13, 14),  # LED 5
   (15,16, 17),  # LED 6
   (18, 19, 20),  # LED 7

   #ROW 3
   (21, 22, 23),  # LED 8
   (24, 25, 26),  # LED 9
   (27, 28, 29),  # LED 10

   #ROW 4
   (30, 31, 32),  # LED 11
   (33, 34, 35),  # LED 12
   (36, 37, 38),  # LED 13
]


#Precision = 1 = Yellow = (1, 1, 0)
#Domination = 2 = Red = (1, 0, 0)
#Sorcery = 3 = Purple =  (1, 0, 1)
#Resolve  = 4 = Green = (0, 1, 0)
#Inspiration = 5 = Blue = (0, 0, 1)


colors = {
```

```
    1: (1.0, 1.0, 0.0),   # Yellow = Precision
    2: (1.0, 0.0, 0.0),   # Red = Domination
    3: (1, 0.0, 1),   # Purple = Sorcery
    4: (0.0, 1.0, 0.0),   # Green = Resolve
    5: (0.0, 0.0, 1.0)    # Blue = Inspiration
}

#LCD Setup
lcd = LCD()
lcd.clear()

# Buttons wired to GND
butr = Button(13)   # red
butg = Button(14)   # green
butb = Button(15)   # blue

#LIBRARY--------------------------------------------------------------------------------
top = [
    (1, 4, 2, 2, 2, 4, 3, 6, "Camille", "Conq", "Res"),
    (1, 4, 2, 1, 3, 4, 3, 6, "Yasuo", "LT", "Res"),
    (1, 4, 2, 1, 3, 4, 6, 7, "Jax", "LT", "Res"),
    (5, 3, 3, 1, 3, 3, 3, 6, "Kaisa", "FS", "Sorc")
]

jg = [
    (1, 4, 2, 1, 1, 5, 2, 7, "Lee Sin", "Conq", "Insp"),
    (2, 1, 3, 1, 3, 3, 1, 4, "Zed", "Elec", "Sorc"),
    (1, 1, 2, 1, 1, 5, 2, 7, "Elise", "Pta", "Insp")
]

mid = [
    (4, 1, 2, 1, 1, 4, 6, 7, "Akshan", "PTA", "Res"),
    (3, 2, 2, 2, 1, 4, 6, 9, "Twisted Fate", "Comet", "Res"),
    (1, 4, 2, 1, 3, 4, 3, 5, "Yasuo", "LT", "Res"),
    (1, 4, 2, 1, 3, 4, 3, 5, "Yone", "LT", "Res")
]

adc = [
    (1, 3, 3, 3, 2, 3, 5, 9, "Jhin", "Fleet", "Sorc"),
    (1, 2, 3, 3, 2, 5, 3, 9, "Ezreal", "LT", "Insp"),
```

```
    (1, 2, 3, 3, 2, 5, 6, 8, "Ashe", "LT", "Insp"),
    (1, 1, 3, 1, 2, 5, 2, 6, "Lucian", "PTA", "Insp"),
    (3, 2, 2, 1, 1, 5, 3, 4, "Varus", "Comet", "Insp")
]

supp = [
    (4, 3, 1, 3, 3, 2, 6, 9, "Rakan", "Guardian", "Dom"),
    (3, 1, 2, 1, 1, 3, 6, 8, "Milio", "Aery", "Res"),
    (2, 3, 1, 3, 2, 3, 3, 5, "Pyke", "HoB", "Sorc")
]
#LIBRARY-----------------------------------------------------------------------------

#Determines Color
def mainr(champ):
    return colors.get(champ[0])

def secr(champ):
    return colors.get(champ[5])

#Determine Leds to Toggle
#R = led groups (RGB)
def ledID(champ, R):
    return R[champ[1] - 1], R[champ[2] + 4 - 1], R[champ[3] + 7 - 1], R[champ[4] + 10 - 1]

def secID(champ, R):
    return R[champ[6] + 4 -1], R[champ[7] + 4 -1]

#Sets the color
#R = led groups (RGB)
#l = led numbers
def setcolor(R, C, l):
    for f in range(4):      # rows
        for i in range(3):  # RGB
            if C[i] == 1:
                R[l[f][i]].toggle()

def secolor(R, C, l):
    for f in range(2):      # rows
        for i in range(3):  # RGB
            if C[i] == 1:
```

```
            R[l[f][i]].toggle()

#Picks the Runes
def runes(champ):
    w = secID(champ, RGB)
    x = ledID(champ, RGB)
    y = mainr(champ)
    z = secr(champ)
    setcolor(led, y, x)
    utime.sleep(1)
    [l.off() for l in led]
    secolor(led, z, w)
    utime.sleep(1)
    [l.off() for l in led]

def write(champ):
    a = champ[8]
    b = champ[9]
    c = champ[10]
    d = champ[9] + " - " + champ[10]
    lcd.write(0, 0, a)
    lcd.write(0, 1, d)

#write(jg[0])
#runes(jg[0])

a = "Hold Red for \nPrevious"
b = "Hold Green for \nNext"
c = "Hold Blue to \nChange Lane"
lcd.clear()
lcd.message(a)
utime.sleep(2)
lcd.clear()
lcd.message(b)
utime.sleep(2)
lcd.clear()
lcd.message(c)
utime.sleep(2)
lcd.clear()
```

```
t = 1
l = "LET GO"

lanes = {
    1: ("TOP", top),
    2: ("JG", jg),
    3: ("MID", mid),
    4: ("ADC", adc),
    5: ("SUPP", supp),
}

while True:
    current_t = t
    name, lib = lanes[current_t]
    lcd.clear()
    lcd.message(name)
    time.sleep(2)
    number = 0
    max_index = len(lib) - 1

    while t == current_t:
        lcd.clear()
        write(lib[number])
        check = number

        while number == check and t == current_t:
            runes(lib[number])

            if butr.is_pressed:
                number -= 1
                if number < 0:
                    number = max_index
                lcd.clear()
                lcd.message(l)
                while butr.is_pressed:
                    print(l)

            if butg.is_pressed:
                number += 1
                if number > max_index:
```

```
            number = 0
        lcd.clear()
        lcd.message(l)
        while butg.is_pressed:
            print(l)

    if butb.is_pressed:
        t += 1
        if t > 5:
            t = 1
        print(t)
        lcd.clear()
        lcd.message(l)
        while butb.is_pressed:
            print(l)
        break
```

Conclusions:

The RGB LEDs Display Panel demonstrates how a complex hardware concept can be transformed into a fully functional embedded prototype through careful planning, systematic wiring, and structured programming. The final system operates reliably, responds to user input in real time, and executes coordinated lighting sequences across a large array of LEDs, confirming the effectiveness of both the electrical design and the firmware logic. Building the project required troubleshooting, iterative testing, and integration of multiple components, ultimately resulting in a robust platform that can be expanded or adapted for future engineering applications. This work highlights the practical value of embedded design principles and serves as a solid foundation for more advanced control systems or custom electronic interfaces.

References:

- Raspberri Pi Pico Library: https://github.com/ncarandini/KiCad-RP-Pico/tree/main
- How to use a breadboard https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all
- Raspberry Pi Pico usage https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html
- LCD wiring instructions https://docs.sunfounder.com/projects/thales-kit/en/latest/micropython/liquid_crystal_display.html

- Using I/O with MicroPython on the Pi Pico https://www.upesy.com/blogs/tutorials/micropython-raspberry-pi-pico-gpio-pins-usage#google_vignette
- Microchip Technology Inc. (2007). *MCP23017/MCP23S17: 16-Bit I/O Expander with Serial Interface* (DS21952B). Microchip Technology.