Dr. Vishwanath Karad MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

# Digital Converter using Pi_Pico_W

## Abstract

A Raspberry Pi Pico W is transformed into a versatile data converter, offering real-time conversion between multiple number systems with display options on a 1602 LCD, LEDs, and a buzzer. The project leverages the Pi Pico's dual cores and asynchronous programming, demonstrating proficiency in computer architecture. Remote control via web interface adds another layer of functionality. This project merges programming and hardware to tackle a practical challenge of digital data conversion, showcasing the application of computer architecture concepts in real-world problem solving.

## Introduction

This project addresses the need for a handy and versatile tool to perform real-time conversions between various number systems. By combining hardware and software components, it provides a practical learning experience for computer architecture concepts and their real-world applications

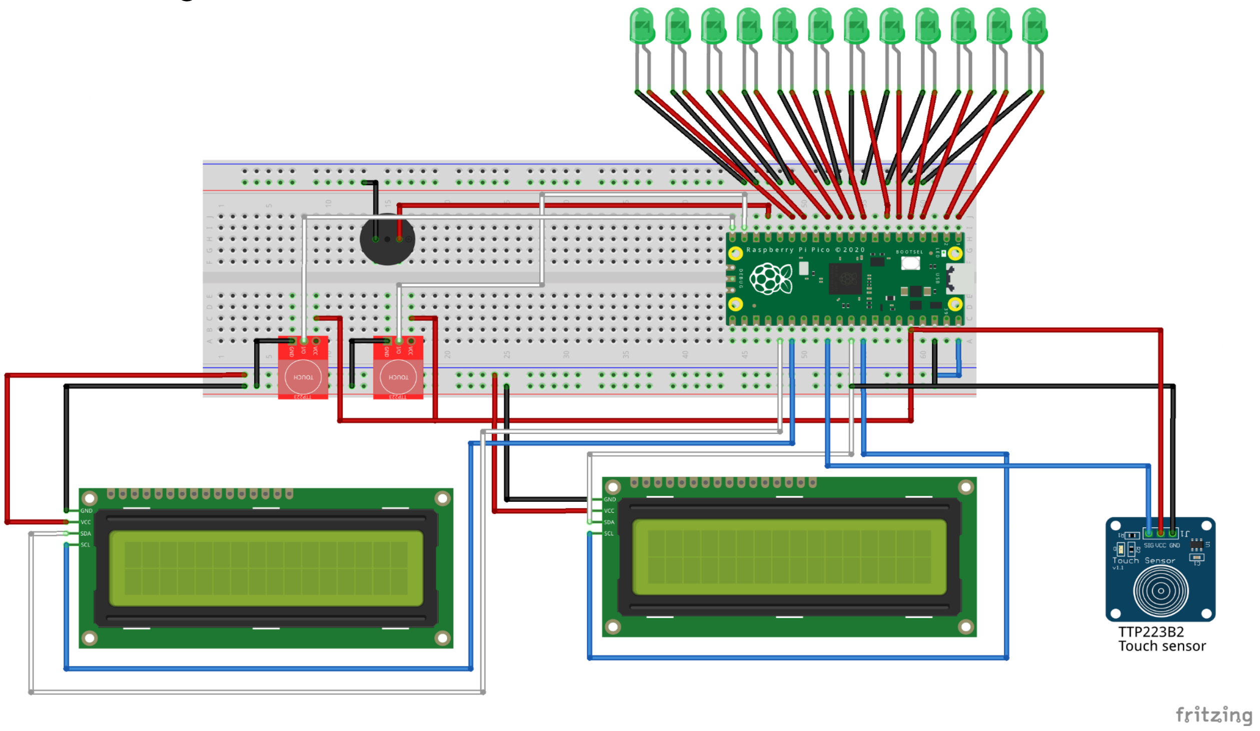
## Importance

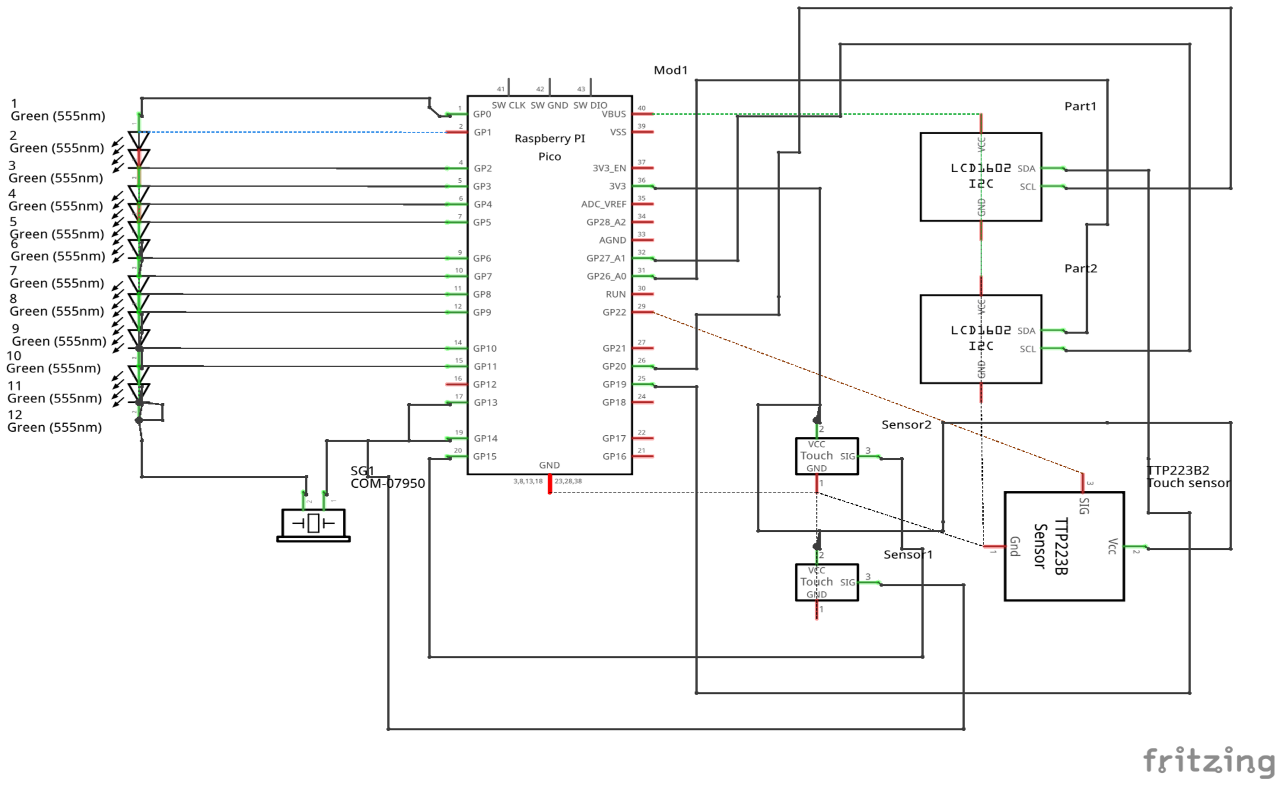
1. **Solidifying Foundational Knowledge**: It reinforces understanding of various number systems, crucial for computer architecture. Hands-on conversion solidifies theoretical knowledge.
2. **Showcasing Multi-Core and Asynchronous Programming**: Utilizes both Pi Pico W cores and asynchronous programming, demonstrating practical application of computer architecture principles for modern multi-core systems.
3. **Enhancing Learning Through Visualization**: Diverse output options (display, LEDs, buzzer) cater to different learning styles and create a more interactive experience.

## Components Used:

| | | |
|---|---|---|
| 1. | Raspberry Pi Pico W | X 1 |
| 2. | 1602 LCD Display | X 2 |
| 3. | Passive Buzzer | X 1 |
| 4. | Green LED’s | X 12 |
| 5. | Touch Sensor | X 2 |
| 6. | Capacitive Touch Sensor | X 1 |
| 7. | Jumper Wires | |

## Block Diagram





## Methodology: Iterative Development with Git Version Control

This project adopted an iterative development approach with Git version control for progressive enhancement of the data converter. Each iteration focused on a specific functionality:

Version 1: Established the core conversion logic through a command-line interface (CLI).

Version 2: Integrated a 1602 LCD display to visualize conversion results.

Version 3: (Optional, if implemented) Introduced touch sensors for binary input, enhancing user interaction.

Version 4: Incorporated LEDs to represent the binary equivalent visually.

Version 5: Added a buzzer for potential audio feedback or conversion confirmation.

Version 6: Making the code asynchronous to run both lcd displays together and tried to use both cores of the pico cpu by threading.

Version 7: Culminated in web server integration, enabling remote control of the converter.

Git facilitated tracking changes, reverting to previous states if necessary, and ensuring a smooth development process. This version control system allowed for efficient collaboration and maintained a clear history of the project's evolution.

## Code of project

**convert_with_lcd.py**

```
from machine import I2C, Pin
import utime
from pico_i2c_lcd import I2cLcd
import buzzer
import uasyncio
import NetworkCredentials
import RequestParser
import Response Builder



no_of_led = 12

led = [Pin(i, Pin.OUT) for i in range(no_of_led)]

button_0 = Pin(14, mode=Pin.IN)
button_1 = Pin(15, mode=Pin.IN)

def button_interrupt_INT(pin):        # PB_Switch Interrupt handler
    global stop_conversion  # Flag to stop convert()
    global count
    count = 0
    global bin_str
    bin_str = "0b"
    for i in range(12):
        led[i].off()
    if button_interrupt.value() == 1:  # Button press detected
        if stop_conversion:  # If stopping conversion
            print("Interrupt: Resuming Conversion")
            stop_conversion = False  # Reset stop flag
            lcd.clear()
            lcd.move_to(0, 0)
            lcd.putstr("Bin:")

        else:  # If conversion not running (or just finished)
            print("Interrupt: Stopping Conversion")
            stop_conversion = True  # Set stop flag
            lcd.clear()
            lcd.move_to(0, 0)
            lcd.putstr("Bin:")


    button_interrupt.irq(handler=button_interrupt_INT)


def bin_input():
    global bin_str
    bin_str = "0b"
    for i in range(12):
        led[i].off()
    global count
    count = 0
    lcd.clear()
    lcd.move_to(0, 0)
    lcd.putstr("Bin:")
    while count < 12 :
        if (button_0.value() == 1 and button_1.value() ==0):
            led[11-count].off()
            count+=1
            bin_str += "0"
            lcd.putstr("0")



        if (button_0.value() == 0 and button_1.value() ==1):
            led[11-count].on()
            count+=1
            bin_str += "1"
            lcd.putstr("1")

        utime.sleep(.12)
    buzzer.play_tone(500)
    print(bin_str)
    return bin_str



def bin_to_dec(bin_num):
    return int(bin_num, 2)

def bin_to_hex(bin_num):
    return hex(int(bin_num, 2))[2:]

def bin_to_oct(bin_num):
    return oct(int(bin_num, 2))[2:]

def bin_to_excess3(bin_num):
    decimal = bin_to_dec(bin_num)
    result = bin(decimal+3)[2:]
    i = 0
    while i<len(result) and result[i]=='0':
        i+=1
    return result[i:]

def bin_to_gray(bin_num):
    bin_str = str(bin_num)  # Convert integer to string
    result = ''
    result += bin_str[0]
    for i in range(1, len(bin_str)):
        result += str(int(bin_str[i - 1]) ^ int(bin_str[i]))
    return result



# Add more conversion functions as needed

def convert_to(data_type, bin_num):

    if data_type == "Dec":
        result = bin_to_dec(bin_num)
    elif data_type == "Hex":
        result = bin_to_hex(bin_num)
    elif data_type == "Oct":
        result = bin_to_oct(bin_num)
    elif data_type == "EX3":
        result = bin_to_excess3(bin_num)
    elif data_type == "GRY":
        result = bin_to_gray(bin_num)

    return result

def display(data_type, bin_num, result):
    lcd.move_to(0, 0)
    lcd.putstr("Bin:{}\n".format(bin_num))
    lcd.move_to(0, 1)  # Move to the second row
    # Only update the characters that change
    lcd.putstr("{:<16}".format("{}:{}".format(data_type, result)))

def input_available():
    return button_pin.value() == 0  # Assuming the button pin is grounded when pressed

def getdata():
    global data
    return data

async def convert(binary=None):
    data_types = ["Dec", "Hex", "Oct", "EX3", "GRY"]

    global run  # Access the global run variable
    global stop_conversion  # Flag to stop convert()
    for i in range(12):
        led[i].off()
    while True:
        stop_conversion = False  # Reset stop flag before starting conversion
        if binary == None:
            bin_num = bin_input()[2:]
        else:
            bin_num = binary
            leading = 12-len(bin_num)
            for count in range(len(bin_num)):
                if bin_num[count] == '0':
                    led[11-leading-count].off()
                elif bin_num[count] == '1':
                    led[11-leading-count].on()
            binary = None
        data["Bin"] = bin_num
        for data_type in data_types:
            data[data_type] = convert_to(data_type, bin_num)
        print(data)
#        await uasyncio.sleep(1)


        while run and not stop_conversion:  # Loop until interrupted or stop flag set
            for data_type in data_types:
                display(data_type, bin_num, data[data_type])
#                utime.sleep(1)
                await uasyncio.sleep(1)
                if stop_conversion:
                    break
            await uasyncio.sleep(1)
```

```
def MIT_LOGO():

    #Panel 00
    lcd2.custom_char(0, bytearray([
    0x00,
    0x00,
    0x00,
    0x00,
    0x01,
    0x02,
    0x05,
    0x05

        ]))

    #Panel 10
    lcd2.custom_char(1, bytearray([
    0x1F,
    0x1F,
    0x15,
    0x1F,
    0x00,
    0x1F,
    0x15,
    0x15
        ]))



    #Panel 01
    lcd2.custom_char(2, bytearray([
    0x00,
    0x04,
    0x04,
    0x1F,
    0x1B,
    0x0A,
    0x11,
    0x11


        ]))

    #Panel 11
    lcd2.custom_char(3, bytearray([
    0x1F,
    0x1F,
    0x15,
    0x1F,
    0x00,
    0x0E,
    0x04,
    0x0E
```

```
        ]))

    #Panel 02
    lcd2.custom_char(4, bytearray([
    0x00,
    0x00,
    0x00,
    0x00,
    0x10,
    0x08,
    0x14,
    0x14

        ]))

    #Panel 12
    lcd2.custom_char(5, bytearray([
    0x1F,
    0x1F,
    0x15,
    0x1F,
    0x00,
    0x0E,
    0x04,
    0x04


        ]))

    lcd2.move_to(0,0)
    lcd2.putchar(chr(0))
    lcd2.move_to(0,1)
    lcd2.putchar(chr(1))
    lcd2.move_to(1,0)
    lcd2.putchar(chr(2))
    lcd2.move_to(1,1)
    lcd2.putchar(chr(3))
    lcd2.move_to(2,0)
    lcd2.putchar(chr(4))
    lcd2.move_to(2,1)
    lcd2.putchar(chr(5))




async def textscroll(length,string):
    x = lcd2.cursor_x
    y = lcd2.cursor_y
    i=0
    for i in range(len(string)-length+1):
        updated_str = string[i:i+length]
        lcd2.move_to(x,y)
        lcd2.putstr(" "*length)
        lcd2.move_to(x,y)
        lcd2.putstr(updated_str)
        i += 1
        await uasyncio.sleep(1)
```

```
async def lcd_2():
    while True:
        lcd2.move_to(4, 1)
        lcd2.putstr(" Group : 07")
        await uasyncio.sleep(5)
        lcd2.move_to(4, 1)
        string = "FCASD MiniProject"

        length = 11
        x = lcd2.cursor_x
        y = lcd2.cursor_y
        i=0
        for i in range(len(string)-length+1):
            updated_str = string[i:i+length]
            lcd2.move_to(x,y)
            lcd2.putstr(" "*length)
            lcd2.move_to(x,y)
            lcd2.putstr(updated_str)
            i += 1
            await uasyncio.sleep(1)
        await uasyncio.sleep(5)


def init():
    # Initialize LCD, buttons, and other resources
    i2c = I2C(0, sda=Pin(20), scl=Pin(21))
    I2C_ADDR = i2c.scan()[0]
    global lcd
    lcd = I2cLcd(i2c, I2C_ADDR, 2, 16)

    I2C_ADDR = 38
    I2C_NUM_ROWS = 2
    I2C_NUM_COLS = 16

    i2c = I2C(1, sda=Pin(26), scl=Pin(27), freq=400000)
    global lcd2
    lcd2 = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

    global button_interrupt
    button_interrupt =  Pin(22, Pin.IN,  Pin.PULL_DOWN)
    button_interrupt.irq(trigger=Pin.IRQ_RISING, handler=button_interrupt_INT)

    global run
    run = True
    global stop_conversion
    stop_conversion = False
    global state
    state = 0

    MIT_LOGO()
    lcd2.move_to(6, 0)
    lcd2.putstr("MIT-WPU")
    lcd2.move_to(4, 1)
    lcd2.putstr(" Group : 07")
    global data
    data = dict()
```

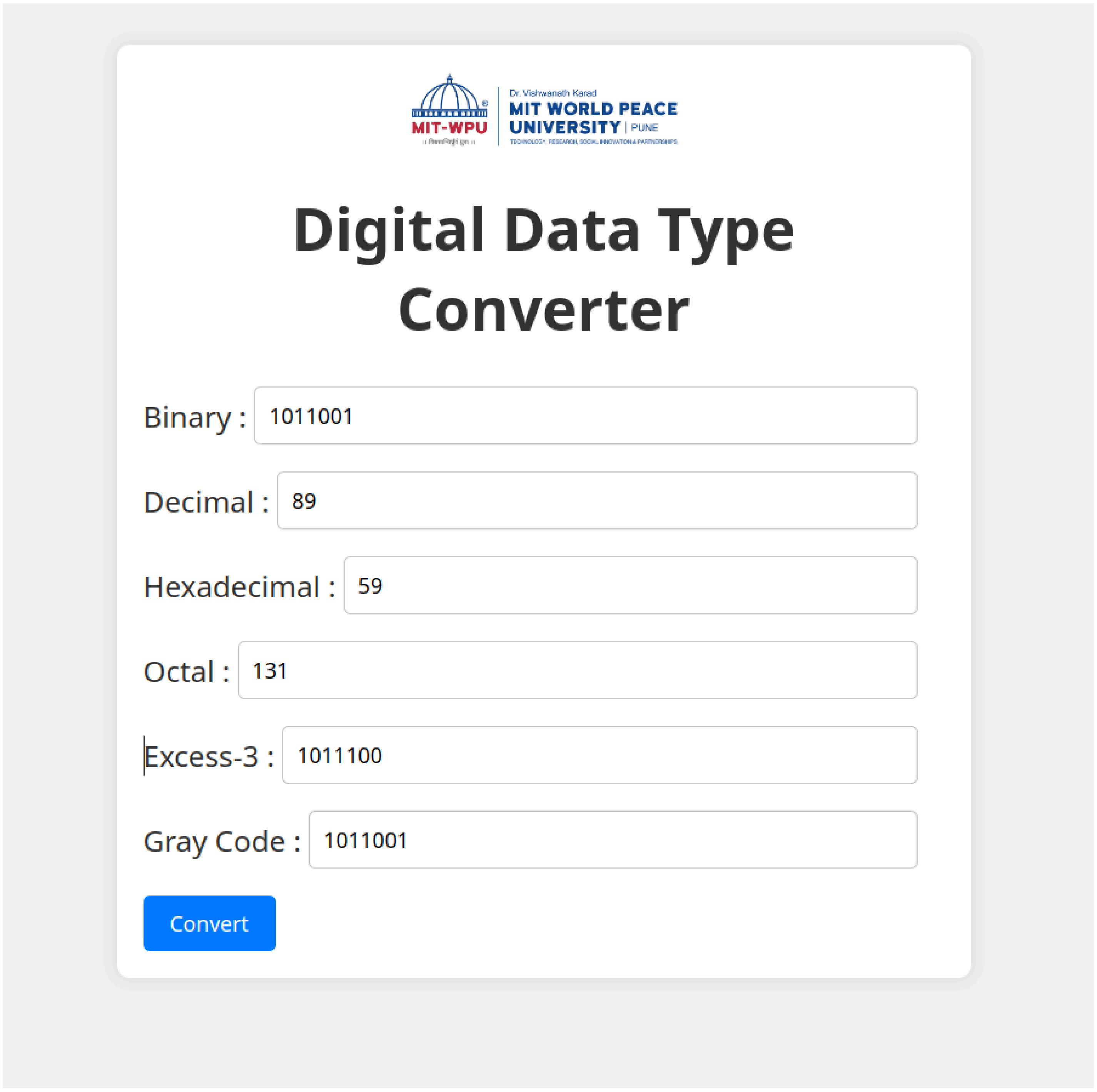
```
def uasync(data=None):
    # Create the event loop and tasks
    async def main():
        # Create asynchronous tasks
        task1 = uasyncio.create_task(convert(data))
        task2 = uasyncio.create_task(lcd_2())

        # Run these tasks concurrently
        await uasyncio.gather(task1, task2)

async def webserver():
    print('Setting up webserver...')
    server = uasyncio.start_server(handle_request, "0.0.0.0", 80)
    uasyncio.create_task(server)


second_thread = _thread.start_new_thread(webserver, ())


    # Start the event loop and run the 'main' coroutine
    try:
        uasyncio.run(main())
    except Exception as e:
        print("An error occurred:", e)
    finally:
        # Reset the event loop after completion or error
        uasyncio.new_event_loop()
```

**buzzer.py**

```
from picozero import Speaker
from time import sleep
import math

# Define frequencies for high and low tones
HIGH_FREQ = 1000  # Adjust for desired high frequency (Hz)
LOW_FREQ = 100   # Adjust for desired low frequency (Hz)

# Define duration of each tone
DURATION = 0.4  # Adjust for desired tone duration (seconds)
DELAY = 0.1 # Duration between each char

speaker = Speaker(13)


def play_tone(frequency):
    speaker.play(int(frequency), duration=DURATION)  # Play tone with specified frequency and
duration
    sleep(DELAY)

def play_binary(binary_string):
    # Define frequencies for high and low tones
    FREQ_1 = 1000
    FREQ_0 = 500

    for bit in binary_string:
        if bit == "1":
            play_tone(FREQ_1)
        else:
            play_tone(FREQ_0)


def calculate_frequency(base, index, min_freq=200, max_freq=5000):
    """Calculates frequency based on position within a number system."""
    return round(min_freq + (index / (base - 1)) * (max_freq - min_freq), 2)

def play_octal(octal_string):
    base = 8
    frequencies = [calculate_frequency(base, i) for i in range(base)]

    for digit in octal_string:
        index = int(digit)
        play_tone(frequencies[index])

def play_decimal(decimal_string):
    base = 10
    frequencies = [calculate_frequency(base, i) for i in range(base)]

    for digit in decimal_string:
        index = int(digit)
        play_tone(frequencies[index])

def play_hexadecimal(hexadecimal_string):
    base = 16
    frequencies = [calculate_frequency(base, i) for i in range(base)]

    for digit in hexadecimal_string:
        index = int(digit, 16)  # Handle both digits and letters (A-F)
        #index = value if value < 10 else value - 10  # Adjust index for A-F
        play_tone(frequencies[index])
```

Note all of the code is version controlled using git on (This link has all of the code including of the web server ) https://github.com/Ayush-Kadali/Digital_Converter_using_Pi_Pico_W

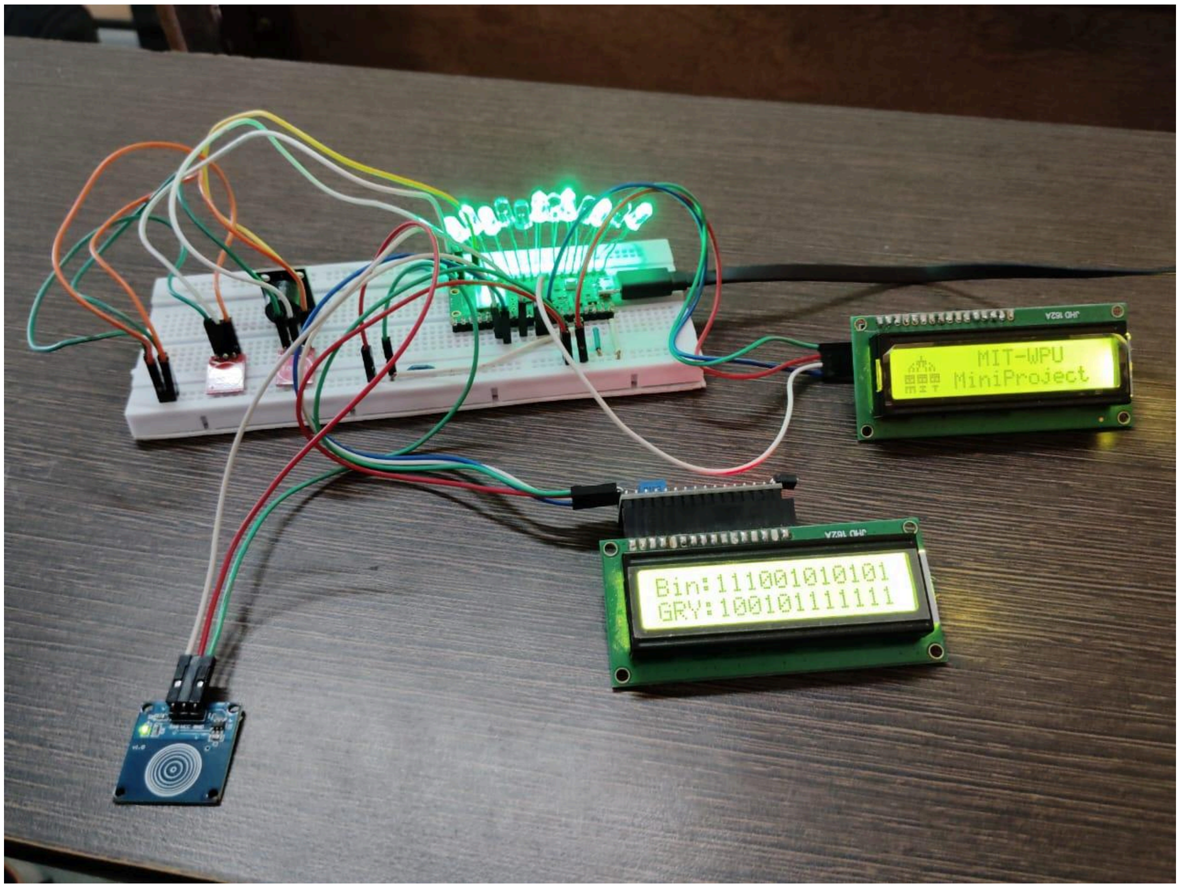
## Snapshots

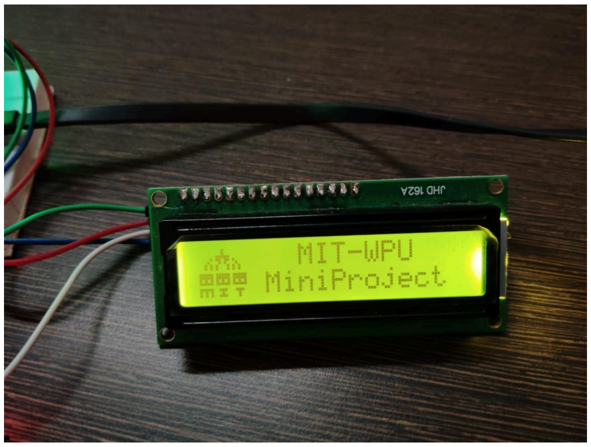


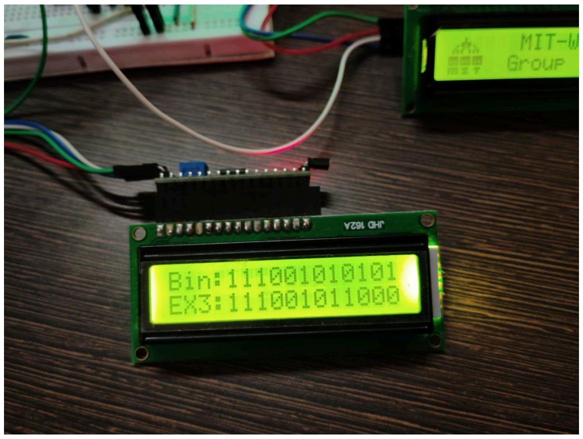
Dr. Vishwanath Karad MIT WORLD PEACE UNIVERSITY | PUNE

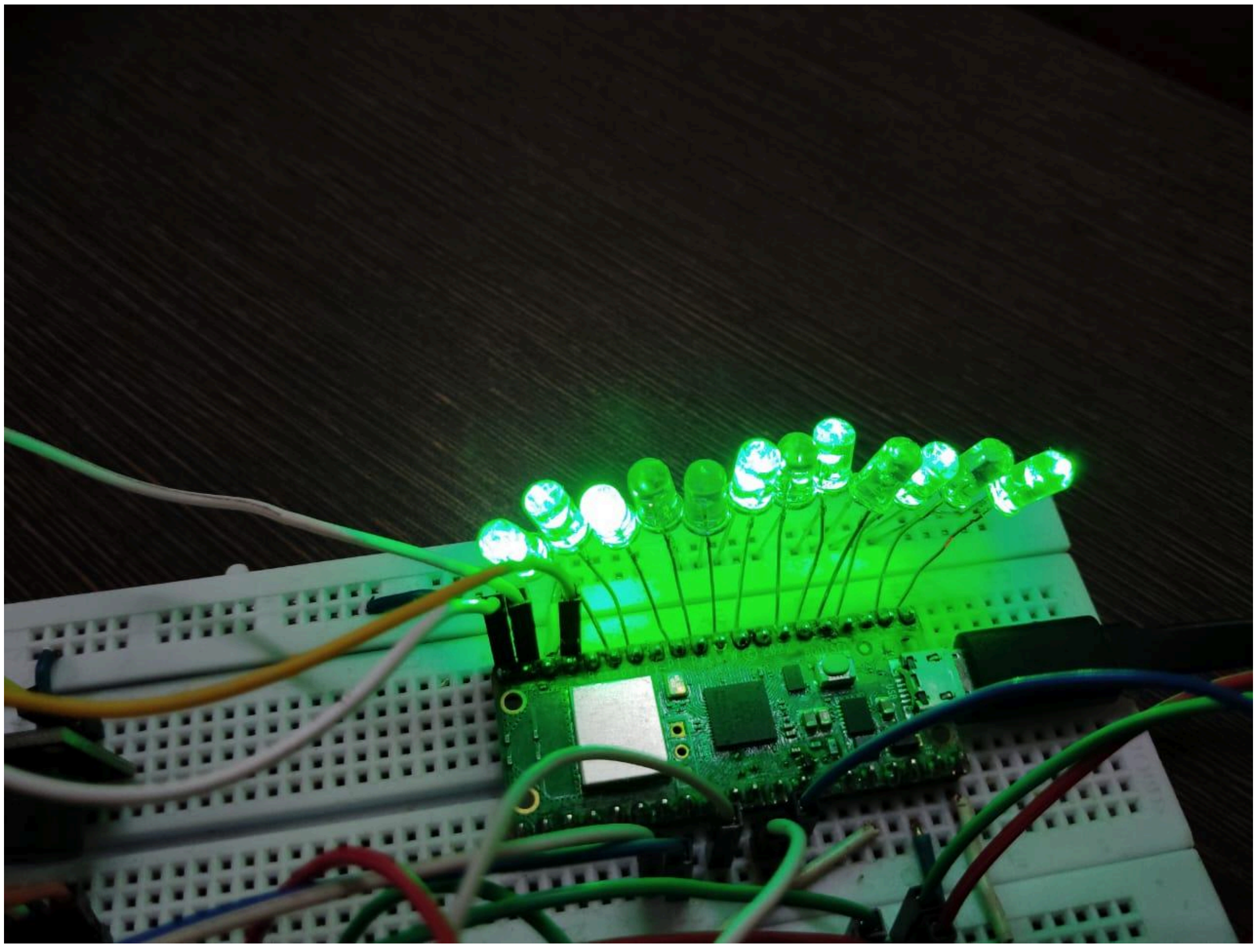
# Digital Data Type Converter

| | |
|---|---|
| Binary : | 1011001 |
| Decimal : | 89 |
| Hexadecimal : | 59 |
| Octal : | 131 |
| Excess-3 : | 1011100 |
| Gray Code : | 1011001 |


MIT-WPU
MiniProject
Bin:111001010101
GRY:100101111111


MIT-WPU
MiniProject


Bin:111001010101
EX3:111001011000
Group



## Name and roll no of students

| | |
|---|---|
| Ayush Kadali | 102045 |
| Dvit Gohil | 102037 |
| Chinmay Gogate | 102036 |
| Vedant Hulage | 102041 |