ABSTRACT

A real-time digital clock is developed using a Raspberry Pi in combination with a 4-digit 7-segment LED display driven by the TM1637 module. The system is designed to retrieve the current time from the Raspberry Pi's internal clock and display it in HH:MM format. A Python-based program handles the time acquisition and drives the display using GPIO communication. Specifically, GPIO 23 is assigned for the CLK (clock) signal and GPIO 24 for the DIO (data input/output), with power supplied through the 3.3V and GND pins.

The TM1637 module simplifies the interface by requiring only two GPIO pins and supports efficient communication for updating digit segments. A specialized TM1637 Python library is utilized to control brightness, refresh the time display every second, and toggle the colon (:) symbol to indicate the passage of seconds. This not only enhances readability but also visually mimics traditional digital clocks.

The implementation provides a practical example of embedded systems programming, digital display control, and hardware-software integration using a high-level language. It also serves as a foundational exercise in working with GPIO pins and interfacing microcontroller-level hardware with the Raspberry Pi. Applications of such systems range from educational tools and DIY electronics to basic home automation components.

TABLE OF CONTENTS

1.AIM: Displaying time over 4-Digit 7-Segment display using Raspberry pi model 3b+.

2.COMPONENTS REQUIRED:

SOFTWARE_REQUIREMENTS:

Raspbian Stretch OS

Python

HARDWARE REQUIREMENTS:

Raspberry pi Model 3b+

4-Digit 7-Segment Display LED (with TM1637 controller)

Jumper Wires (Female to Female)

USB Cables

Wi-Fi or Ethernet

Power Cable

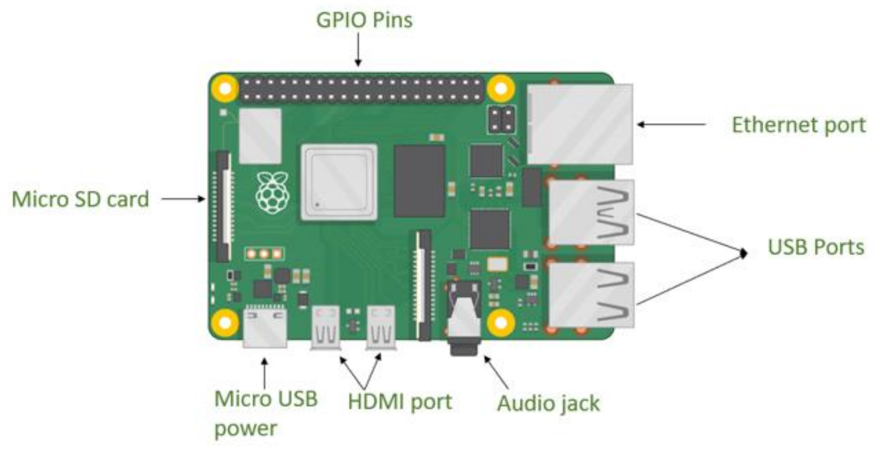
SD Card (for Memory)

3.THEORY:

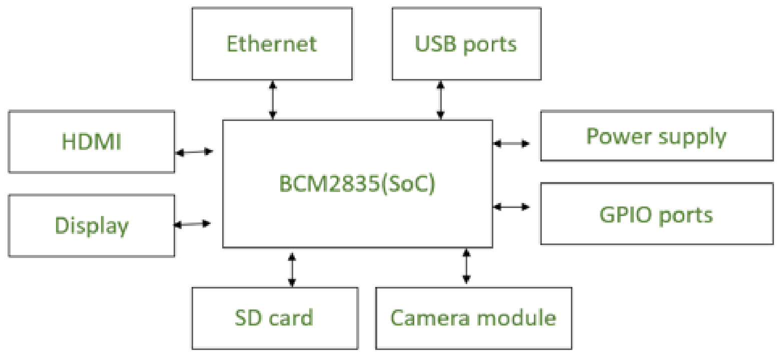
Raspberry Pi Model 3b+

Raspberry Pi is a small single-board computer (SBC). It is a credit card-sized computer that can be plugged into a monitor. It acts as a minicomputer by connecting the keyboard, mouse, and display. Raspberry Pi has an ARM processor and 512MB of RAM. The architecture of Raspberry Pi is discussed in this article.

The following diagram shows the architecture of Raspberry Pi:



The following diagram shows some main blocks of Raspberry Pi:



Raspberry Pi mainly consists of the following blocks:

- **Processor:** Raspberry Pi uses Broadcom BCM2835 system on chip which is an ARM processor and Video core Graphics Processing Unit (GPU). It is the heart of the Raspberry Pi which controls the operations of all the connected devices and handles all the required computations.
- **HDMI:** High Definition Multimedia Interface is used for transmitting video or digital audio data to a computer monitor or to digital TV. This HDMI port helps Raspberry Pi to connect its signals to any digital device such as a monitor digital TV or display through an HDMI cable.
- **GPIO ports:** General Purpose Input Output ports are available on Raspberry Pi which allows the user to interface various I/P devices.
- **Audio output**: An audio connector is available for connecting audio output devices such as headphones and speakers.
- **USB ports:** This is a common port available for various peripherals such as a mouse, keyboard, or any other I/P device. With the help of a USB port, the system can be expanded by connecting more peripherals.
- **SD card: The** SD card slot is available on Raspberry Pi. An SD card with an operating system installed is required for booting the device.
- **Ethernet**: The ethernet connector allows access to the wired network, it is available only on the model B of Raspberry Pi.
- **Power supply:** A micro USB power connector is available onto which a 5V power supply can be connected.
- **Camera module:** Camera Serial Interface (CSI) connects the Broadcom processor to the Pi camera.
- **Display:** Display Serial Interface (DSI) is used for connecting LCD to Raspberry Pi using 15 15-pin ribbon cables. DSI provides a high-resolution display interface that is specifically used for sending video data.

Warnings:

Use only a 5V/2.5A DC power supply that meets local regulations. Operate the Raspberry Pi 3 Model B+ in a well-ventilated area, and avoid covering it if enclosed. Place it on a stable, non-conductive surface, away from conductive items. Connecting incompatible devices to GPIO pins can cause damage and void the warranty. Ensure all peripherals (keyboards, monitors, mice, etc.) comply with local standards and have proper insulation for safety.

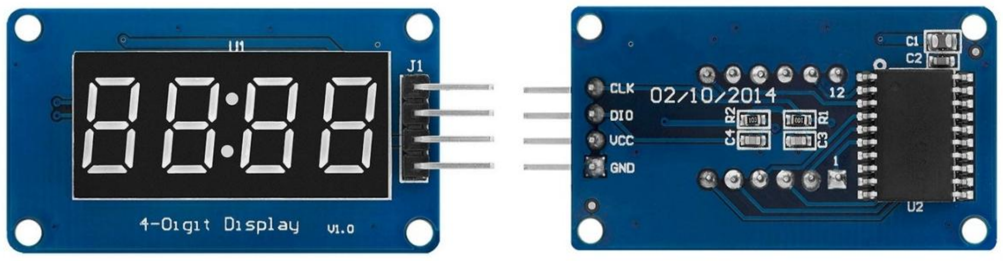
Safety Instructions:

- Keep away from **water, moisture**, and **conductive surfaces** during use.
- Avoid **exposure to heat**; operate at normal ambient temperatures.
- Do not expose the board to **high-intensity light** (e.g., flash or laser) while powered.
- Handle with care to prevent **mechanical or electrical damage**.
- When powered, handle only by the **edges** to reduce the risk of **electrostatic discharge**.

4-Digit 7-Segment Display LED with TM1637 Controller:

Light Emitting Diode (LED) is the most widely used semiconductor which emits either visible light or invisible infrared light when forward biased. Remote controls generate invisible light. A Light-emitting diode (LED) is optical-electrical energy into light energy when voltage is applied.

Seven segment displays are the output display device that provides a way to display information in the form of images or text or decimal numbers which is an alternative to the more complex dot matrix displays. It is widely used in digital clocks, basic calculators, electronic meters, and other electronic devices that display numerical information. It consists of seven segments of light-emitting diodes (LEDs) which are assembled like numerical 8.



This module combines a 4-digit 7-segment LED display with a **TM1637 driver IC**, which simplifies control over the display using just two pins (CLK and DIO). It is commonly used in projects involving time, counters, or any application where numerical display is required.

Pin Description

- **CLK**: Clock signal for data communication
- **DIO**: Data Input/Output
- **VCC**: Power supply (3.3V–5V)
- **GND**: Ground

Working Principle

The TM1637 IC manages the communication and control of the 4-digit 7-segment display. By sending serial commands via the CLK and DIO pins, you can display numbers, control brightness, and turn the display on/off without the need to manually handle each segment.

Applications

- Digital clocks
- Timers
- Scoreboards
- Simple numeric displays for embedded projects
- DIY electronics using Raspberry Pi or Arduino

Advantages

- Easy to use with only 2 GPIO pins
- Compact and low power
- Widely supported with open-source libraries (e.g., tm1637 Python library for Raspberry Pi)

Jumper Wires:

Jumper wires are small, conductive wires with connectors used for temporary, flexible connections in electronics, especially on breadboards. They enable quick circuit modifications without soldering and come in male-to-male, male-to-female, and female-to-female types. Their main function is to bridge components during prototyping or troubleshooting. While wire colors have no electrical significance, they help visually distinguish connections like power and ground.

Types of Jumper wires:

Jumper wires come in three versions:

1. Male-to-male jumper
2. Male-to-female jumper
3. Female-to-female jumper

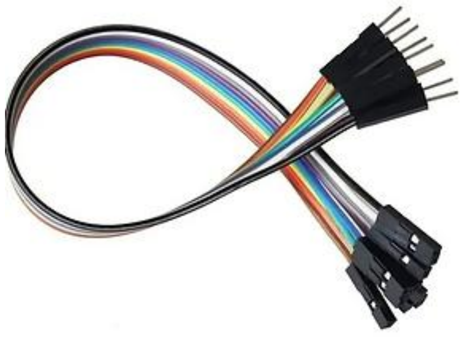
There are two different kinds of head shapes: round and square. The wire's termination determines how each differs from the other. Female ends are also used for plugging; however they do not include a pin that protrudes as male ends do.

1. Male-to-Male jumper:

With the help of the male-to-male breadboard jumper wire, creating your own circuits on a breadboard is simple. These cables have a prototype board connection on both ends, making them versatile. For building circuits between your microcontroller and the breadboard on the bots, the jumper cable is perfect.

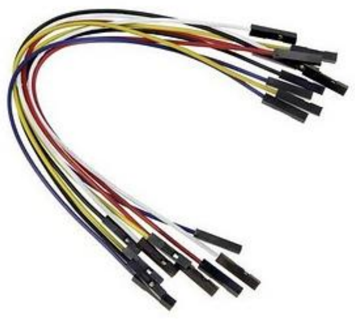
2. Male-to-Female jumper:

These male-to-female jumper cables are used to join any development board with a female header pin to other development boards with a male connection. These are straightforward cables with connection pins on both ends that enable you to link two locations together.

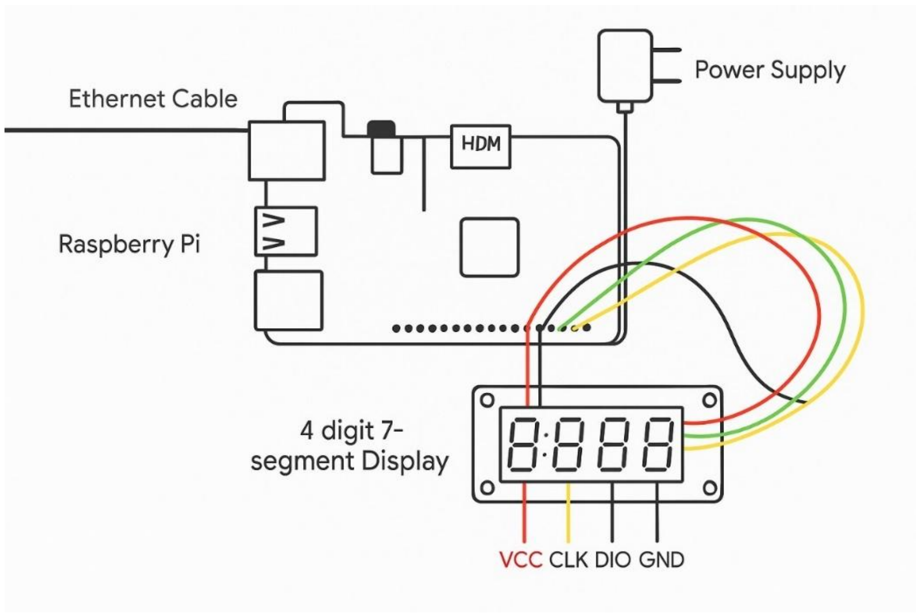


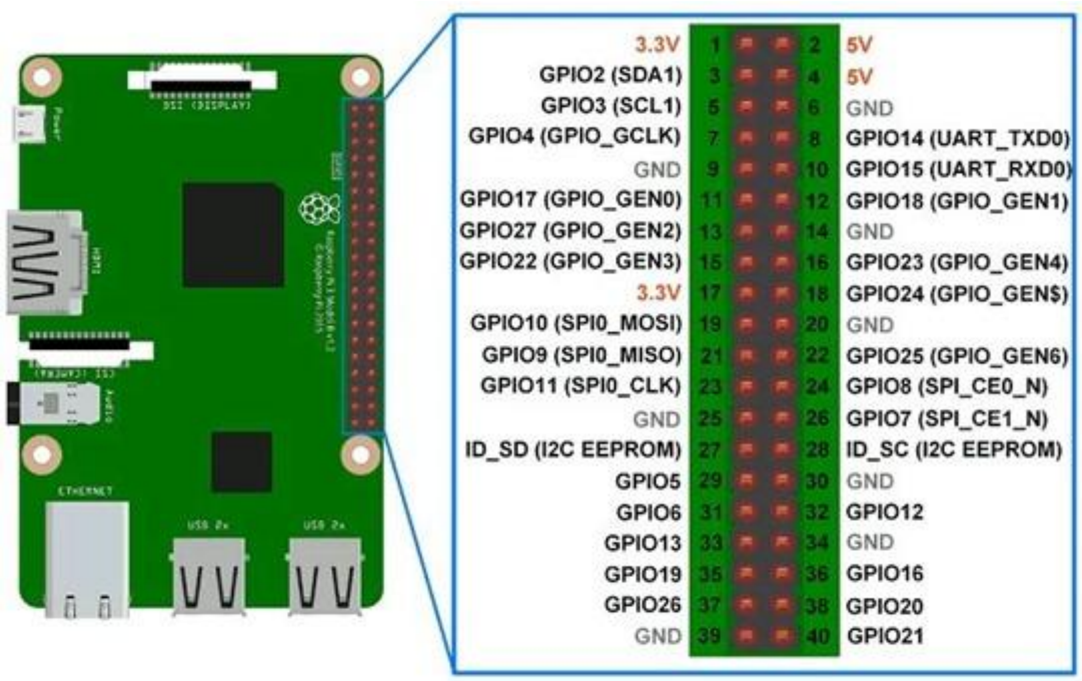
3. Female-to-Female jumper:

These are jumper wires that link the female header pin of an Arduino or other development board to the female pin of another development board. These are basic cables with connector pins on both ends that enable you to connect two places to one another.



4.CIRCUIT CONNECTION AND RASPBERRY PIN DESCRIPTION:




3.3V 1 2 5V
GPIO2 (SDA1) 3 4 5V
GPIO3 (SCL1) 5 6 GND
GPIO4 (GPIO_GCLK) 7 8 GPIO14 (UART_TXD0)
GND 9 10 GPIO15 (UART_RXD0)
GPIO17 (GPIO_GEN0) 11 12 GPIO18 (GPIO_GEN1)
GPIO27 (GPIO_GEN2) 13 14 GND
GPIO22 (GPIO_GEN3) 15 16 GPIO23 (GPIO_GEN4)
3.3V 17 18 GPIO24 (GPIO_GEN$)
GPIO10 (SPI0_MOSI) 19 20 GND
GPIO9 (SPI0_MISO) 21 22 GPIO25 (GPIO_GEN6)
GPIO11 (SPI0_CLK) 23 24 GPIO8 (SPI_CE0_N)
GND 25 26 GPIO7 (SPI_CE1_N)
ID_SD (I2C EEPROM) 27 28 ID_SC (I2C EEPROM)
GPIO5 29 30 GND
GPIO6 31 32 GPIO12
GPIO13 33 34 GND
GPIO19 35 36 GPIO16
GPIO26 37 38 GPIO20
GND 39 40 GPIO21

5.PROCEDURE:

1. Hardware Setup

- Connect the TM1637 4-digit 7-segment display to the Raspberry Pi GPIO pins:

- DIO (Data In/Out) → GPIO 20

- CLK (Clock) → GPIO 21

- VCC → 3.3V

- GND → GND

2. Download Project Files

- Visit the GitHub repository:

  https://github.com/timwaizenegger/raspberrypi-examples

- Navigate to the actor-led-7segment-4numbers folder.

- Click on Code > Download ZIP.

- Extract the ZIP file to a folder on your Raspberry Pi.

3. Transfer and Modify Files

- Copy the necessary files: Clock.py and tm1637.py to your working directory.

- Modify the files as needed to suit your project requirements.

4. Run the Clock Program

- Open a terminal and navigate to your project directory.

- Run the Python script Clock.py.

- The 7-segment display will now show the current time, updating in real-time.

5. Stop the Clock

- Press CTRL + C in the terminal to stop the clock.

- The script will clean up GPIO pins safely using the cleanup() method.

SOURCE CODE:

Clock.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from time import sleep
import tm1637
try:
    import thread
except ImportError:
    import _thread as thread
# Initialize the clock (GND, VCC=3.3V, Example Pins are DIO-20 and CLK21)
Display = tm1637.TM1637(CLK=21, DIO=20, brightness=1.0)
try:
    print ("Starting clock in the background (press CTRL + C to stop):")
    Display.StartClock(military_time=False)
    print ('Continue Python script and tweak Display!')
    sleep(1)
    Display.ShowDoublepoint(False)
    sleep(1)
    loops = 3
```

```
except KeyboardInterrupt:
    print ("Properly closing the clock and open GPIO pins")
    Display.cleanup()
```

Tm1637.py:

```
import math
import RPi.GPIO as IO
import threading
from time import sleep, localtime
# from tqdm import tqdm
# IO.setwarnings(False)
IO.setmode(IO.BCM)
HexDigits = [0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
             0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71]
ADDR_AUTO = 0x40
ADDR_FIXED = 0x44
STARTADDR = 0xC0
# DEBUG = False
class TM1637:
    __doublePoint = False
    __Clkpin = 0
    __Datapin = 0
    __brightness = 1.0  # default to max brightness
    __currentData = [0, 0, 0, 0]
    def __init__(self, CLK, DIO, brightness):
        self.__Clkpin = CLK
        self.__Datapin = DIO
        self.__brightness = brightness
        IO.setup(self.__Clkpin, IO.OUT)
        IO.setup(self.__Datapin, IO.OUT)
    def cleanup(self):
        """Stop updating clock, turn off display, and cleanup GPIO"""
        self.StopClock()
        self.Clear()
        IO.cleanup()
    def Clear(self):
        b = self.__brightness
        point = self.__doublePoint
        self.__brightness = 0
        self.__doublePoint = False
        data = [0x7F, 0x7F, 0x7F, 0x7F]
        self.Show(data)
        # Restore previous settings:
        self.__brightness = b
        self.__doublePoint = point
    def ShowInt(self, i):
        s = str(i)
        self.Clear()
        for i in range(0, len(s)):
            self.Show1(i, int(s[i]))
    def Show(self, data):
        for i in range(0, 4):
            self.__currentData[i] = data[i]
        self.start()
        self.writeByte(ADDR_AUTO)
        self.br()
        self.writeByte(STARTADDR)
        for i in range(0, 4):
            self.writeByte(self.coding(data[i]))
        self.br()
        self.writeByte(0x88 + int(self.__brightness))
        self.stop()
    def Show1(self, DigitNumber, data):
        """show one Digit (number 0...3)"""
        if(DigitNumber < 0 or DigitNumber > 3):
            return  # error
        self.__currentData[DigitNumber] = data
        self.start()
        self.writeByte(ADDR_FIXED)
        self.br()
        self.writeByte(STARTADDR | DigitNumber)
        self.writeByte(self.coding(data))
        self.br()
        self.writeByte(0x88 + int(self.__brightness))
        self.stop()
    def SetBrightness(self, percent):
        """Accepts percent brightness from 0 - 1"""
        max_brightness = 7.0
        brightness = math.ceil(max_brightness * percent)
        if (brightness < 0):
            brightness = 0
        if(self.__brightness != brightness):
            self.__brightness = brightness
            self.Show(self.__currentData)
    def ShowDoublepoint(self, on):
        """Show or hide double point divider"""
        if(self.__doublePoint != on):
            self.__doublePoint = on
            self.Show(self.__currentData)
    def writeByte(self, data):
        for i in range(0, 8):
            IO.output(self.__Clkpin, IO.LOW)
            if(data & 0x01):
                IO.output(self.__Datapin, IO.HIGH)
            else:
                IO.output(self.__Datapin, IO.LOW)
            data = data >> 1
            IO.output(self.__Clkpin, IO.HIGH)
        # wait for ACK
        IO.output(self.__Clkpin, IO.LOW)
        IO.output(self.__Datapin, IO.HIGH)
        IO.output(self.__Clkpin, IO.HIGH)
        IO.setup(self.__Datapin, IO.IN)
        while(IO.input(self.__Datapin)):
            sleep(0.001)
            if(IO.input(self.__Datapin)):
                IO.setup(self.__Datapin, IO.OUT)
                IO.output(self.__Datapin, IO.LOW)
                IO.setup(self.__Datapin, IO.IN)
        IO.setup(self.__Datapin, IO.OUT)
    def start(self):
        """send start signal to TM1637"""
        IO.output(self.__Clkpin, IO.HIGH)
        IO.output(self.__Datapin, IO.HIGH)
        IO.output(self.__Datapin, IO.LOW)
        IO.output(self.__Clkpin, IO.LOW)
    def stop(self):
        IO.output(self.__Clkpin, IO.LOW)
        IO.output(self.__Datapin, IO.LOW)
        IO.output(self.__Clkpin, IO.HIGH)
        IO.output(self.__Datapin, IO.HIGH)
    def br(self):
        """terse break"""
        self.stop()
        self.start()
    def coding(self, data):
        if(self.__doublePoint):
            pointData = 0x80
        else:
            pointData = 0
        if(data == 0x7F):
            data = 0
        else:
            data = HexDigits[data] + pointData
        return data
    def clock(self, military_time):
        """Clock script modified from:
            https://github.com/johnlr/raspberrypi-tm1637"""
        self.ShowDoublepoint(True)
        while (not self.__stop_event.is_set()):
            t = localtime()
            hour = t.tm_hour
            if not military_time:
                hour = 12 if (t.tm_hour % 12) == 0 else t.tm_hour % 12
            d0 = hour // 10 if hour // 10 else 0
            d1 = hour % 10
            d2 = t.tm_min // 10
            d3 = t.tm_min % 10
            digits = [d0, d1, d2, d3]
            self.Show(digits)
            # # Optional visual feedback of running alarm:
            # print digits
            # for i in tqdm(range(60 - t.tm_sec)):
            for i in range(60 - t.tm_sec):
                if (not self.__stop_event.is_set()):
                    sleep(1)
    def StartClock(self, military_time=True):
        # Stop event based on: http://stackoverflow.com/a/6524542/3219667
        self.__stop_event = threading.Event()
        self.__clock_thread = threading.Thread(
            target=self.clock, args=(military_time,))
        self.__clock_thread.start()
    def StopClock(self):
        try:
            print ('Attempting to stop live clock')
            self.__stop_event.set()
        except:
            print ('No clock to close')
if __name__ == "__main__":
    """Confirm the display operation"""
    display = TM1637(CLK=21, DIO=20, brightness=1.0)
    display.Clear()
    digits = [1, 2, 3, 4]
    display.Show(digits)
    print ("1234  - Working? (Press Key)")
    scrap = raw_input()
    print ("Updating one digit at a time:")
    display.Clear()
    display.Show1(1, 3)
    sleep(0.5)
    display.Show1(2, 2)
    sleep(0.5)
    display.Show1(3, 1)
    sleep(0.5)
```

```
display.Show1(0, 4)
print ("4321  - (Press Key)")
scrap = raw_input()
print ("Add double point\n")
display.ShowDoublepoint(True)
sleep(0.2)
print ("Brightness Off")
display.SetBrightness(0)
sleep(0.5)
print ("Full Brightness")
display.SetBrightness(1)
sleep(0.5)
print ("30% Brightness")
display.SetBrightness(0.3)
sleep(0.3)
# See clock.py for how to use the clock functions!
```

6.OBSERVATION:

This project implements a real-time digital clock using a **Raspberry Pi 3B+** and a **4-digit 7-segment display** driven by the **TM1637 module**. It uses only two GPIO pins for control and employs **multiplexing** to display digits efficiently. The Python script fetches the current time via the datetime module, with a blinking colon (:) for visual feedback using Display.ShowDoublepoint(second % 2). Brightness is adjustable (levels 0–7) for varying light conditions. The code is modular, separating the display driver and main logic for clarity and reusability.

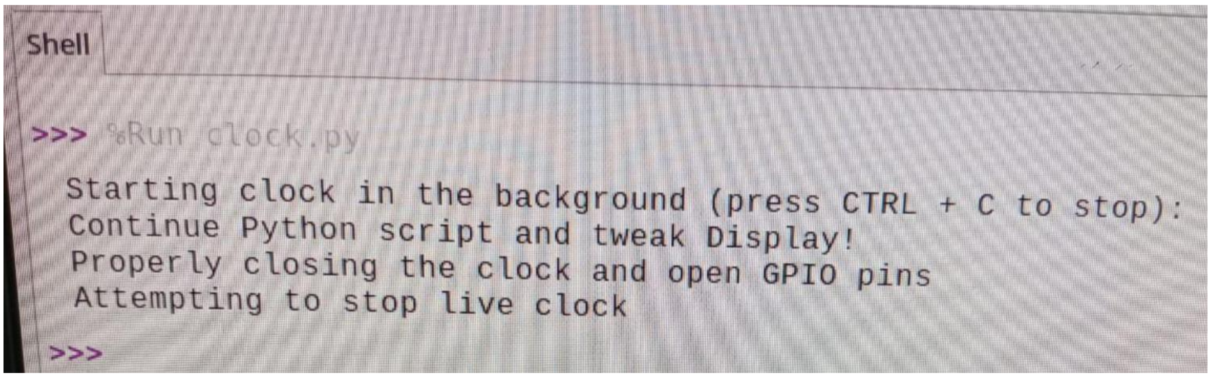
7.ANALYSIS:

This project demonstrates effective Raspberry Pi-to-display interfacing using minimal GPIO pins via the TM1637 driver. Real-time timekeeping is handled by Python's datetime module, showcasing Python's power in embedded systems. Multiplexing ensures efficient, flicker-free digit display. A blinking colon offers a clear time progression cue. The modular code structure enhances readability, debugging, and scalability. Adjustable brightness improves visibility across lighting conditions. Overall, the project is a strong example of integrating Python software with hardware, ideal for learning IoT and embedded systems development.

8.PRECAUTIONS:

- Always power off the Raspberry Pi before connecting or disconnecting any hardware.
- Check GPIO connections carefully to avoid short circuits or incorrect wiring.
- Use a stable power supply (5V, 2.5A or more) for reliable operation.
- Avoid running multiple scripts that access the GPIO at the same time.
- Handle components with care to prevent static discharge or physical damage.

9.OUTPUT:



Shell

```
>>> %Run clock.py

  Starting clock in the background (press CTRL + C to stop):
  Continue Python script and tweak Display!
  Properly closing the clock and open GPIO pins
  Attempting to stop live clock

>>>
```

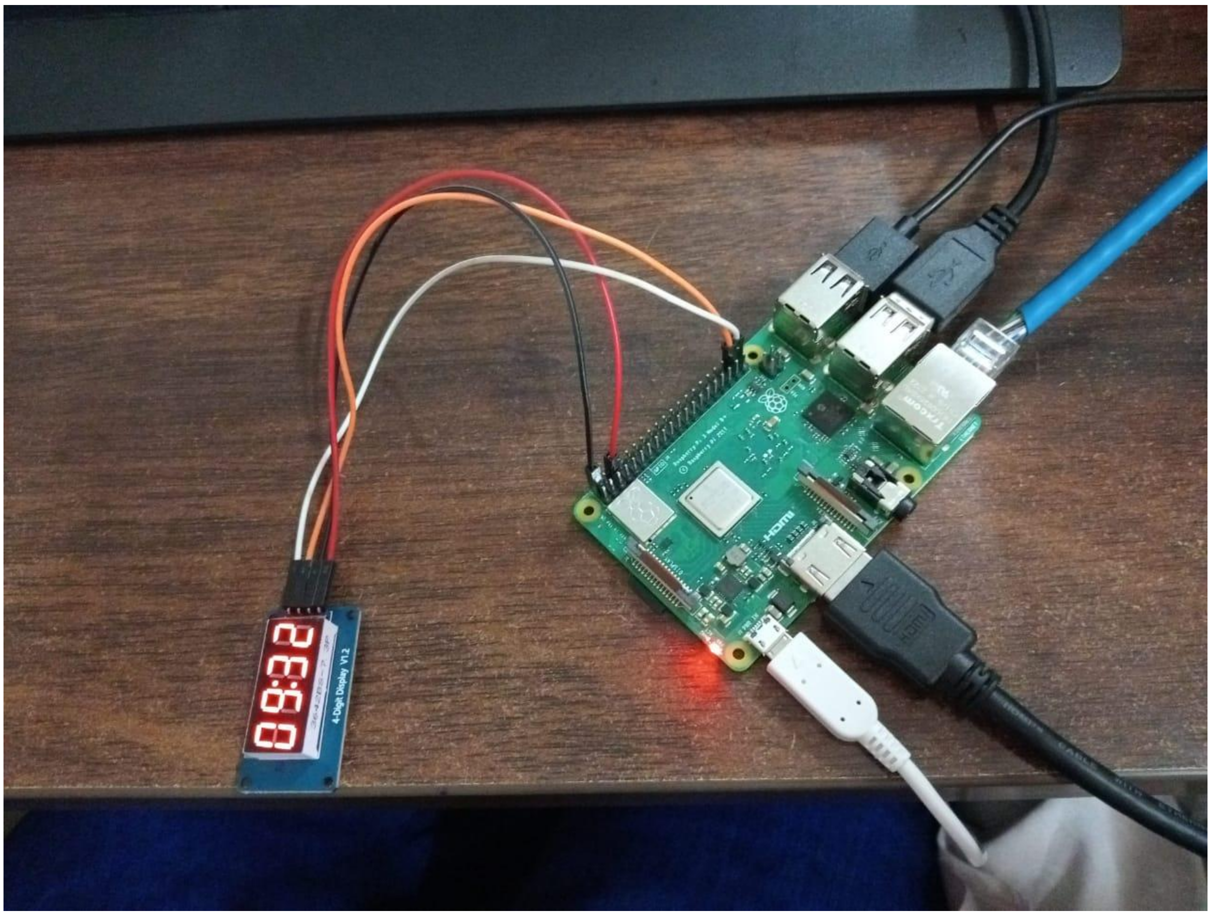


Fig: 4-digit 7 segment display interfacing with raspberry pi model 3b+

10.RESULT:

The project successfully displays the current time on a 4-digit 7-segment display using a Raspberry Pi. The time updates in real time, with a blinking colon every second. The system operates efficiently, using minimal GPIO pins, and the brightness control allows for adjustment in different environments.

11.CONCLUSION:

- ➢ Successfully demonstrated controlling a 4-digit 7-segment display using Raspberry Pi.
- ➢ Real-time timekeeping achieved with Python and the datetime module.
- ➢ Efficient hardware usage with minimal GPIO pins through the TM1637 driver and multiplexing.
- ➢ Modular code structure ensures easy future enhancements and adjustments.
- ➢ Provides a practical foundation for embedded system and IoT applications.

12.REFERENCES:

- “Displaying Time over 4-Digit 7-Segment Display Using Raspberry Pi". https://www.scribd.com/document/809070213/Practical-2-Displaying-Time-over-4-Digit-7-Segment-Display-Using-Raspberry-Pi
- "Getting Started with Raspberry Pi. " https://www.raspberrypi.org/documentation/
- TM1637 4-Digit Display Module Datasheet. https://www.electronicwings.com
- Python Software Foundation. "Python Documentation." https://docs.python.org/3/
- Adafruit. "TM1637 4-Digit Display Module with Raspberry Pi." https://learn.adafruit.com/tm1637-seven-segment-display
- https://github.com/timwaizenegger/raspberrypi-examples/tree/master/actor-led-7segment-4numbers