DS3231 RTC Module interface with Raspberry Pi

Introduction:

The **DS3231 Real-Time Clock (RTC) module** is an electronic component that helps keep track of time and date accurately, even when the main power is turned off. This is useful because the **Raspberry Pi does not have a built-in real-time clock**, meaning it loses track of time when it is powered down or restarted.

The DS1302 module solves this problem by using a small **coin cell battery (CR2032 3V)** as a backup power source, ensuring that time and date settings are maintained even when the Raspberry Pi is turned off.

The module communicates with the Raspberry Pi using a simple **Serial Peripheral Interface (SPI)-like protocol** with only **three data pins**—**Clock (CLK)**, **Data (DAT)**, and **Chip Enable (CE)**. This allows the Raspberry Pi to read and write time data easily. The DS1302 can store and manage details such as:

- **Year, Month, Date** (Calendar Information)
- **Day of the Week**
- **Hours, Minutes, and Seconds**

Since the Raspberry Pi relies on an internet connection to fetch the correct time, using the DS1302 RTC module is beneficial in situations where there is **no internet access** or when the device needs to keep time accurately over long periods. This module is widely used in projects involving **data logging, automation systems, security systems, and time-based controls.**

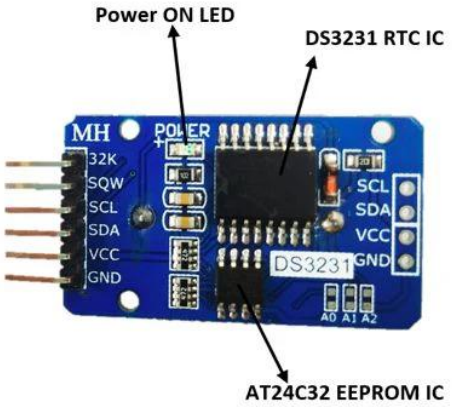

Fig 1. DS3231 RTC

Components Required:

- Raspberry Pi (any model with GPIO pins)
- DS1302 RTC module
- CR2032 3V coin cell battery
- Jumper wires
- Breadboard (optional)

Wiring the DS1302 RTC with Raspberry Pi:

The DS1302 uses three GPIO pins for communication: CE (Chip Enable), I/O (Data), and SCLK (Clock Signal). The pin configuration is given below:

DS1302 Pin	**Function**	**Raspberry Pi Pin**
VCC	Power (3.3V-5V)	3.3V (Pin 1) or 5V (Pin 2)
GND	Ground	GND (Pin 6)
CLK	Serial Clock	GPIO21 (Pin 40)
DAT	Data	GPIO20 (Pin 38)
RST (CE)	Clip Enable	GPIO16 (Pin 36)

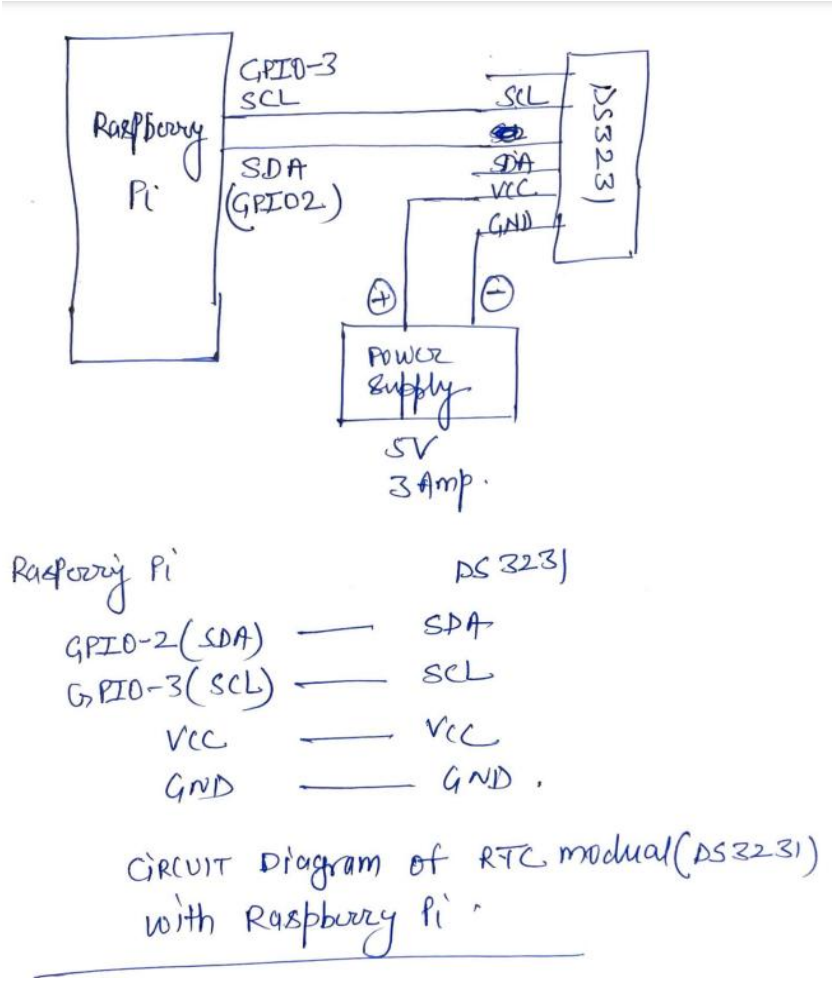

Fig 2. Connection of RTC with Raspberry Pi

Applications:

- Timekeeping for Embedded Systems – Maintains accurate time without internet access.
- Data Logging – Records timestamps for sensor data in IoT and monitoring systems.
- Power Outage Recovery – Retains time settings after a power failure.
- Digital Clocks & Displays – Shows real-time clock updates on LED/LCD screens.

Learnings:

- **Understanding RTC Modules:** Gained knowledge of how the DS1302 RTC module functions, maintaining accurate time independently of the Raspberry Pi's power supply using a backup battery.
- **Interfacing RTC with Raspberry Pi**: Learned how to establish communication between the DS1302 module and the Raspberry Pi using GPIO pins and a simple SPI-like protocol for data transfer.
- **Implementing Python Scripts for RTC:** Developed Python scripts to set, read, and update the date and time on the RTC module, enabling real-time tracking even after power loss.
- **Configuring Persistent Timekeeping**: Understood how the RTC module retains time settings when the Raspberry Pi is turned off and how to synchronize it with the system clock.
- **Troubleshooting Hardware and Software Issues:** Gained problem-solving experience by debugging wiring connections, incorrect time settings, and library installation issues, ensuring smooth integration with the Raspberry Pi.

Conclusion:

This project successfully integrated the **DS1302 RTC module** with a **Raspberry Pi** for accurate, power-independent timekeeping. We learned how to interface hardware, write Python scripts to manage time, and troubleshoot common issues. This system is useful for **real-time applications** like **data logging and automation** and can be enhanced with an LED display for better visibility.

Outcome:

Fig 3. Output on the LED Screen

Program:

```
import smbus
import time
from datetime import datetime
from RPLCD.i2c import CharLCD
lcd = CharLCD('PCF8574', 0x27)

bus = smbus.SMBus(1)
RTC_ADDRESS = 0x68

lcd.clear()
lcd.write_string("RTC Clock Ready")

bus = smbus.SMBus(1)
RTC_ADDRESS = 0x68

def dec_to_bcd(value):
    return (value // 10) * 16 + (value % 10)

def set_time():
    now = datetime.now()
    seconds = dec_to_bcd(now.second)
```

```
    minutes = dec_to_bcd(now.minute)
    hours = dec_to_bcd(now.hour)
    day = dec_to_bcd(now.day)
    month = dec_to_bcd(now.month)
    year = dec_to_bcd(now.year - 2000)
    weekday = dec_to_bcd(now.isoweekday())  # Monday = 1, Sunday = 7

    data = [seconds, minutes, hours, weekday, day, month, year]
    bus.write_i2c_block_data(RTC_ADDRESS, 0x00, data)
    print(f"RTC Time Set to: {now.strftime('%Y-%m-%d %H:%M:%S')} | Day of the week:
{now.strftime('%A')}")

def bcd_to_dec(value):
    return (value // 16) * 10 + (value % 16)

def read_time():
    data = bus.read_i2c_block_data(RTC_ADDRESS, 0x00, 7)
    seconds = bcd_to_dec(data[0])
    minutes = bcd_to_dec(data[1])
    hours = bcd_to_dec(data[2])
    weekday = bcd_to_dec(data[3])  # 1 = Monday, 7 = Sunday
    day = bcd_to_dec(data[4])
    month = bcd_to_dec(data[5])
    year = bcd_to_dec(data[6]) + 2000

    weekdays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
    print(f"RTC Time: {year}-{month:02d}-{day:02d} {hours:02d}:{minutes:02d}:{seconds:02d} | Day:
{weekdays[weekday-1]}")

    lcd.clear()

    lcd.cursor_pos = (0, 0)  # First row
```

```
    lcd.write_string(f'RTC DS3231')

    lcd.cursor_pos = (1, 0)  # First row
    lcd.write_string(f'{year}-{month:02d}-{day:02d}')

    lcd.cursor_pos = (2, 0)  # Second row
    lcd.write_string(f'{hours:02d}:{minutes:02d}:{seconds:02d}')

    lcd.cursor_pos = (3, 0)  # Third row
    lcd.write_string(f"Day: {weekdays[weekday-1]}")



while True:
    read_time()
    time.sleep(1)

#read_time()
#set_time()
```