# Interface Micro SD Card Module with Raspberry Pi Pico

In this user guide, we will learn how to interface a micro SD card with Raspberry Pi Pico using the microSD card module. This module provides an SPI interface to connect an SD card module with any microcontroller which supports the SPI communication interface. Using a micro SD card becomes very handy for applications where we need to store files or any data.

Additionally, we will learn how to handle files in the microSD card including reading/writing to a file. Storage data including text, video, audio, CSV, HTML, JavaScript, and CSS files can all be conveniently stored in the microSD card. It is one of the most reliable and practical ways to store data in devices such as mobile phones, laptops, and personal computers.

## Prerequisites

Before we start this lesson, make sure you are familiar with and have the latest version of Python3 installed in your system and set up MicroPython in your Raspberry Pi Pico. Additionally, you should have a running Integrated Development Environment(IDE) to do the programming. We will be using the same Thonny IDE as we have done previously when we learned how to blink and chase LEDs in MicroPython here:



- Getting Started with Raspberry Pi Pico using Thonny IDE

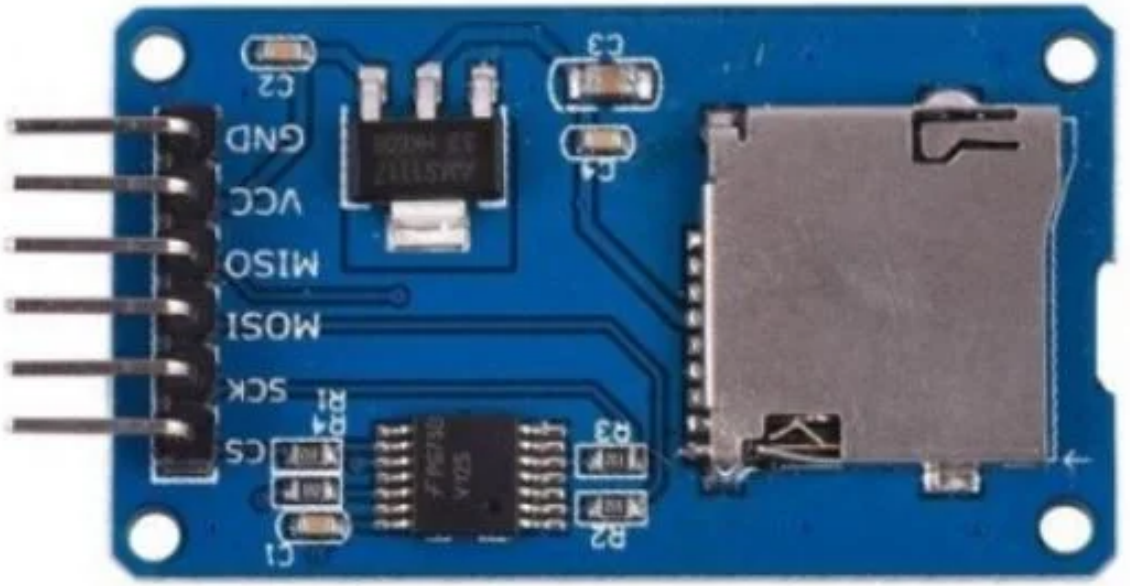
If you are using uPyCraft IDE, you can check this getting started guide:

- Getting Started with Raspberry Pi Pico using uPyCraft IDE

## MicroSD Card Module Introduction

The microSD card Modules are designed to communicate with the MicroSD cards. These connectors provide the required hardware and pinout to connect SD cards with microcontrollers such as ESP32, Arduino, ESP8266, Raspberry Pi, etc. Although, they are compatible with almost all SD cards which are commonly used in cell phones. But they can handle a maximum of 16GB capacity microSD cards and only 2GB capacity for standard SD cards.

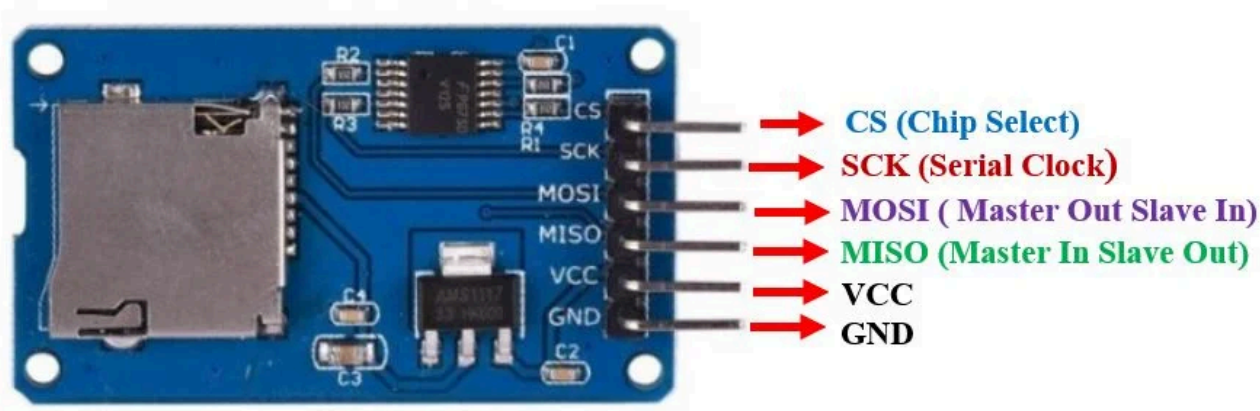
With the help of these modules, we will be able to read and write data to and from SD cards through the SPI communication protocol. There are several different types of microSD card modules easily available in the market. But, the one which we will be using in this article is shown below:




MicroSD card module

## Pinout

This microSD card module has 6 terminals consisting of SPI and power supply terminals. Below you can view the pinout of this module with some description of the individual pins.




Pinout of MicroSD card Module

| Pin Name | Description |
| --- | --- |
| GND | This is the ground pin which should be connected with the ground pin of the microcontroller. |

| Pin Name | Description |
|---|---|
| VCC | This pin supplies power to the module. The power supply of ~4.5V-5.5V. The adapter consists of a 3.3V voltage regulator circuit as well. It is connected with 5V pin of the microcontroller. |
| CS | This is the Chip Select pin for SPI communication. |
| MOSI | This is called the 'Master Out Slave In.' It is used as the SPI input to the module. |
| SCK | This is called the 'Serial Clock' pin which is used in SPI serial clock output. |
| MISO | This is called the 'Master in Slave Out.' It is used as the SPI output from the module. |

Table showing the pin names and their short descriptions

## MicroSD card module Interfacing with Raspberry Pi Pico

Required Hardware:

Following components are required:

- 1 x Raspberry Pi Pico board
- MicroSD card
- 1 x MicroSD card module

Let us first learn about the Raspberry Pi Pico SPI interface.

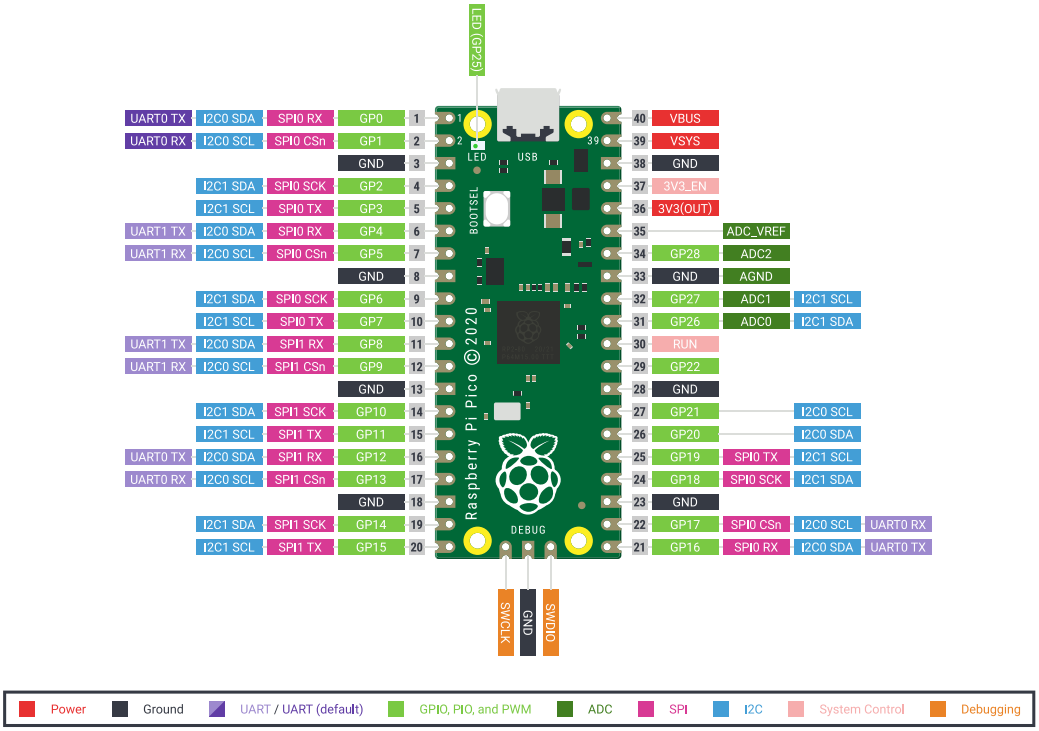
## Raspberry Pi Pico SPI Pins

Raspberry Pi Pico supports two SPI peripherals. Both SPI module pins are accessible through GPIO pins of Raspberry Pi Pico. The following table

shows the connection of GPIO pins with both SPI modules. Each connection of SPI controller pins can be configured through multiple GPIO pins as shown in the figure. But before using SPI, you should configure in software which GPIO pins you want to use with a specific SP peripheral.

| SPI Controller | GPIO Pins |
|---|---|
| SPI0_RX | GP0/GP4/GP16 |
| SPI0_TX | GP3/GP7/GP19 |
| SPI0_SCK | GP2/GP6/GP18 |
| SPI0_CSn | GP1/GP5/GP17 |
| SPI1_RX | GP8/GP12 |
| SPI1_TX | GP11/GP15 |
| SPI1_SCK | GP10/GP14 |
| SPI1_CSn | GP9/GP13 |

The figure below shows the SPI pins of Raspberry Pi Pico.




Each SPI controller supports master and slave mode and are compatible with the following four modes:

- Motorola SPI Interface
- TI Serial Interface
- National Semiconductor Serial Interface

It has 8 buffers for each transmitter and receiver of the SPI controller. Moreover, it can also be driven with interrupt or DMA.

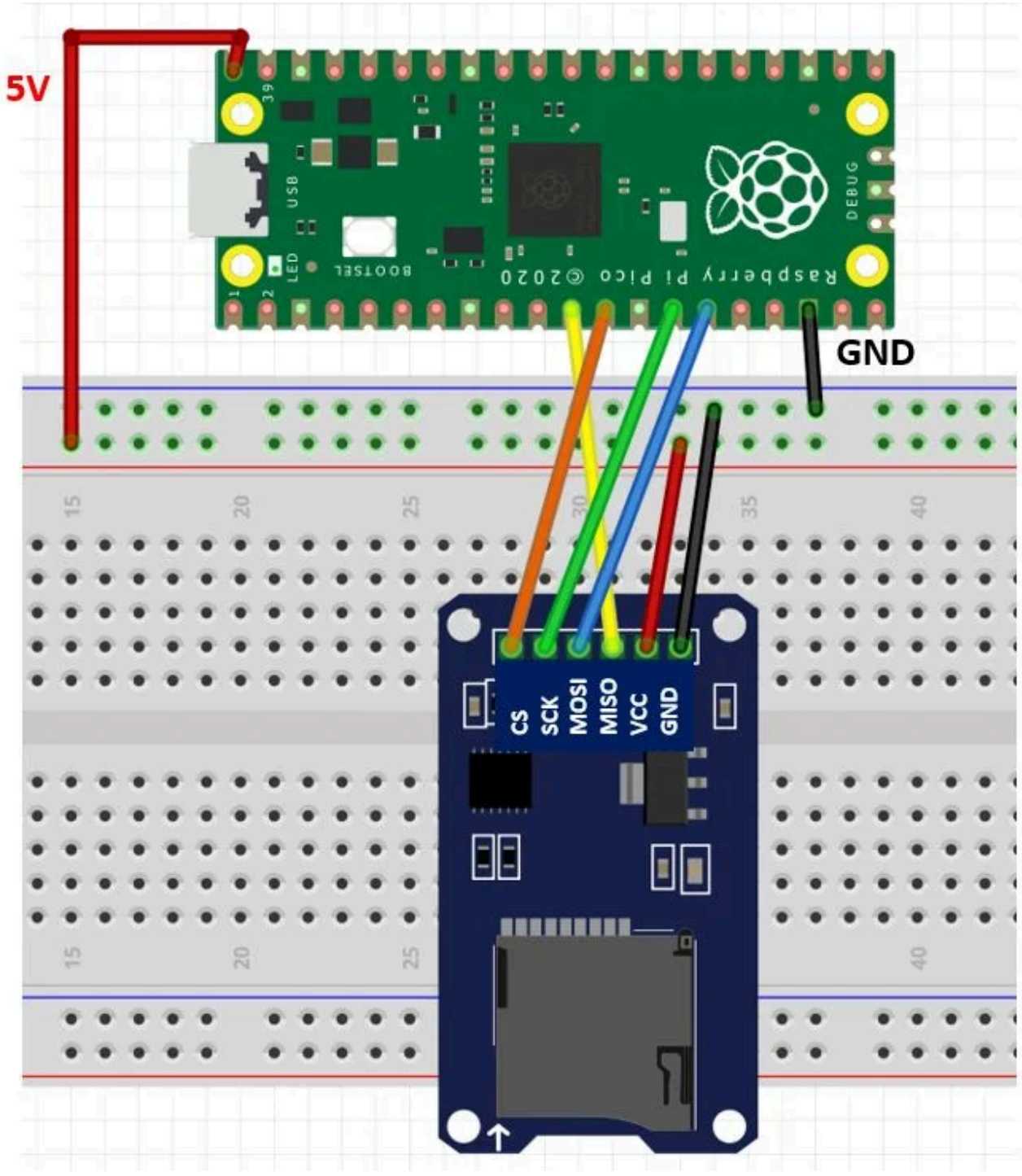
## Schematic Diagram

Now let us see how to connect the microSD card module and Raspberry Pi Pico. The table below shows the connections between the two devices:

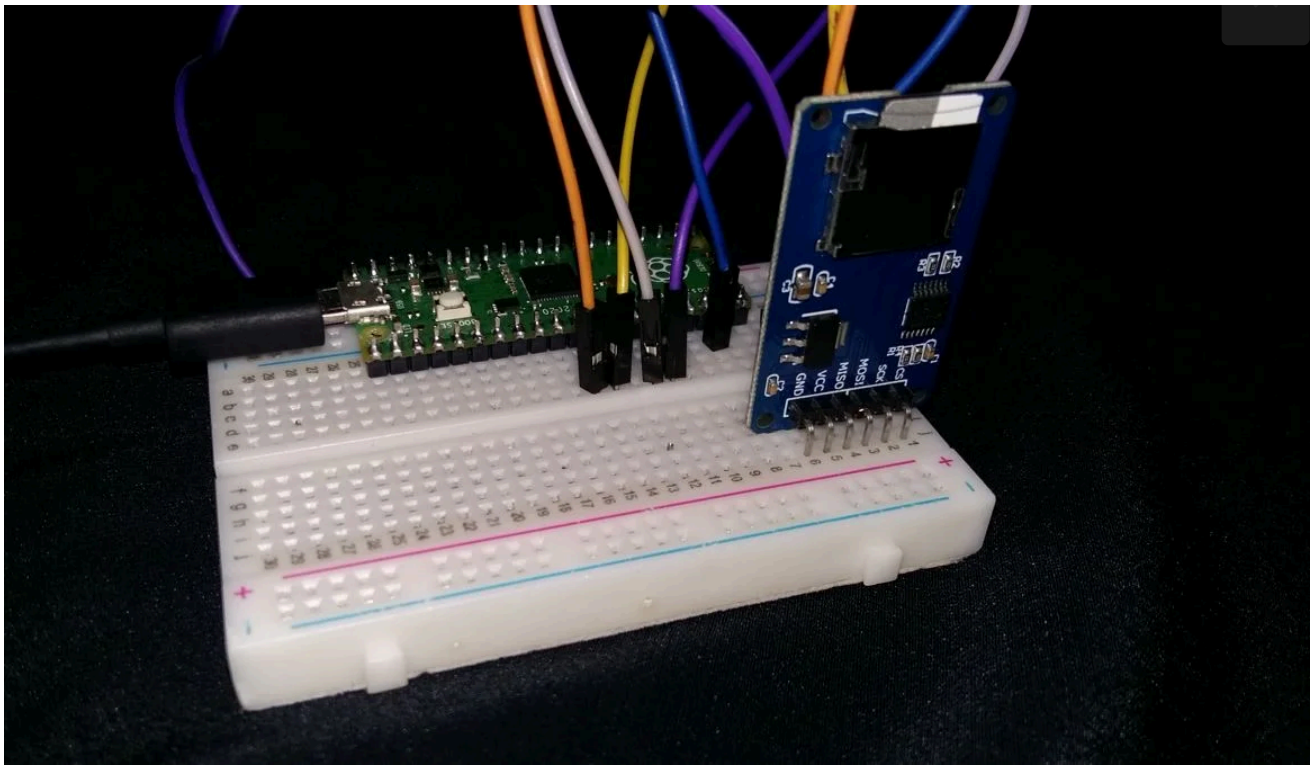
| MicroSD card module | Raspberry Pi Pico |
|---|---|
| GND | GND |
| VCC | 5V |
| CS | GP9 (SPI1_CSn) |

| MicroSD card module | Raspberry Pi Pico |
| --- | --- |
| MOSI | GP11 (SPI1_TX) |
| SCK | GP10 (SPI1_SCK) |
| MISO | GP8 (SPI1_RX) |

As shown from the table, we will connect the VCC terminal of the MicroSD card module with 5V pin of Raspberry Pi Pico. Both grounds will be common. We have used the same connections for SPI pins as specified in the table above. However you can use other combinations of SPI pins as well.




Raspberry Pi Pico with microSD card module connection diagram

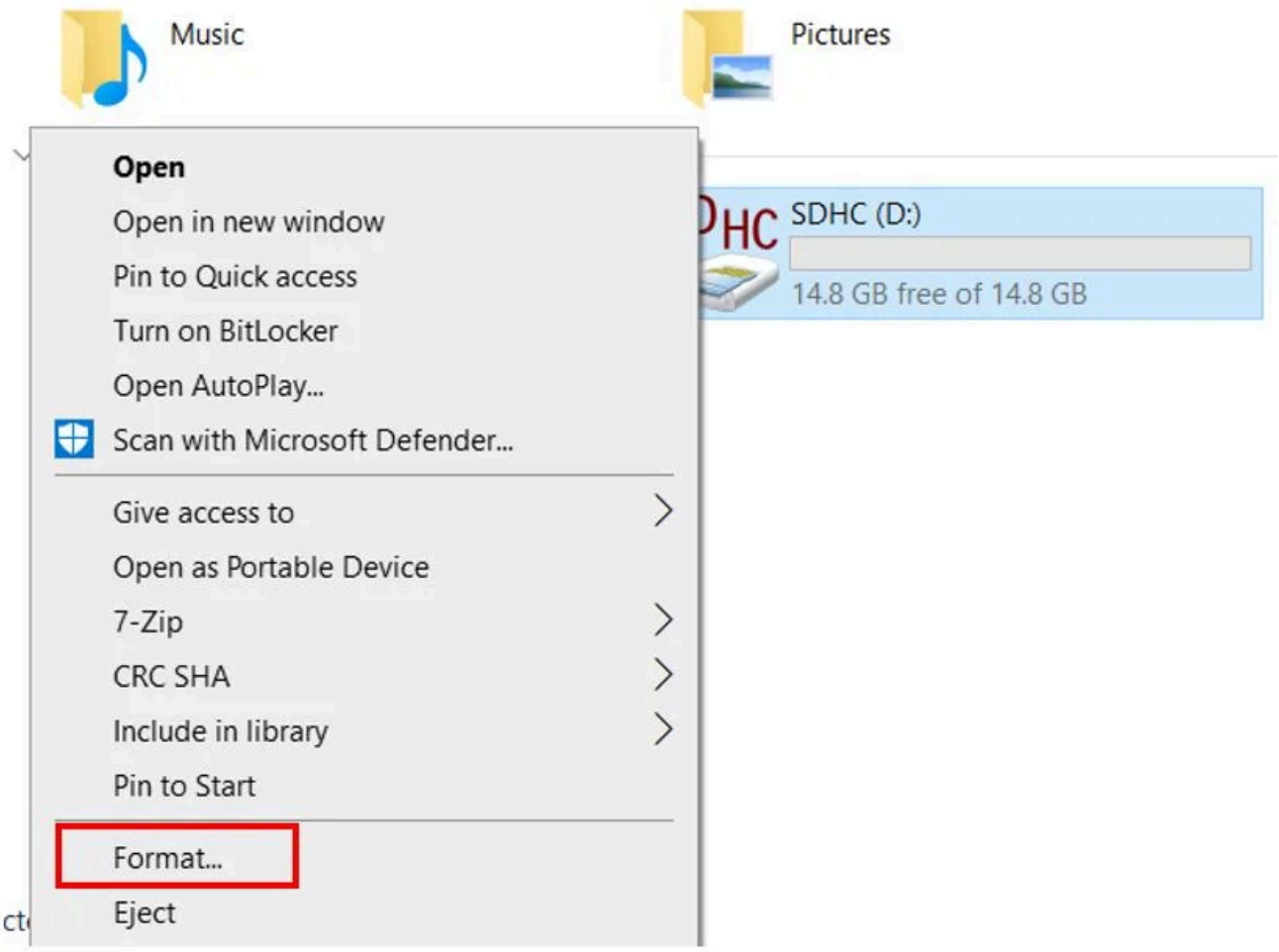



Now, as we know how to interface the microSD card module and Raspberry Pi Pico together let us learn how to prepare the microSD card to handle files in Thonny IDE using MicroPython.

## Formatting the MicroSD card

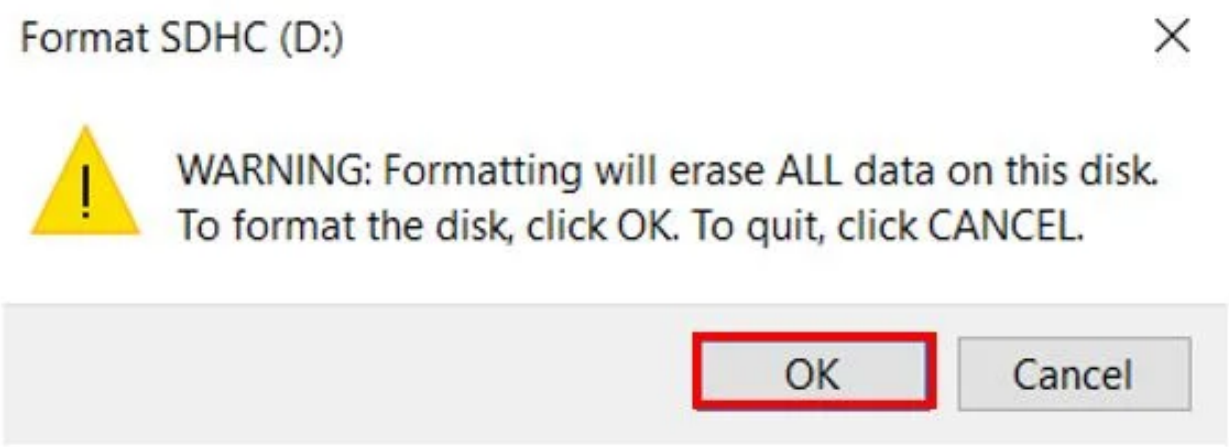
As we have to use our microSD card with Raspberry Pi Pico, so we would have to format it as FAT32. We will have to follow a series of steps to accomplish it successfully.

- First, insert your microSD card in your laptop/computer. Now go to ‘This PC’ and click on SD card icon. Then click on Format by right clicking the SD card icon.

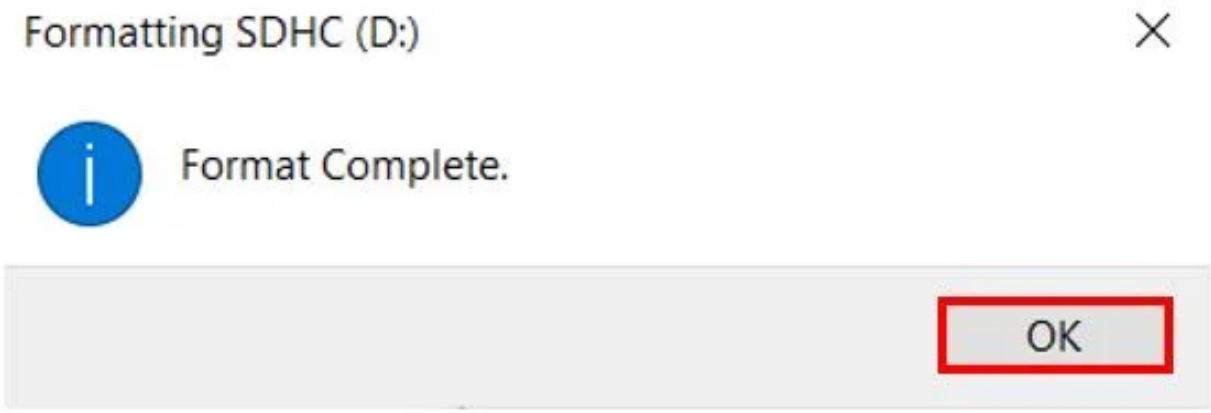



- The following window will appear. Select FAT32 from the dialog box of ‘File System’ and click on ‘START.’


- You will receive a warning message that formatting will erase all previous data saved on the microSD card. Click ‘OK.’




- After a few moments, your microSD card will be formatted successfully. Click 'OK.'




## Installing SD Card Library

For this project we will require sdcard.py library. Copy this library and save it in your Raspberry Pi Pico with the respective file name from the GitHub link.

Open a new file in Thonny. Copy the library given below or from the link given above. Save it to Raspberry Pi Pico with the name **sdcard.py** under the lib folder.

**sdcard.py**

```
"""
MicroPython driver for SD cards using SPI bus.

Requires an SPI bus and a CS pin.  Provides readblocks and writeblocks
methods so the device can be mounted as a filesystem.

Example usage on pyboard:

  import pyb, sdcard, os
  sd = sdcard.SDCard(pyb.SPI(1), pyb.Pin.board.X5)
  pyb.mount(sd, '/sd2')
  os.listdir('/')

Example usage on ESP8266:

  import machine, sdcard, os
  sd = sdcard.SDCard(machine.SPI(1), machine.Pin(15))
  os.mount(sd, '/sd')
  os.listdir('/')

"""

from micropython import const
import time


_CMD_TIMEOUT = const(100)

_R1_IDLE_STATE = const(1 << 0)
# R1_ERASE_RESET = const(1 << 1)
_R1_ILLEGAL_COMMAND = const(1 << 2)
# R1_COM_CRC_ERROR = const(1 << 3)
# R1_ERASE_SEQUENCE_ERROR = const(1 << 4)
# R1_ADDRESS_ERROR = const(1 << 5)
# R1_PARAMETER_ERROR = const(1 << 6)
_TOKEN_CMD25 = const(0xFC)
_TOKEN_STOP_TRAN = const(0xFD)
_TOKEN_DATA = const(0xFE)


class SDCard:
  def __init__(self, spi, cs, baudrate=1320000):
    self.spi = spi
    self.cs = cs

    self.cmdbuf = bytearray(6)
```

```
        self.dummybuf = bytearray(512)
        self.tokenbuf = bytearray(1)
        for i in range(512):
            self.dummybuf[i] = 0xFF
        self.dummybuf_memoryview = memoryview(self.dummybuf)

        # initialise the card
        self.init_card(baudrate)

    def init_spi(self, baudrate):
        try:
            master = self.spi.MASTER
        except AttributeError:
            # on ESP8266
            self.spi.init(baudrate=baudrate, phase=0, polarity=0)
        else:
            # on pyboard
            self.spi.init(master, baudrate=baudrate, phase=0, polarity=0)

    def init_card(self, baudrate):

        # init CS pin
        self.cs.init(self.cs.OUT, value=1)

        # init SPI bus; use low data rate for initialisation
        self.init_spi(100000)

        # clock card at least 100 cycles with cs high
        for i in range(16):
            self.spi.write(b"\xff")

        # CMD0: init card; should return _R1_IDLE_STATE (allow 5 attempts)
        for _ in range(5):
            if self.cmd(0, 0, 0x95) == _R1_IDLE_STATE:
                break
        else:
            raise OSError("no SD card")

        # CMD8: determine card version
        r = self.cmd(8, 0x01AA, 0x87, 4)
        if r == _R1_IDLE_STATE:
            self.init_card_v2()
        elif r == (_R1_IDLE_STATE | _R1_ILLEGAL_COMMAND):
            self.init_card_v1()
        else:
            raise OSError("couldn't determine SD card version")
```

```
    # get the number of sectors
    # CMD9: response R2 (R1 byte + 16-byte block read)
    if self.cmd(9, 0, 0, 0, False) != 0:
        raise OSError("no response from SD card")
    csd = bytearray(16)
    self.readinto(csd)
    if csd[0] & 0xC0 == 0x40:  # CSD version 2.0
        self.sectors = ((csd[8] << 8 | csd[9]) + 1) * 1024
    elif csd[0] & 0xC0 == 0x00:  # CSD version 1.0 (old, <=2GB)
        c_size = csd[6] & 0b11 | csd[7] << 2 | (csd[8] & 0b11000000) << 4
        c_size_mult = ((csd[9] & 0b11) << 1) | csd[10] >> 7
        self.sectors = (c_size + 1) * (2 ** (c_size_mult + 2))
    else:
        raise OSError("SD card CSD format not supported")
    # print('sectors', self.sectors)

    # CMD16: set block length to 512 bytes
    if self.cmd(16, 512, 0) != 0:
        raise OSError("can't set 512 block size")

    # set to high data rate now that it's initialised
    self.init_spi(baudrate)

def init_card_v1(self):
    for i in range(_CMD_TIMEOUT):
        self.cmd(55, 0, 0)
        if self.cmd(41, 0, 0) == 0:
            self.cdv = 512
            # print("[SDCard] v1 card")
            return
    raise OSError("timeout waiting for v1 card")

def init_card_v2(self):
    for i in range(_CMD_TIMEOUT):
        time.sleep_ms(50)
        self.cmd(58, 0, 0, 4)
        self.cmd(55, 0, 0)
        if self.cmd(41, 0x40000000, 0) == 0:
            self.cmd(58, 0, 0, 4)
            self.cdv = 1
            # print("[SDCard] v2 card")
            return
    raise OSError("timeout waiting for v2 card")

def cmd(self, cmd, arg, crc, final=0, release=True, skip1=False):
    self.cs(0)
```

```
        # create and send the command
        buf = self.cmdbuf
        buf[0] = 0x40 | cmd
        buf[1] = arg >> 24
        buf[2] = arg >> 16
        buf[3] = arg >> 8
        buf[4] = arg
        buf[5] = crc
        self.spi.write(buf)

        if skip1:
            self.spi.readinto(self.tokenbuf, 0xFF)

        # wait for the response (response[7] == 0)
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            response = self.tokenbuf[0]
            if not (response & 0x80):
                # this could be a big-endian integer that we are getting here
                for j in range(final):
                    self.spi.write(b"\xff")
                if release:
                    self.cs(1)
                    self.spi.write(b"\xff")
                return response

        # timeout
        self.cs(1)
        self.spi.write(b"\xff")
        return -1

    def readinto(self, buf):
        self.cs(0)

        # read until start byte (0xff)
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            if self.tokenbuf[0] == _TOKEN_DATA:
                break
            time.sleep_ms(1)
        else:
            self.cs(1)
            raise OSError("timeout waiting for response")

        # read data
        mv = self.dummybuf_memoryview
        if len(buf) != len(mv):
```

```
            mv = mv[: len(buf)]
        self.spi.write_readinto(mv, buf)

        # read checksum
        self.spi.write(b"\xff")
        self.spi.write(b"\xff")

        self.cs(1)
        self.spi.write(b"\xff")

    def write(self, token, buf):
        self.cs(0)

        # send: start of block, data, checksum
        self.spi.read(1, token)
        self.spi.write(buf)
        self.spi.write(b"\xff")
        self.spi.write(b"\xff")

        # check the response
        if (self.spi.read(1, 0xFF)[0] & 0x1F) != 0x05:
            self.cs(1)
            self.spi.write(b"\xff")
            return

        # wait for write to finish
        while self.spi.read(1, 0xFF)[0] == 0:
            pass

        self.cs(1)
        self.spi.write(b"\xff")

    def write_token(self, token):
        self.cs(0)
        self.spi.read(1, token)
        self.spi.write(b"\xff")
        # wait for write to finish
        while self.spi.read(1, 0xFF)[0] == 0x00:
            pass

        self.cs(1)
        self.spi.write(b"\xff")

    def readblocks(self, block_num, buf):
        nblocks = len(buf) // 512
        assert nblocks and not len(buf) % 512, "Buffer length is invalid"
        if nblocks == 1:
```

```
            # CMD17: set read address for single block
            if self.cmd(17, block_num * self.cdv, 0, release=False) != 0:
                # release the card
                self.cs(1)
                raise OSError(5)  # EIO
            # receive the data and release card
            self.readinto(buf)
        else:
            # CMD18: set read address for multiple blocks
            if self.cmd(18, block_num * self.cdv, 0, release=False) != 0:
                # release the card
                self.cs(1)
                raise OSError(5)  # EIO
            offset = 0
            mv = memoryview(buf)
            while nblocks:
                # receive the data and release card
                self.readinto(mv[offset : offset + 512])
                offset += 512
                nblocks -= 1
            if self.cmd(12, 0, 0xFF, skip1=True):
                raise OSError(5)  # EIO

    def writeblocks(self, block_num, buf):
        nblocks, err = divmod(len(buf), 512)
        assert nblocks and not err, "Buffer length is invalid"
        if nblocks == 1:
            # CMD24: set write address for single block
            if self.cmd(24, block_num * self.cdv, 0) != 0:
                raise OSError(5)  # EIO

            # send the data
            self.write(_TOKEN_DATA, buf)
        else:
            # CMD25: set write address for first block
            if self.cmd(25, block_num * self.cdv, 0) != 0:
                raise OSError(5)  # EIO
            # send the data
            offset = 0
            mv = memoryview(buf)
            while nblocks:
                self.write(_TOKEN_CMD25, mv[offset : offset + 512])
                offset += 512
                nblocks -= 1
            self.write_token(_TOKEN_STOP_TRAN)

    def ioctl(self, op, arg):
```

```
        if op == 4:  # get number of blocks
            return self.sectors
```

## MicroPython Sketch: Read/Write on a File

Open your Thonny IDE and go to **File > New** to open a new file. Copy the following code in that file. This sketch will read and write data on a .txt file that will get saved on our microSD card.

```
import machine
import sdcard
import uos


CS = machine.Pin(9, machine.Pin.OUT)
spi =
machine.SPI(1,baudrate=1000000,polarity=0,phase=0,bits=8,firstbit=machine.S
PI.MSB,sck=machine.Pin(10),mosi=machine.Pin(11),miso=machine.Pin(8))

sd = sdcard.SDCard(spi,CS)

vfs = uos.VfsFat(sd)
uos.mount(vfs, "/sd")

# Create a file and write something to it
with open("/sd/data.txt", "w") as file:
    print("Writing to data.txt...")
    file.write("Welcome to microcontrollerslab!\r\n")
    file.write("This is a test\r\n")

# Open the file we just created and read from it
with open("/sd/data.txt", "r") as file:
    print("Reading data.txt...")
    data = file.read()
    print(data)
```

## How the Code Works?

Now let us understand how each part of the code works.

### Importing Libraries

The first step is to include all the libraries that are necessary for this project. We will import machine, sdcard and uos modules required for this task.

```
import machine
import sdcard
import uos
```

Configure the GPIO pin connected with the CS pin as an output pin. This is done by using the Pin() method and passing the GPIO number as the first parameter and Pin.OUT as the second parameter.

```
CS = machine.Pin(9, machine.Pin.OUT)
```

The next step is to initialize the SPI interface with the required parameters including the SPI channel number, baud rate, SPI pins etc:

```
spi =
machine.SPI(1,baudrate=1000000,polarity=0,phase=0,bits=8,firstbit=machine.S
PI.MSB,sck=machine.Pin(10),mosi=machine.Pin(11),miso=machine.Pin(8))
```

## Initializing the microSD card

The following line of code will initialize the microSD card using the SDcard() function on the sdcard library. The function takes in the spi object as the first parameter and the CS Pin as an second parameter. Thus, it will start the SPI communication at the stated parameters.

```
sd = sdcard.SDCard(spi,CS)
```

Next, we will mount the filesystem using the following lines of code:

```
vfs = uos.VfsFat(sd)
uos.mount(vfs, "/sd")
```

## Writing data to file

Next, we will open the data.txt file on the microSD card using open() and specify the first parameter as the file name and second parameter ‘w’ indicating that we want to write on the file. If the file does not exist, it will get created.

In the Thonny shell terminal we will print “Writing to data.txt…”

```
with open("/sd/data.txt", "w") as file:
  print("Writing to data.txt...")
```

Using the write() method on the file object, we will write ‘Welcome to microcontrollerlab!’ in the data.txt file. Next, we will write “This is a test” on the second line in the file.

```
  file.write("Welcome to microcontrollerslab!\r\n")
  file.write("This is a test\r\n")
```

## Reading data from file

Now, we will read the data which we just wrote on our data.txt file. To do that we will first open the data.txt file by using open() method and specify the file name as the first parameter and ‘r’ as the second parameter indicating that we want to read from the file. Then by using read() method on the file object, we will read the data.txt file and print the data in the serial monitor.

```
with open("/sd/data.txt", "r") as file:
  print("Reading data.txt...")
```

```
data = file.read()
print(data)
```

## Demonstration

Upload the above code as main.py file to Raspberry Pi Pico. You will view the messages in the Thonny shell terminal that we are writing to the data.txt file and then reading from it. The data written to the file (highlighted in red) can also be seen printed on the terminal:


Take the microSD card out of the module and insert it in your system to view the data.txt file.

Open the data.txt file. Inside it you will be able to view the two messages that we wrote to the file:

## Conclusion

In conclusion, we learned how to use a micro SD card with Raspberry Pi Pico. We successfully initialized our microSD card and were able to read and write to a .txt file.

You may like to read these SD card guides for Arduino and sensors:

- BME280 Data Logger with Arduino and Micro SD Card
- DHT22 Data Logger with Arduino and Micro SD Card
- GPS Data Logger with Arduino and Micro SD Card – GPS Tracker

SD card interfacing with other development boards:

- ESP32 Web Server Hosting Files from Micro SD card (Arduino IDE)
- MicroSD Card Module with ESP32 using Arduino IDE
- ESP32 Data Logging Temperature Sensor Readings to microSD card (Arduino IDE)

Raspberry Pi Pico

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Email Address

SUBSCRIBE

## 1 thought on “Interface Micro SD Card Module with Raspberry Pi Pico”

**Vic**
June 2, 2022 at 10:42 am

>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
File “”, line 12, in
OSError: [Errno 19] ENODEV
>>>

Code (sdcard.py and runtime) multiple checks.

2G micro sd card….

Multiple modules(hardware)…
Multiple wiring checks… All good (even replace wires a time or two)
Mutiple sd card partitioning, formatting fat32 and fat16

Card reads fine on win and linux boxes.
Voltage to card checks to sd card module pins(all of those tried).

Same error message every time.
Code recognizes when the sd card is pulled.

Web search no help :-(.

Any thoughts appreciated.

Thanks MUCH!

Reply

## Leave a Comment

Name *

Email *

Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

POST COMMENT



## Subscribe to Blog via Email

Email Address

SUBSCRIBE

## Categories

Select Category

## Recent Posts

DS18B20 Sensor with STM32 Nucleo using STM32CubeIDE

DHT22 Sensor with STM32 Nucleo using STM32CubeIDE

BME280 Sensor with STM32 Nucleo using STM32CubeIDE

I2C LCD with STM32 Nucleo using STM32CubeIDE

HC-SR04 Ultrasonic Sensor with STM32 Nucleo using STM32CubeIDE