

Hướng dẫn giao tiếp STM32 và thẻ nhớ SD sử dụng giao thức SDIO

Nhóm tác giả

Trần Hoàng Minh

Nguyễn Thị Tâm

Võ Văn Bửu

Lương Như Quỳnh

Hướng dẫn và kiểm tra

Nguyễn Huỳnh Nhật Thương

Nguyễn Quang Phương

Contents

A. SDIO	1
I. Tổng quan về giao tiếp SDIO.	1
1. Thẻ nhớ SD là gì? Mối liên hệ giữa thẻ SD và giao thức SDIO.....	1
2. Giao thức SDIO	3
II. SDIO trong STM32.....	3
1. Khởi SDIO trong STM32F411.	3
2. Cơ chế hoạt động.	4
3. Tổng quan về thư viện FATFs.	8
4. Sơ đồ kết nối phần cứng.	9
5. Cấu hình CubeMX.	12
6. Mã code.....	25
7. Kết quả	31
8. Một số lỗi	34

A. SDIO

I. Tổng quan về giao tiếp SDIO.

1. Thẻ nhớ SD là gì? Mối liên hệ giữa thẻ SD và giao thức SDIO.

- Thẻ nhớ SD (Secure Digital) là định dạng thẻ nhớ bộ nhớ được phát triển bởi SD Association để đáp ứng các yêu cầu về bảo mật, dung lượng, hiệu suất và môi trường vốn có trong các thiết bị điện tử tiêu dùng âm thanh và video. Thẻ SD sử dụng hệ thống tệp FAT32.
- Ngoài thẻ nhớ SD, còn có thẻ SD I/O (SDIO) là những thẻ không phải để lưu trữ, nhưng có giao diện với thiết bị ngoại vi. Bằng cách này, có thể tích hợp với GPS, Camera,

Wifi, Ethernet, máy đọc mã vạch, Bluetooth và được sử dụng rộng rãi trong lĩnh vực chế tạo rô bốt, đặc biệt là để tích hợp giao tiếp và định vị vào rô bốt.

- Thẻ nhớ SD được phát triển để tương thích được giống với thẻ MultiMedia (MMC), cho nên trong hầu hết trường hợp, các thiết bị hỗ trợ thẻ MMC cũng có thể sử dụng thẻ SD. Ngoài ra còn có các phiên bản kích thước nhỏ hơn như RS-MMC, miniSD và microSD với chức năng tương tự.
- microSD là loại thẻ nhớ được sử dụng nhiều nhất hiện nay với kích thước 15 x 11 x 1 mm. Trong bài viết này, chúng mình sẽ sử dụng thẻ microSD thì tính phổ biến của nó.

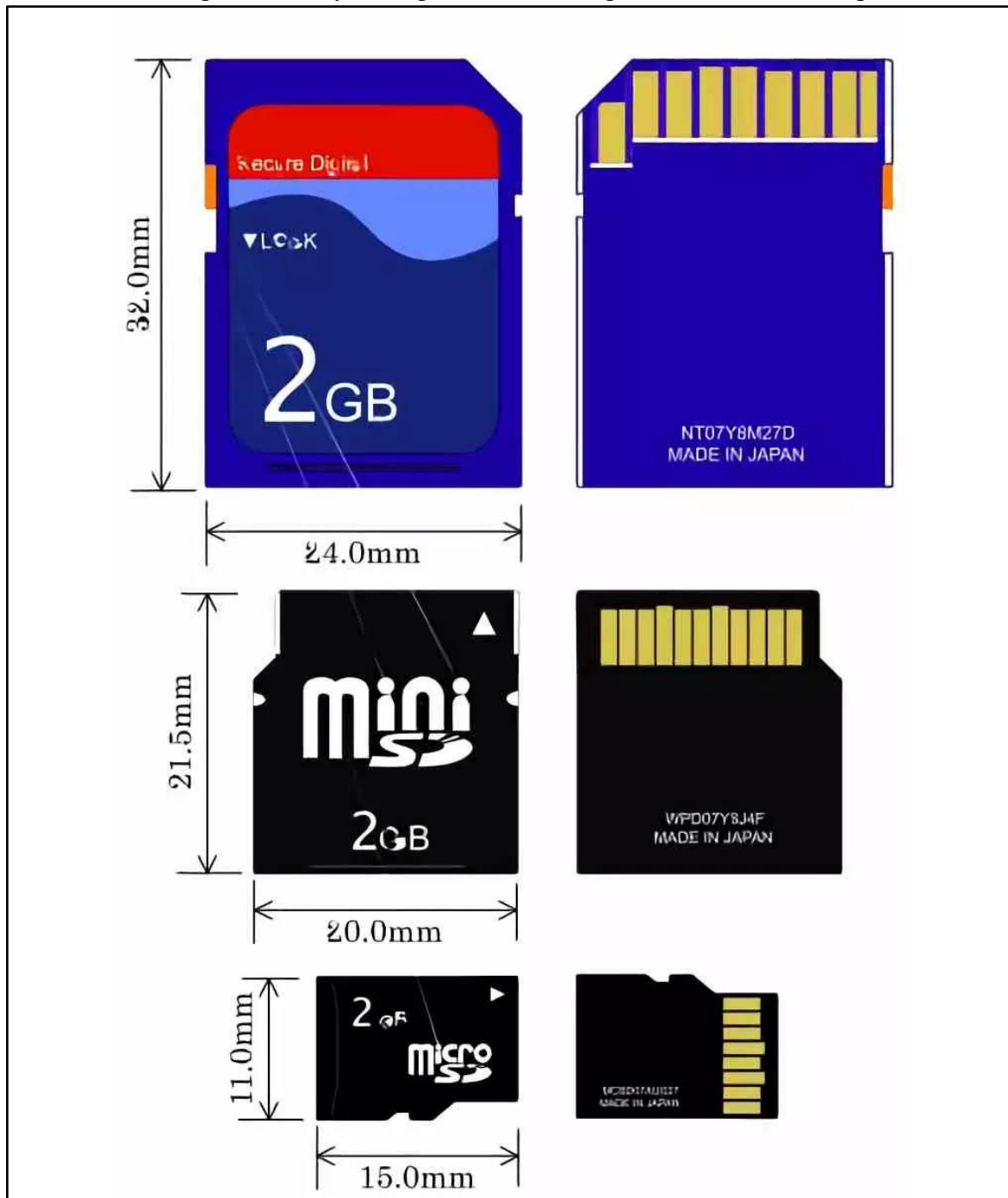


Figure 1: Các loại thẻ SD.

2. Giao thức SDIO

Như đã nói ở trên, SDIO (Secure Digital Input/ Output) được phát triển và nâng cấp từ giao thức dành cho thẻ SD. Mục đích của giao tiếp thẻ SDIO là cung cấp giao tiếp dữ liệu I/O tốc độ cao với mức tiêu thụ điện năng thấp cho các thiết bị điện tử.

Một số tính năng của giao thức SDIO:

- Được nhắm đến cho các ứng dụng di động và cố định.
- Yêu cầu thay đổi tối thiểu hoặc không thay đổi gì đến bus vật lý SD.
- Thay đổi tối thiểu đối với phần mềm trình điều khiển bộ nhớ.
- Có sẵn hình dạng vật lý mở rộng cho các ứng dụng chuyên dụng.
- Hỗ trợ Plug and Push (PnP).
- Hỗ trợ đa chức năng bao gồm nhiều I/O và kết hợp I/O và bộ nhớ.
- Hỗ trợ lên đến 7 chức năng I/O cộng với một bộ nhớ trên một thẻ.
- Cho phép thẻ ngắt kết nối máy chủ.
- Dải điện áp hoạt động: 2.7-3.6V.
- Thông số ứng dụng cho các chức năng SDIO tiêu chuẩn.

SDIO dùng để giao tiếp giữa Bus ngoại vi APB2 và MultiMediaCards(MMC) , thẻ nhớ SD , SDIO và thiết bị CE-ATA.

II. SDIO trong STM32.

1. Khối SDIO trong STM32F411.

Trong STM32F411 chúng ta có 1 khối SDIO. Khối này bao gồm:

- Bus APB2 có thể kết nối với DMA để tăng tốc độ truyền nhận dữ liệu Khối.
- SDMMC adapter: giúp kết nối với thẻ nhớ được điều khiển bởi các chân CK, MD và D.

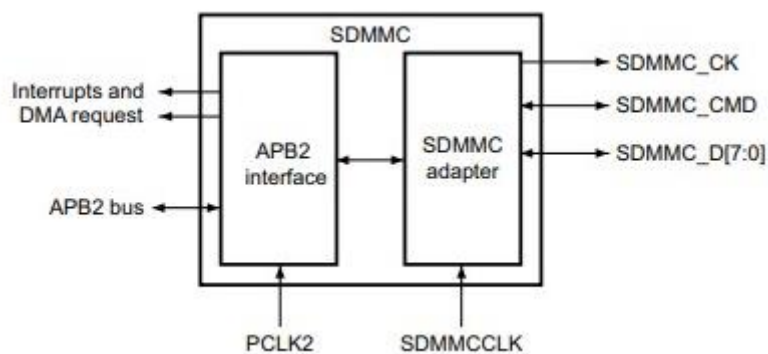


Figure 2: SDIO block diagram.

Chi tiết trong khối SDMMC Adapter:

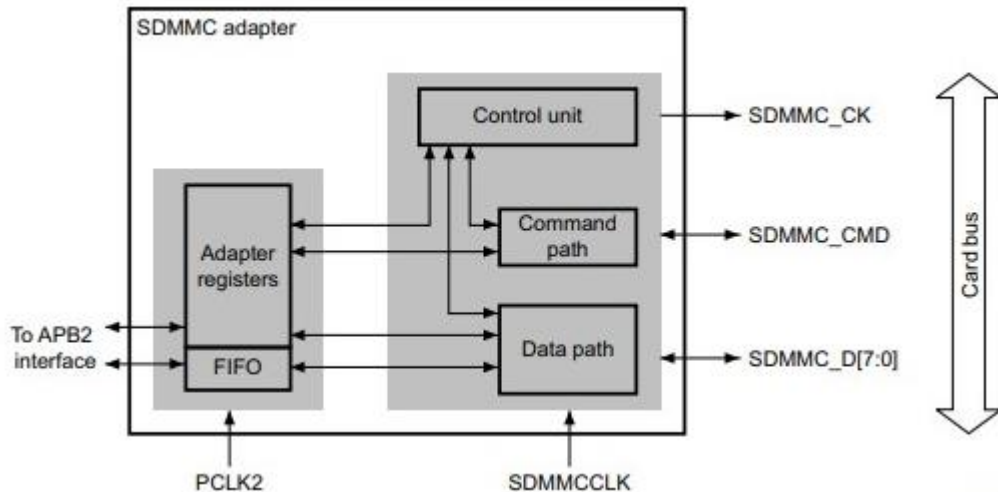


Figure 3: SDIO adapter.

Truyền nhận dữ liệu từ thẻ nhớ SD/SDIO được thực hiện trong các khối dữ liệu SDIO chỉ hỗ trợ mode giao tiếp 1bit (mặc định) và 4bit. SDIO 4 bit có tốc độ truyền nhận rất nhanh. Nếu sử dụng kèm theo DMA thì tốc độ xử lý sẽ rất nhanh SDIO trở nên phổ biến bằng cách có được tính năng kết nối bus SD đơn giản và hỗ trợ các chế độ tốc độ bus cao hơn.

2. Cơ chế hoạt động.

a. Giao thức bus SD.

Giao tiếp qua bus SD dựa trên luồng bit lệnh và dữ liệu được khởi tạo bởi một bit bắt đầu và được kết thúc bằng một bit dừng.

- **Command:** Lệnh là một mã thông báo bắt đầu một thao tác. Lệnh được gửi từ máy chủ đến một thẻ duy nhất (lệnh định địa chỉ) hoặc đến tất cả các thẻ được kết nối (lệnh phát sóng). Lệnh được truyền nối tiếp trên dòng CMD.
- **Response:** Phản hồi là một mã thông báo được gửi từ thẻ được định địa chỉ hoặc (đồng bộ) từ tất cả các thẻ được kết nối, đến máy chủ để trả lời cho lệnh đã được nhận trước đó. Phản hồi được truyền nối tiếp trên dòng CMD.
- **Data:** Dữ liệu có thể được truyền từ thẻ sang máy chủ hoặc ngược lại. Dữ liệu được truyền qua các đường dữ liệu.

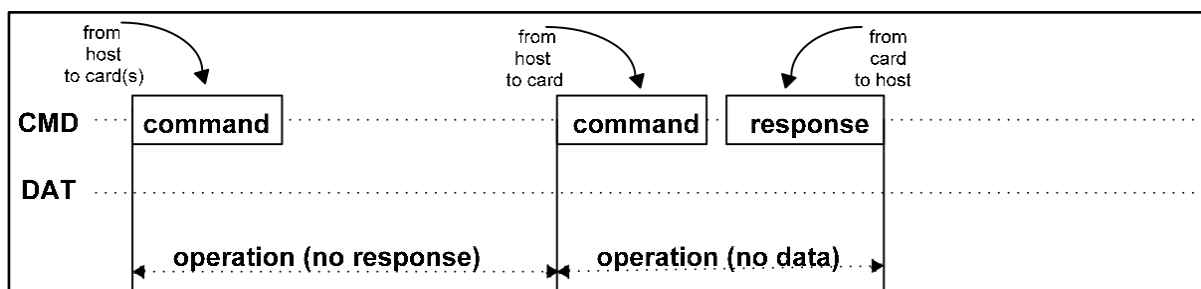


Figure 4: “no response” and “no data” Operations.

Việc định địa chỉ thẻ được thực hiện bằng cách sử dụng địa chỉ phiên, được gán cho thẻ trong giai đoạn khởi tạo.

Việc truyền dữ liệu đến / từ Thẻ nhớ SD được thực hiện theo các khối. Các khối dữ liệu luôn được kết thúc bởi các bit Cyclic Redundancy Code (CRC). Lưu ý rằng ở chế độ Multiple Block vận hành tốt hơn và ghi được nhanh hơn. Truyền ở chế độ Multiple Block bị chấm dứt khi lệnh dừng theo sau trên dòng CMD. Truyền dữ liệu có thể được cấu hình bởi máy chủ để sử dụng một hoặc nhiều dòng dữ liệu.

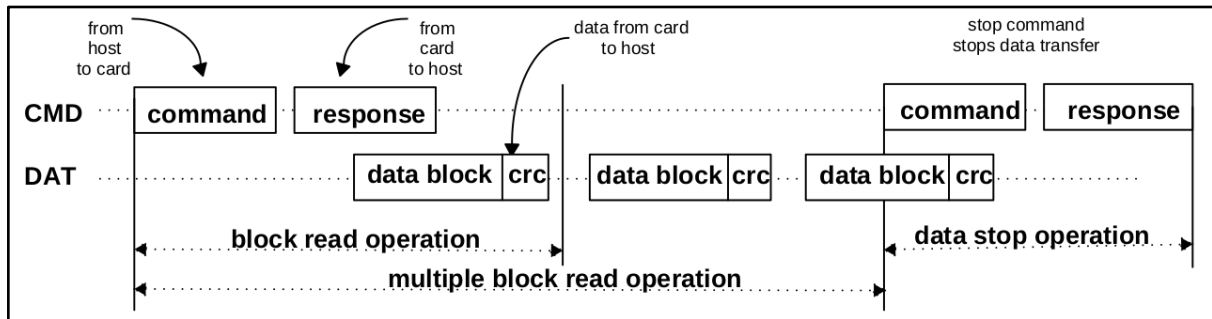


Figure 5: (Multiple) Block Read Operation.

Thao tác ghi khối sử dụng tín hiệu Busy đơn giản để báo hiệu thời gian thực hiện thao tác ghi trên dòng dữ liệu DAT0, bất kể số lượng dòng dữ liệu được sử dụng để truyền dữ liệu.

Mã thông báo lệnh có sơ đồ mã hóa sau:

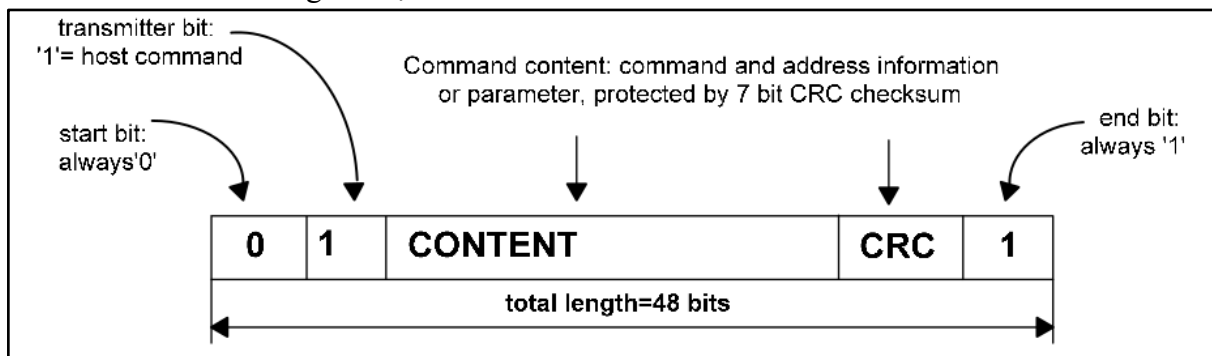


Figure 6: Command Token Format.

Mỗi mã thông báo lệnh bắt đầu bằng một bit bắt đầu (0) và kết thúc bằng một bit kết thúc (1). Tổng chiều dài là 48 bit. Mỗi mã thông báo được bảo vệ bởi các bit CRC để có thể phát hiện lỗi truyền và có thể lặp lại thao tác.

Mã phản hồi có một trong bốn sơ đồ mã hóa, tùy thuộc vào nội dung của chúng. Độ dài mã thông báo là 48 hoặc 136 bit. Thuật toán bảo vệ CRC cho dữ liệu khối là một đa thức CCITT 16 bit.

Trong dòng CMD, bit quan trọng nhất (MSB) được truyền trước, bit ít quan trọng nhất (LSB) là bit được truyền cuối cùng. Khi sử dụng tùy chọn bus rộng, dữ liệu được truyền 4 bit cùng một lúc. Các bit bắt đầu và kết thúc, cũng như các bit CRC, được truyền cho mỗi dòng DAT. Các bit CRC được tính toán và kiểm tra cho từng dòng DAT riêng lẻ. Phản hồi trạng thái CRC và Chỉ báo Busy sẽ chỉ được gửi đến máy chủ và lưu trữ trên DAT0. Có hai loại định dạng

gói Dữ liệu cho thẻ SD. (1) Dữ liệu thông thường (độ rộng 8 bit): Dữ liệu thông thường được gửi bằng LSB (Least Significant Byte) trước, MSB (Most Significant Byte) sau cùng. Nhưng trong từng byte riêng lẻ, nó là MSB (Most Significant Bit) đầu tiên, LSB (Least Significant Bit) cuối cùng. (2) Dữ liệu độ rộng lớn (SD Memory Register): Dữ liệu độ rộng được dịch chuyển từ bit MSB.

- Định dạng gói dữ liệu cho dữ liệu thông thường (độ rộng 8 bit).

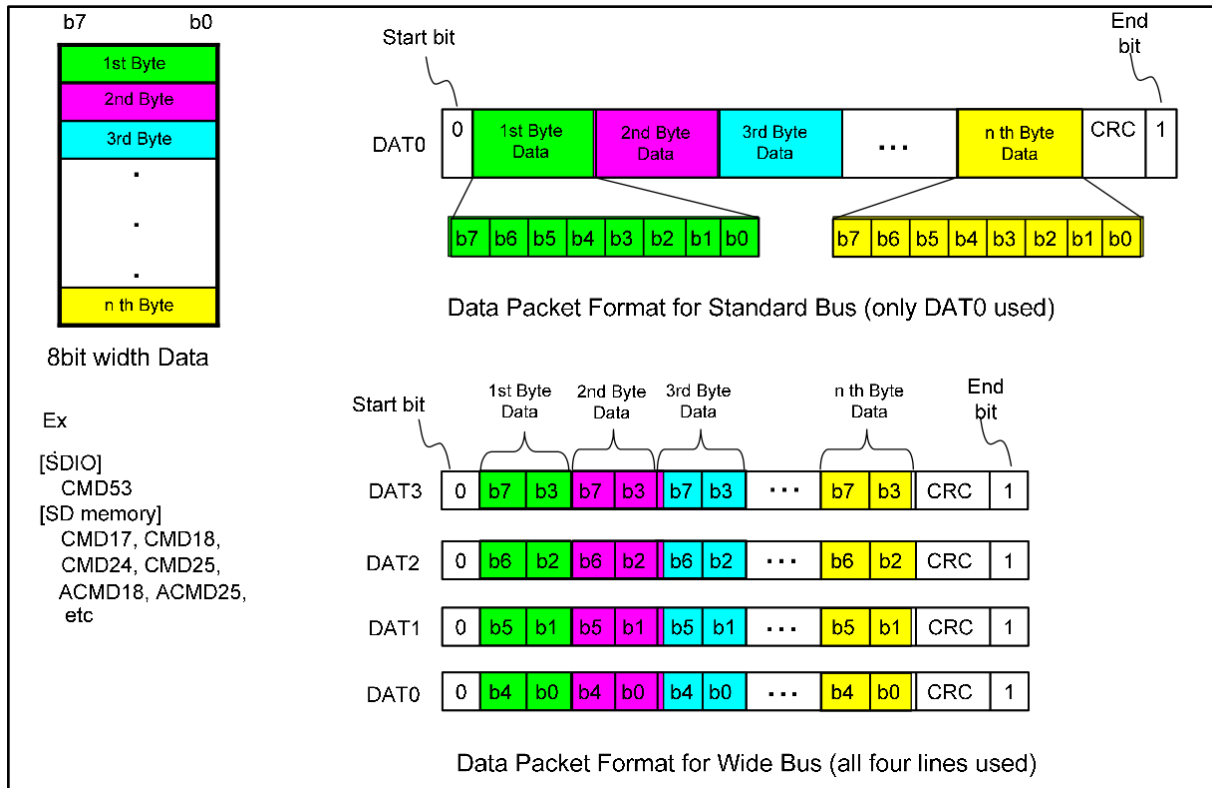


Figure 7: Data Packet Format - Usual Data.

- Định dạng gói dữ liệu cho dữ liệu có độ rộng lớn (ví dụ: ACMD13).

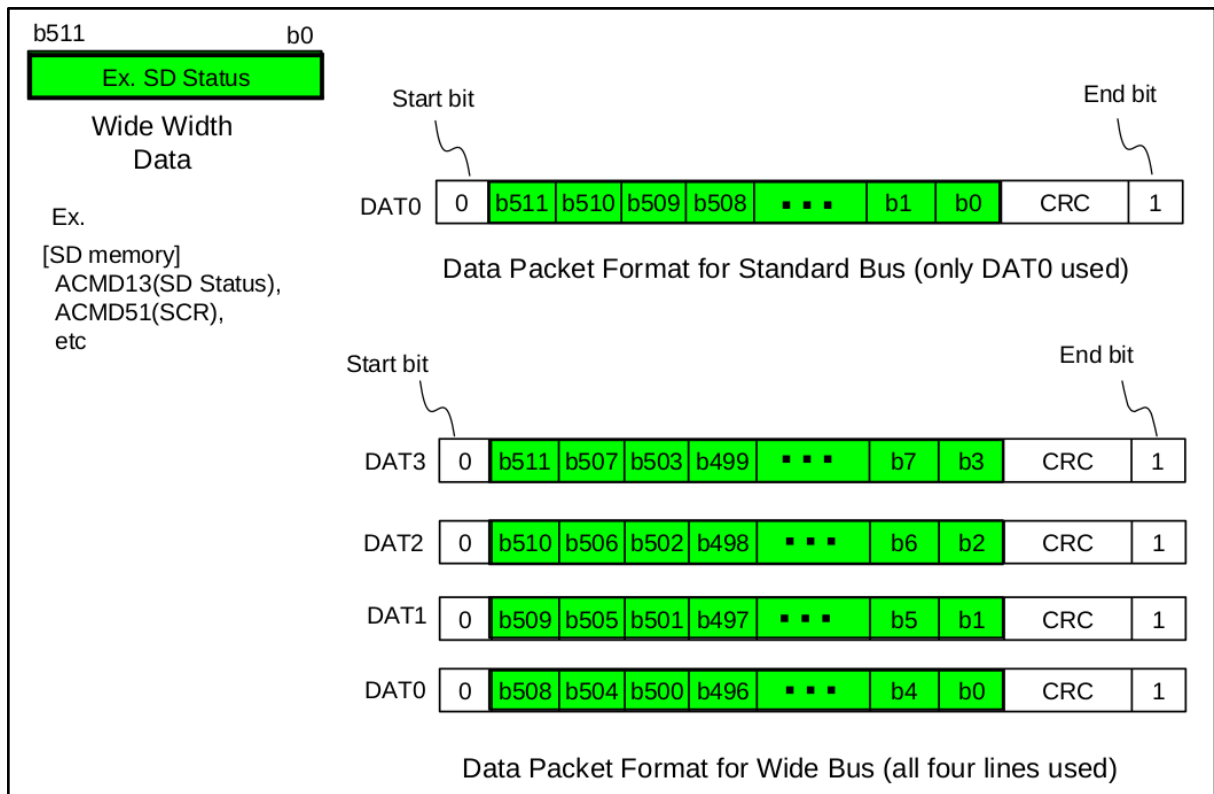


Figure 8: Data Packet Format - Wide Width Data.

b. Một số thanh ghi quan trọng của khối SDIO trong STM32F4.

- SDIO data control register (SDIO_DCTRL).

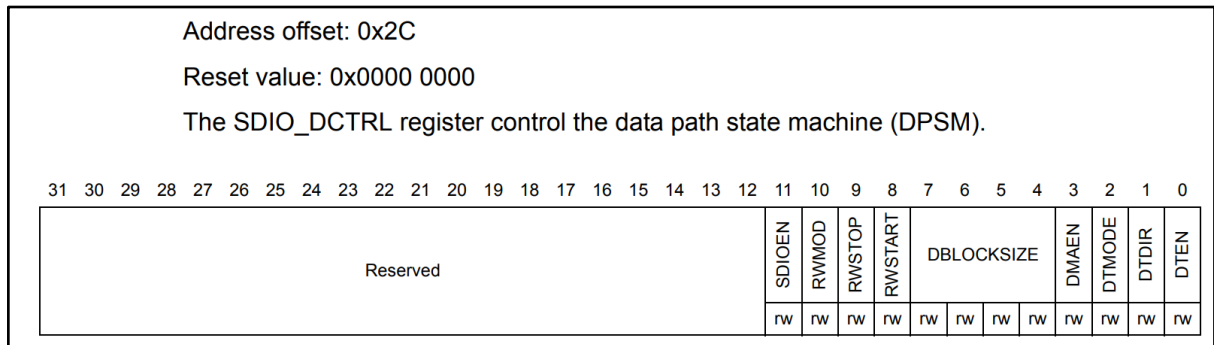


Figure 9:

- SDIO status register (SDIO_STA).

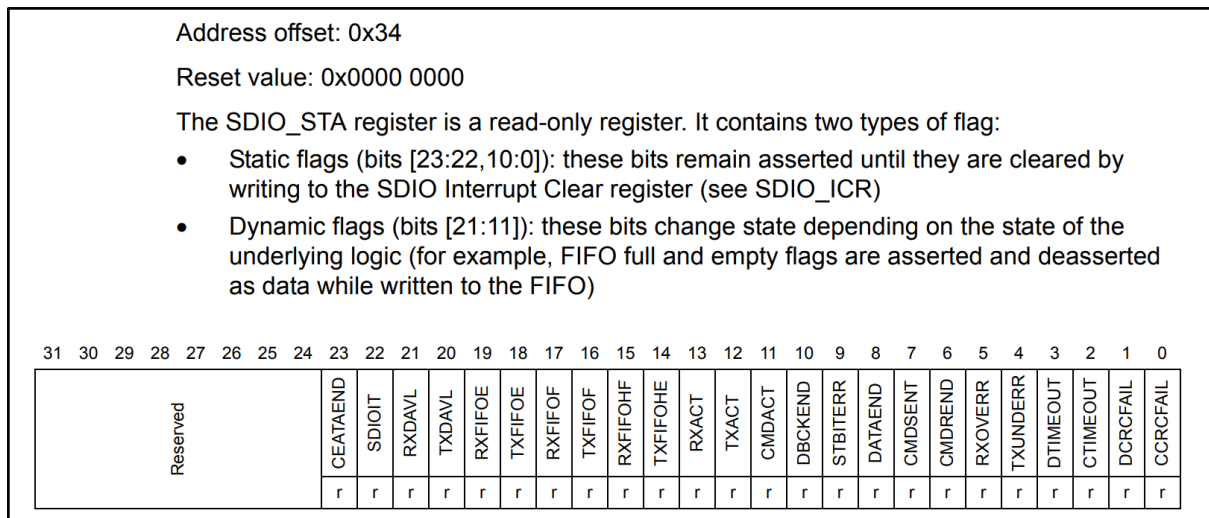


Figure 10:

3. Tổng quan về thư viện FATFS.

FATFS là một loại module hệ thống tệp FAT chung cho các hệ thống nhúng. FATFS được viết phù hợp với ANCI C và tách biệt với lớp I/O nên nó độc lập với phần cứng.

FATFS nằm ở tầng Middleware giữa tầng hardware và Application trong hệ thống nhúng. Nghĩa là chúng không phụ thuộc vào phần cứng cũng không phụ thuộc vào phần mềm.

Các lệnh FATFS chúng ta hay sử dụng chủ yếu là:

- `f_mount()`: Kiểm tra kết nối hoặc hủy kết nối với thẻ.
- `f_open()`: Mở/Tạo file.
- `f_close()`: Đóng file.
- `f_read()`: Đọc dữ liệu từ file.
- `f_write()`: Ghi dữ liệu vào file.
- `f_lseek()`: Di chuyển con trỏ đọc/ghi, Mở rộng kích thước file.
- `f_truncate()`: Cắt ngắn kích thước file.
- `f_sync()`: Đẩy dữ liệu cache.
- `f_opendir()`: Mở đường dẫn.

Tham khảo thêm ở website: http://elm-chan.org/fsw/ff/00index_e.html

Trước khi thao tác với thẻ cần phải có bước gắn thẻ (mount the SD card) để phân vùng làm việc cho ổ đĩa cũng như kiểm tra thẻ và phần cứng. Bước cuối cùng cần phải hủy kết nối với thẻ sau khi thực hiện xong (unmount the SD card).

Để xử lý 1 file trong thẻ thông thường ta cần thực hiện 3 bước: Mở file – Đọc/ ghi/ chỉnh sửa file – Đóng file. Nếu xóa file thì chỉ cần làm 2 bước: Mở file - xóa file.

Chính vì thế nên trong project này chúng ta sẽ thực hiện tổng cộng 5 bước: Gắn thẻ - Mở file – Xử lý file – (Đóng file) - Huỷ thẻ.

Nội dung Project:

- Gắn thẻ.
- Tìm và in dung lượng thẻ và dung lượng còn trống trên màn hình SWV ITM Data console.
- Tạo file văn bản mới.
- Ghi vào file văn bản - Sử dụng hàm `f_puts()`.
- Ghi vào file văn bản - Sử dụng hàm `f_write()`.
- Đọc từ file văn bản - Sử dụng hàm `f_gets()`.
- Đọc từ file văn bản - Sử dụng hàm `f_read()`.
- Chỉnh sửa và cập nhật một file hiện có.
- Xóa file.
- Huỷ kết nối với thẻ.

Thực hiện trên thẻ nhớ microSD sử dụng giao thức SDIO với STM32F411.

4. Sơ đồ kết nối phần cứng.

Trong bài này chúng ta sẽ xét tới chế độ 4 bit. Đầu tiên các bạn nối các chân của SDIO vào thẻ nhớ như sau:

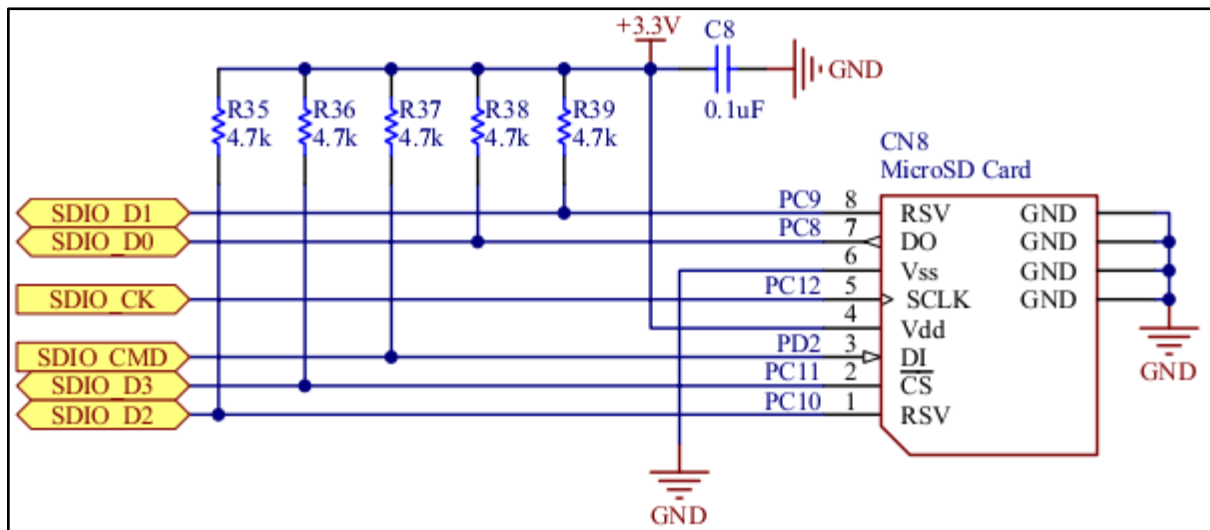


Figure 11:

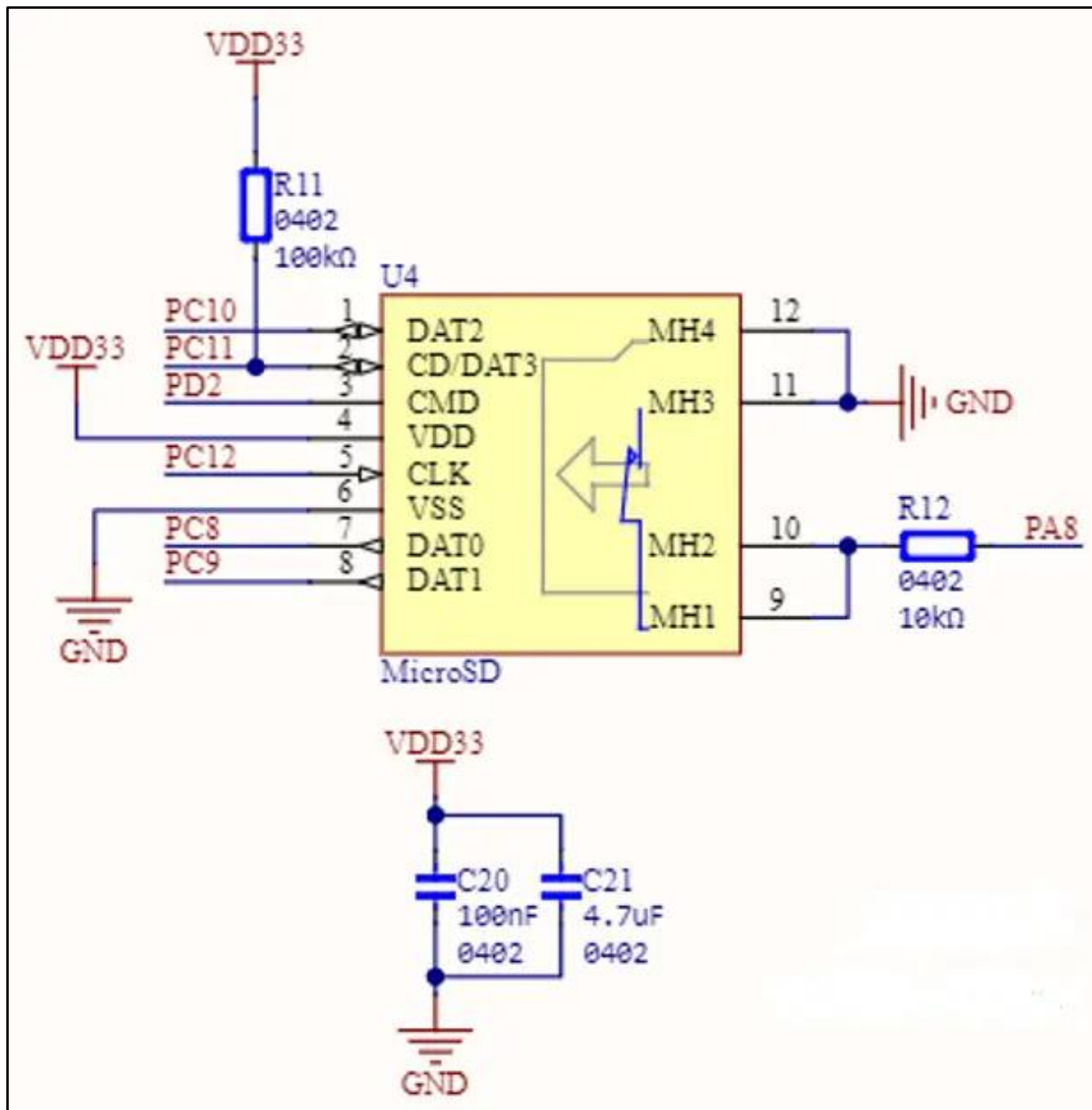


Figure 12:

Ý nghĩa của các chân:

- Các chân D0 D1 D2 D3 là các chân dữ liệu , tối đa là 4 bit. Với chế độ 1bit sẽ mặc định là chân D0 ,các chân dữ liệu có trở kéo lên.
- Chân CMD (Command) chọn gửi lệnh hay Data.
- Chân GND và 3.3(V) là chân cấp nguồn.
- CLK là chân cấp xung clock.

Chương trình sẽ mặc định các chân GPIO của stm32F411 ứng với module SDIO như sau:

PA6	SDIO_CMD
PB15	SDIO_CK

PC8	SDIO_D0
-----	---------

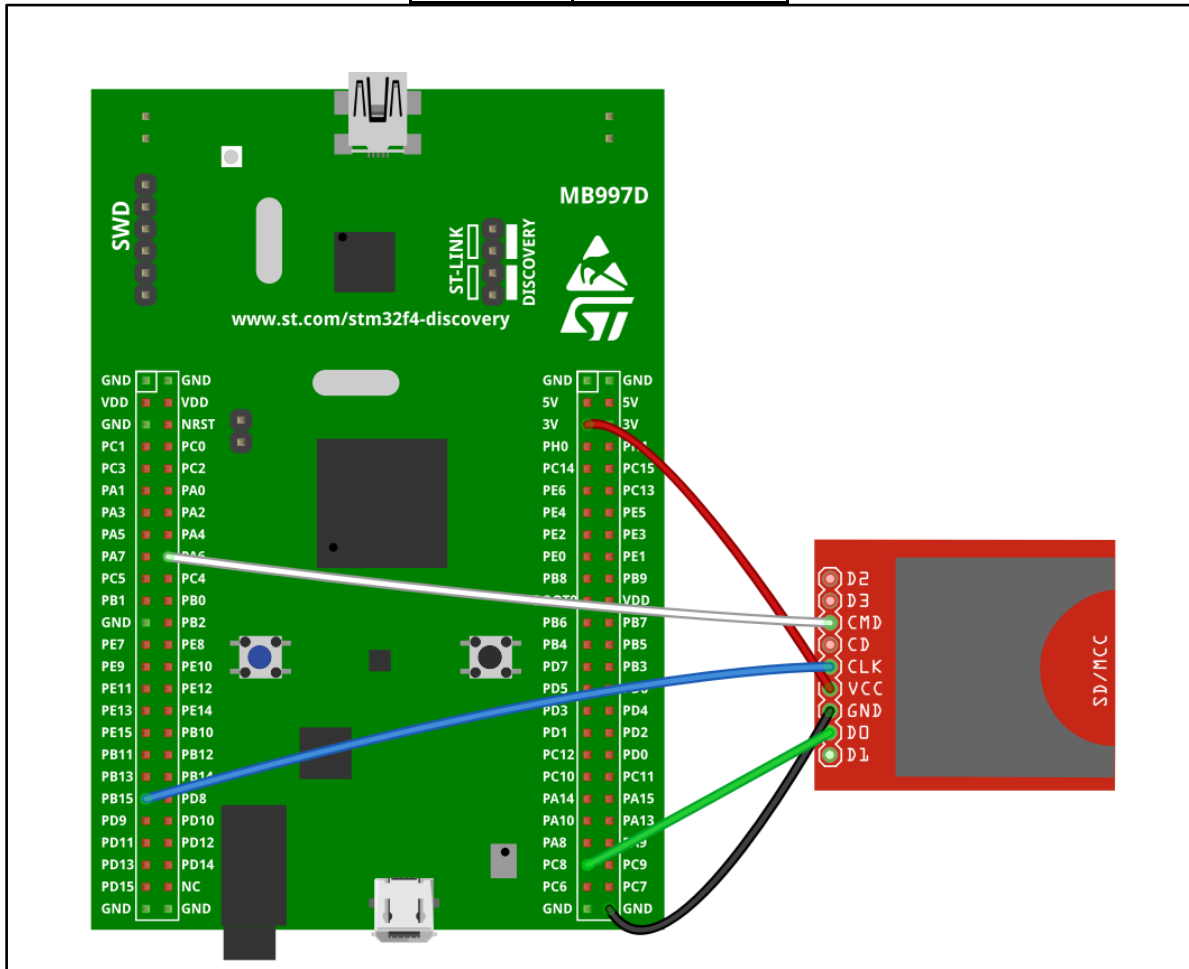


Figure 13: Sơ đồ nối dây

Đầu tiên các bạn hãy chuẩn bị thẻ SD và module đọc thẻ SD theo chuẩn SDIO.



Figure 14: module đọc thẻ SD theo chuẩn SDIO

Tiếp đó các bạn chuẩn bị một đầu đọc thẻ để test kết quả file đã ghi.



Figure 15: đầu đọc thẻ nhớ

Ngoài ra các bạn nên format lại thẻ microSD để thẻ ở đúng định dạng F

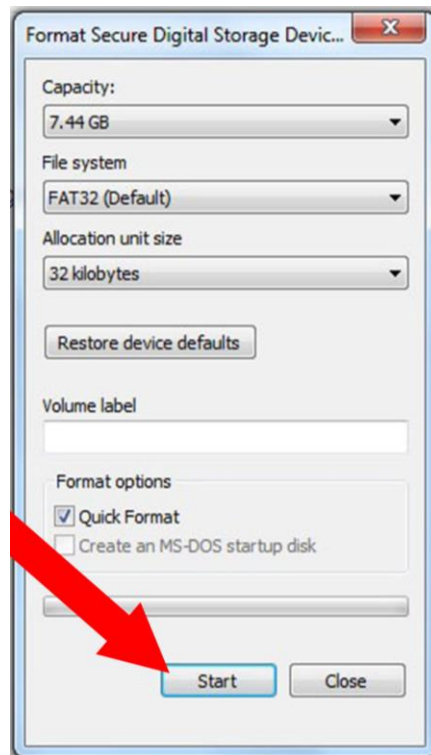


Figure 16: Format thẻ nhớ SD.

5. Cấu hình CubeMX.

- Bước 1: Khởi động phần mềm STM32CubeIDE. Kích chọn dòng STM32F411VETx và chọn “Next”.

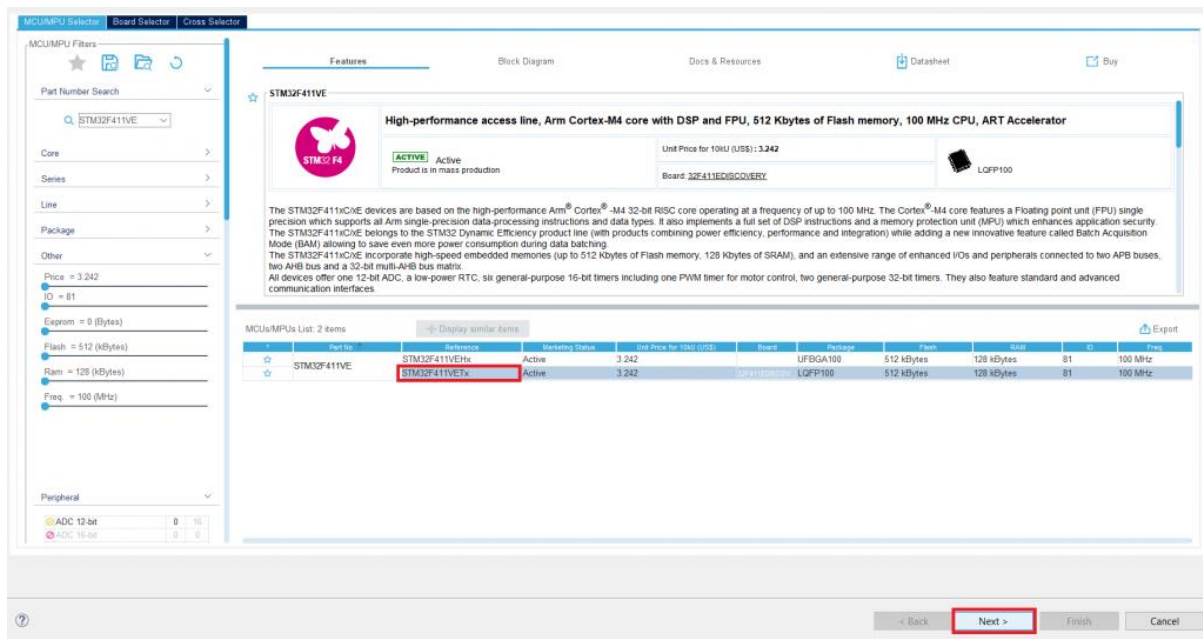


Figure 17:

- Bước 2: Ở mục System core -> SYS -> Debug-> Bật serial wire.

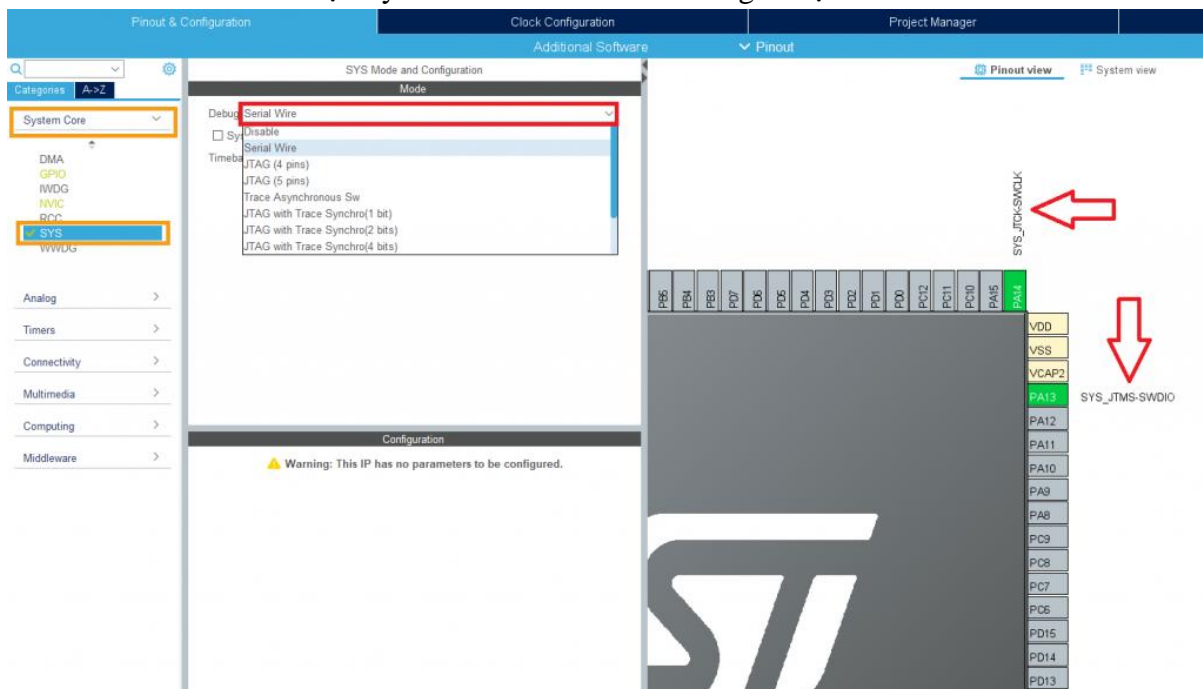


Figure 18:

- Bước 3: Cấu hình xung clock là thạch anh ngoài.
 - Ở mục System Core-> RCC-> High Speed Clock (HSE)-> Crystal/Ceramic Resonator.

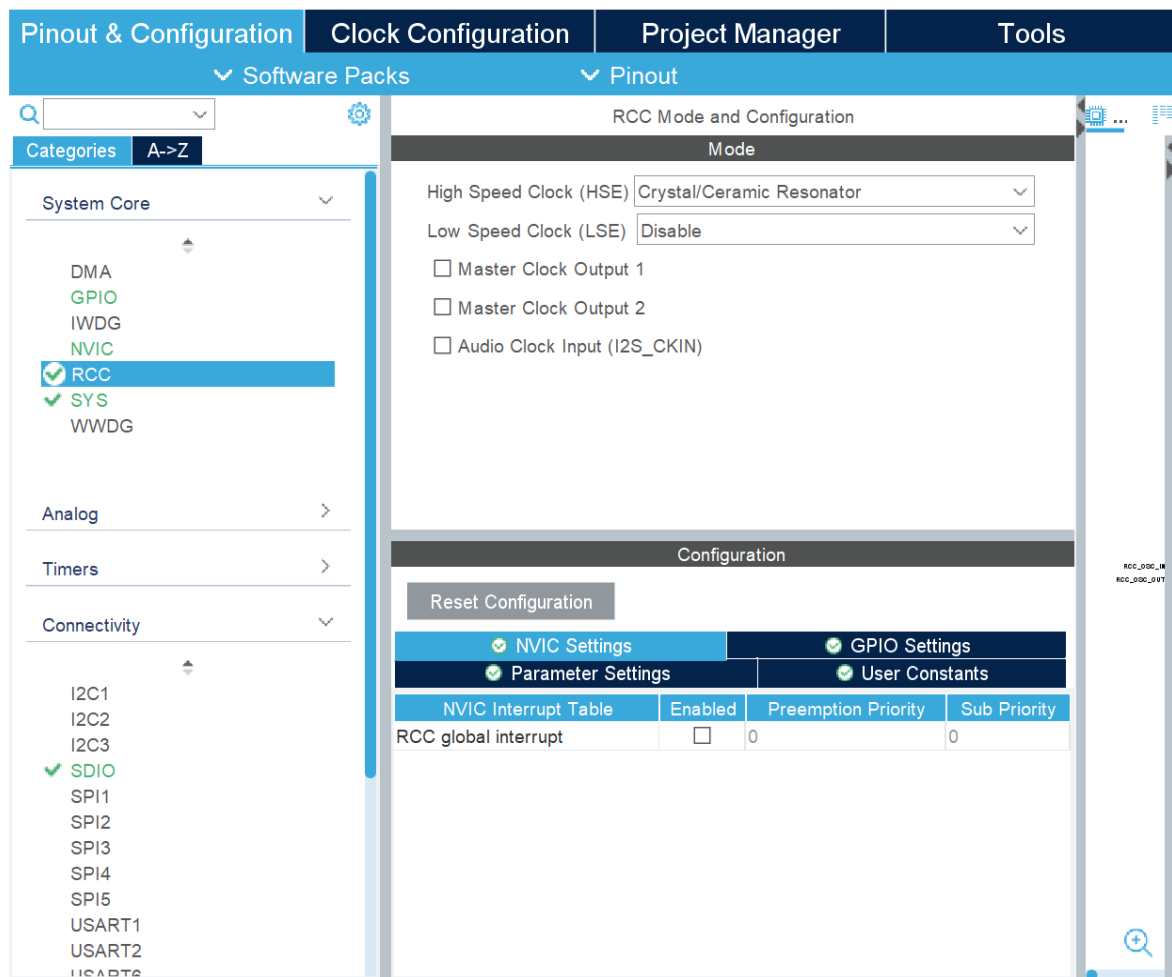


Figure 19:

- Ở mục Clock Configuration, chúng ta sẽ để input frequency ở 8MHz và chọn nguồn clock HSE với các thông số được cài đặt như hình bên dưới:

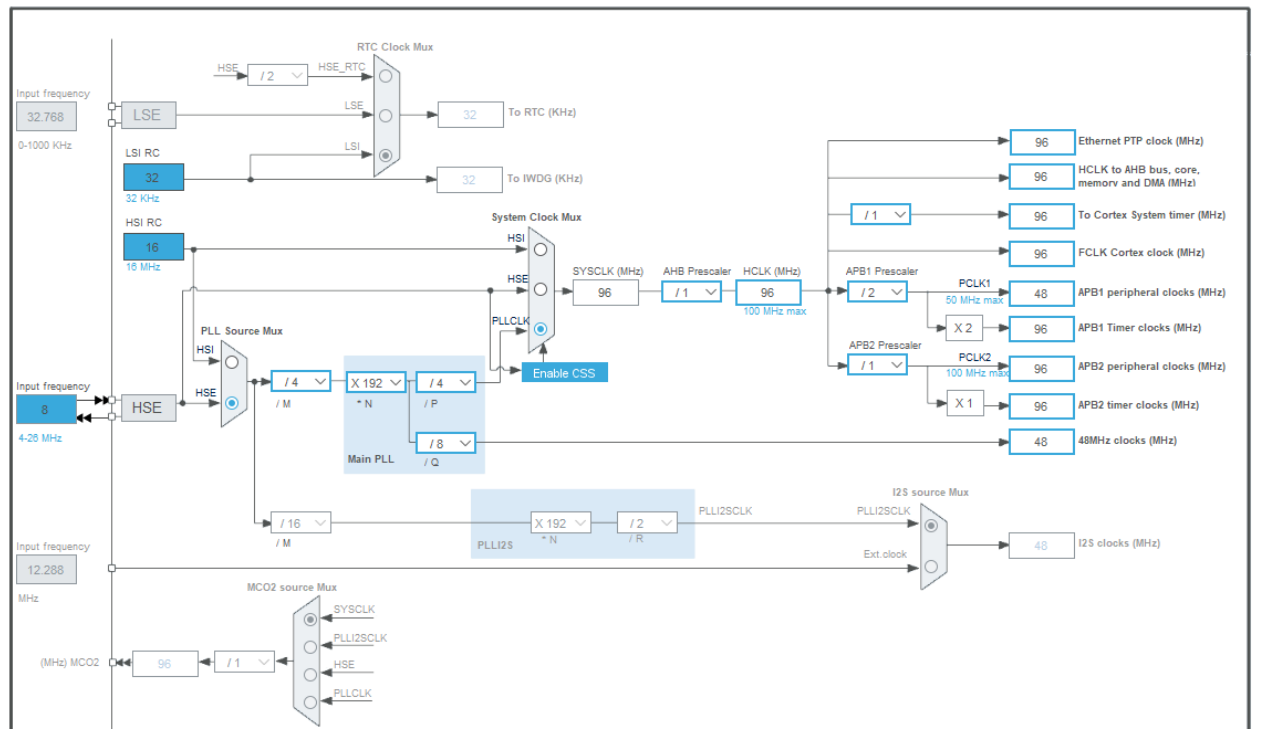


Figure 20:

- Bước 4: Cấu hình SDIO.
 - Ở mục Connectivity -> SDIO-> Mode-> SD 1 bit (chế độ 1 bit hoặc 4 bit đều tương tự nhau).
 - Ở mục configuration->NVIC settings -> Tick vào mục Enabled.

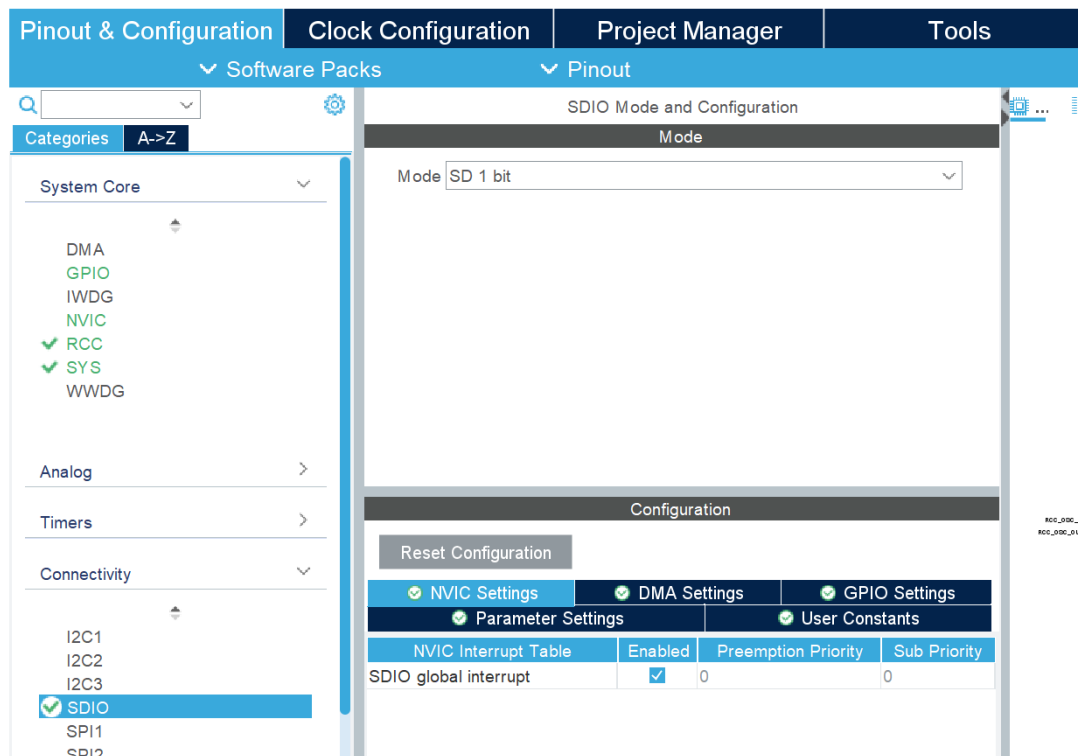


Figure 21:

- Chú ý, trong mục configuration->Parameter Setting : chúng ta có thể chọn số chia cho SDIOCLK clock divide factor để lấy tần số đọc thẻ nhớ. Ở đây ta để SDIOCLK clock divide factor là 0 để lấy tần số đọc lớn nhất ($48/2 = 24$ Mhz) .Chúng ta cần chọn hệ số chia cho phù hợp để được tần số đọc thẻ nhớ thỏa mãn thông số của thẻ nhớ ta đang dùng.

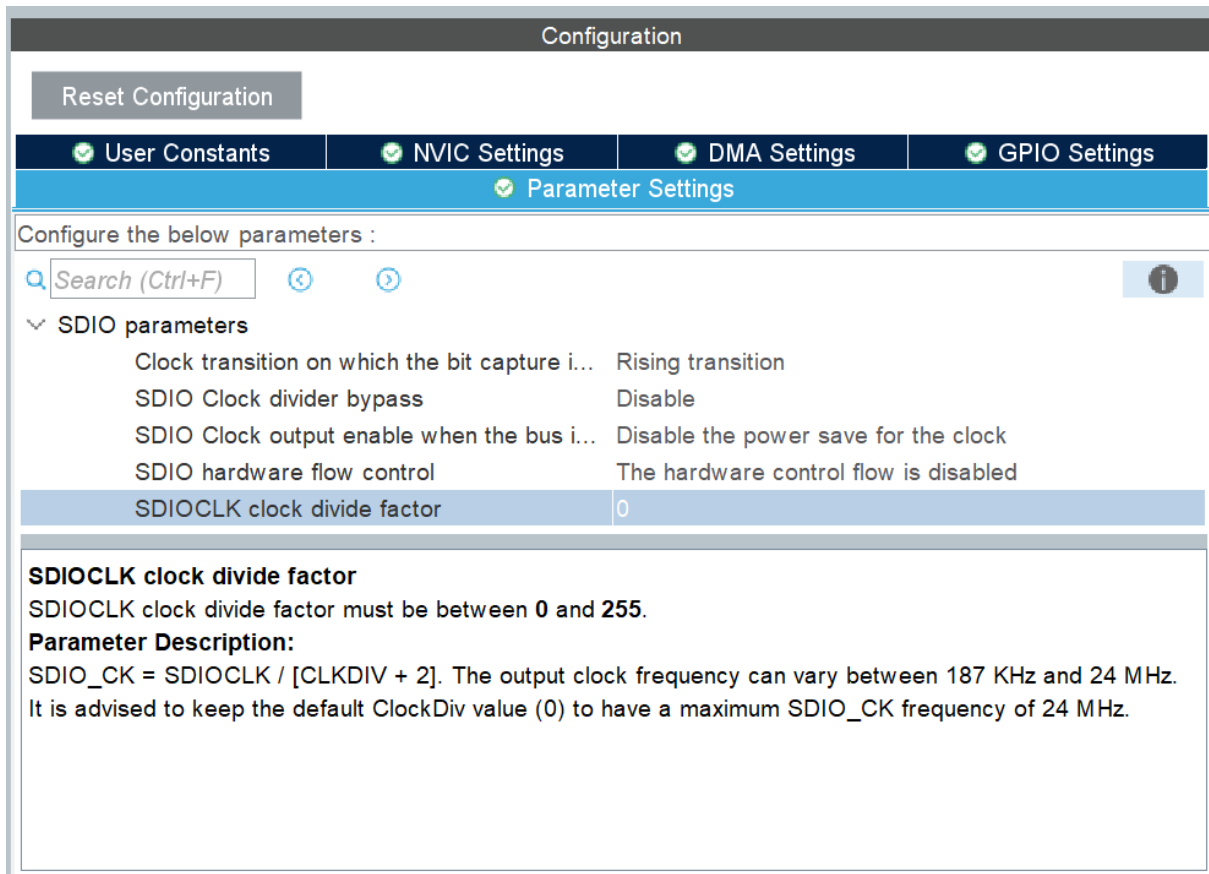


Figure 22:

- Bước 5: Cấu hình FATFS, chọn SDCard
 - Ở mục Middleware and Software Packs -> FATFS->FATFS Mode and Configuration.
 - Ở mục Mode -> Tick in to SD Card. Mình có thể cấu hình mặc định cũng được.

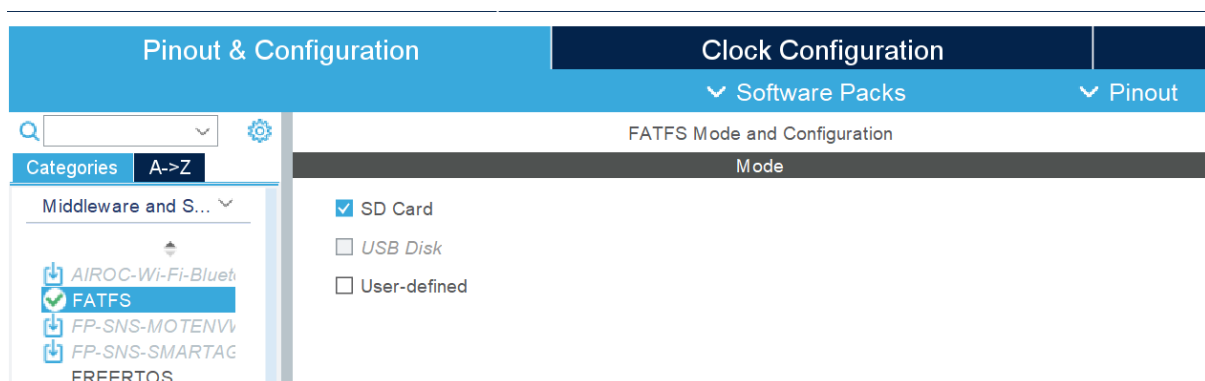


Figure 23:

- Bước 6: Cấu hình chân Detect_SDIO.
 - Nếu bạn không làm bước này thì khi sinh code chương trình sẽ hiện lên warning như thế này. Thực ra chân này các bạn không dùng cũng được. Nhấn Yes để tiếp tục.

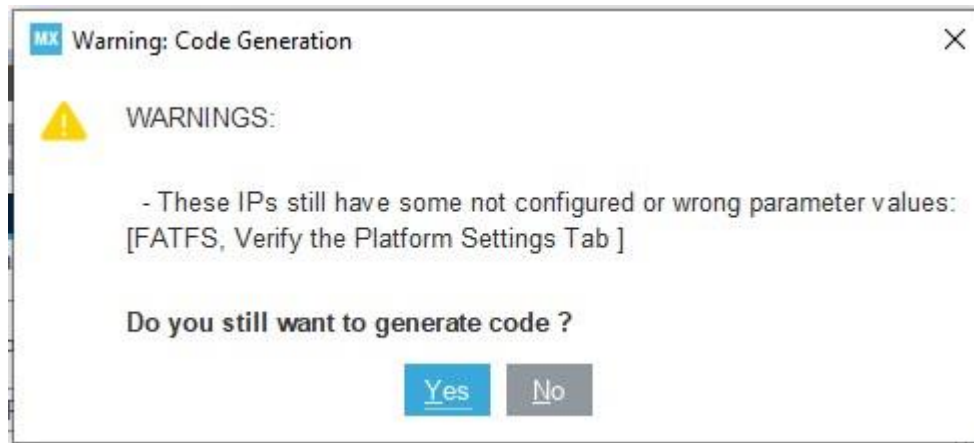


Figure 24:

- Chọn PD0 at mode GPIO input (hoặc chân nào bất kì) -> sau đó vào mục Platform Settings, ở mục Found Solutions-> ấn chọn tên của chân mà chúng ta vừa chọn (Đây là chân để chúng ta dùng để detect cho việc phát hiện thẻ nhớ đã được cắm vào hay chưa).

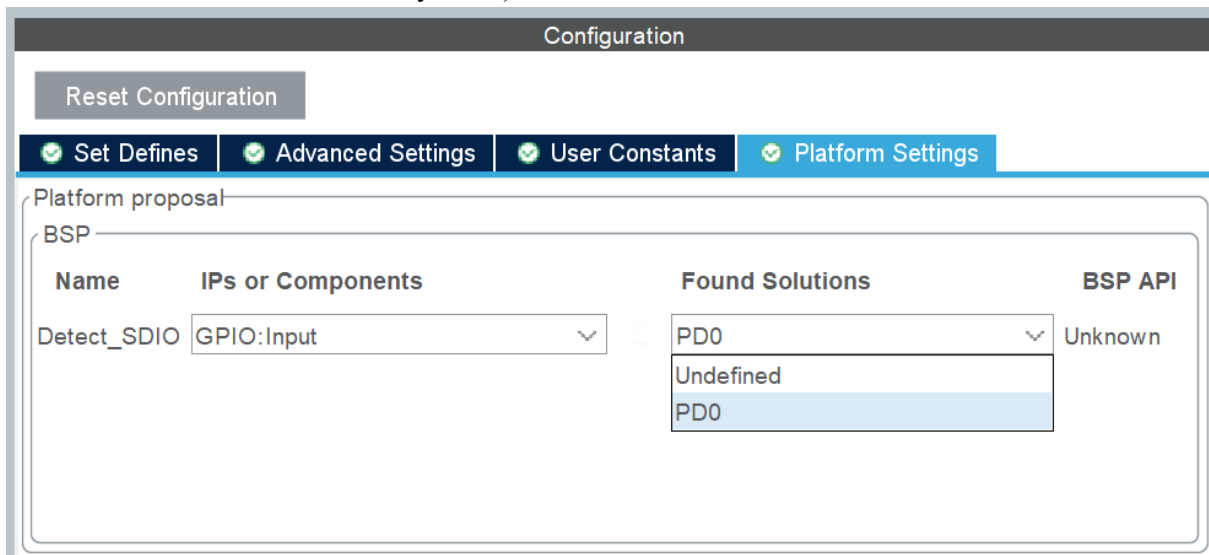


Figure 25:

- Bước 7: Sinh code bằng cách nhấn tổ hợp phím Alt+K.

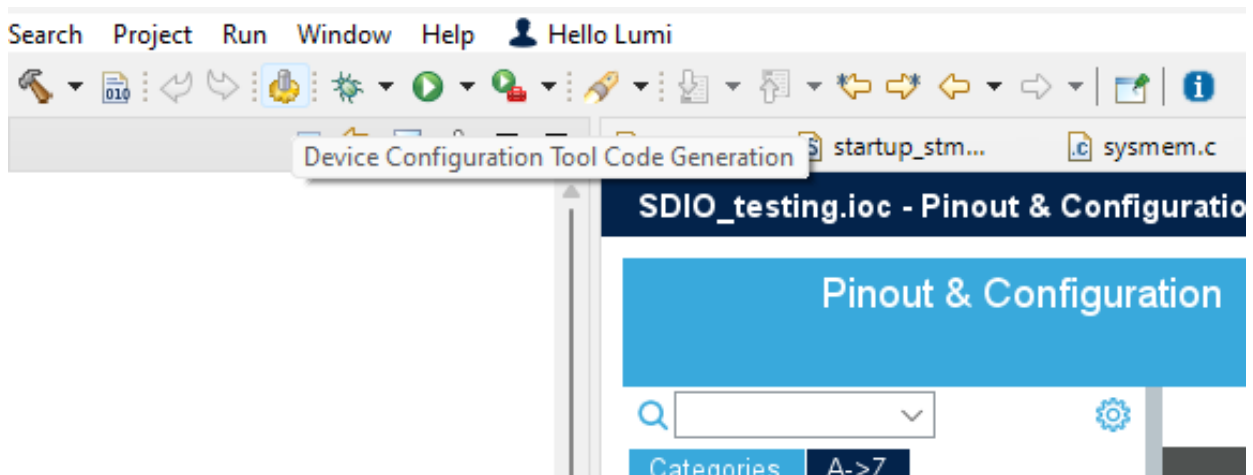


Figure 26:

- Bước 8: Lập trình bằng phần mềm STM32Cube IDE.
 Chúng ta sẽ viết một chương trình để tạo một file .txt trong thẻ nhớ và sao đó thực hiện mở file để ghi dữ liệu sau đó đóng file.
 - Trước tiên chúng ta sẽ thêm các header file string.h, stdio.h cho kiểu dữ liệu string và hiển thị thông báo ra màn hình SWV ITM Data console:


```
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <string.h>
/* USER CODE END Includes */
```
 - Khai báo các object của FATFS, FIL, FRESULT, các biến biến để phục vụ việc đọc ghi trong đó:
 1. FatFs là biến hệ thống, mỗi biến này tương ứng với một thẻ nhớ.
 2. Fil là biến tương ứng với 1 file trong thẻ, chúng ta sẽ sử dụng biến đó để thao tác như đọc, ghi, xóa...
 3. FR_Status là biến hiển thị kết quả mỗi thao tác.
 4. FS_Ptr: Biến này là con trỏ đến cấu trúc FATFS.
 5. RWC, WWC: lưu trữ số byte được đọc từ file trong quá trình gọi hàm f_read, f_write.
 6. FreeClusters: lưu trữ số lượng cụm trống có sẵn trên thẻ SD.
 7. TotalSize, FreeSpace: lưu trữ kích thước tổng thẻ và dung lượng trống có sẵn trên thẻ SD.
 8. RW_Buffer: Mảng ký tự sử dụng làm bộ đệm để lưu trữ dữ liệu được đọc từ hoặc ghi vào thẻ SD.

```
/* USER CODE BEGIN PV */
FATFS FatFs;
FIL Fil;
FRESULT FR_Status;
FATFS *FS_Ptr;
UINT RWC, WWC; // Read/Write Word Counter
DWORD FreeClusters;
uint32_t TotalSize, FreeSpace;
```

```
char RW_Buffer[200];
```

```
/* USER CODE END PV */
```

- Chúng ta sẽ code để thực hiện các nhiệm vụ:
 1. Gắn thẻ và kiểm tra tình trạng kết nối với SDIO module.
 2. Lấy thông tin về kích thước và dung lượng trống của thẻ SD.
 3. Mở file để đọc dữ liệu từ file vừa được tạo và lưu vào mảng rồi đóng file.
 4. Mở file cập nhật nội dung của tệp văn bản hiện có và đóng file.
 5. Mở file để đọc data từ file vừa thêm dữ liệu và lưu vào mảng rồi đóng file.
 6. Xóa file.
 7. Huỷ gắn thẻ.

```
/* USER CODE BEGIN 2 */
```

```
HAL_Delay(1000);
```

```
do
```

```
{
```

```
//-----[ Mount The SD Card ]-----//
```

```
// Checking the status of connection with SD card is FR_OK
```

```
FR_Status = f_mount(&FatFs, "", 1);
```

```
if (FR_Status != FR_OK)
```

```
{
```

```
printf("Error! While Mounting SD Card, Error Code: (%i)\r\n",
```

```
FR_Status);
```

```
break;
```

```
}
```

```
printf("SD Card Mounted Successfully! \r\n\r\n");
```

```
//----[ Get & Print The SD Card Size & Free Space]-----
```

```
f_getfree("", &FreeClusters, &FS_Ptr);
```

```
TotalSize = (uint32_t)((FS_Ptr->n_fatent - 2) * FS_Ptr->csize * 0.5);
```

```
FreeSpace = (uint32_t)(FreeClusters * FS_Ptr->csize * 0.5);
```

```
printf("Total SD Card Size: %lu Bytes\r\n", TotalSize);
```

```
printf("Free SD Card Space: %lu Bytes\r\n\r\n", FreeSpace);
```

```
//--[ Open A Text File For Write & Write Data and then Close file ]-
```

```
-----//
```

```
//Open the file
```

```
FR_Status = f_open(&Fil, "STM32.TXT",
```

```
FA_CREATE_ALWAYS | FA_WRITE);
```

```
if (FR_Status != FR_OK)
```

```
{
```

```

    printf("Error! While Creating/Opening A New Text File, Error
Code: (%i)\r\n", FR_Status);
    break;
}
//Write data into the file
printf("Text File Created & Opened! Writing Data To The Text
File..\r\n\n");
// (1) Write Data To The Text File [ Using f_puts() Function ]
f_puts("Hello! From STM32 To SD Card Over SDIO, Using
f_puts()\n", &Fil);
// (2) Write Data To The Text File [ Using f_write() Function ]
strcpy(RW_Buffer, "Hello! From STM32 To SD Card Over SDIO,
Using f_write()\r\n");
f_write(&Fil, RW_Buffer, strlen(RW_Buffer), &WWC);
// Close The File
f_close(&Fil);

//--[ Open A Text File For Read & Read Its Data and then Close file
]-----
// Open The File
FR_Status = f_open(&Fil, "STM32.TXT", FA_READ);
if(FR_Status != FR_OK)
{
    printf("Error! While Opening (STM32.TXT) File For Read.. \r\n");
    break;
}
// (1) Read The Text File's Data [ Using f_gets() Function ]
f_gets(RW_Buffer, sizeof(RW_Buffer), &Fil);
printf("Data Read From (STM32.TXT) Using f_gets():%s",
RW_Buffer);
// (2) Read The Text File's Data [ Using f_read() Function ]
f_read(&Fil, RW_Buffer, f_size(&Fil), &RWC);
printf("Data Read From (STM32.TXT) Using f_read():%s",
RW_Buffer);
// Close The File
f_close(&Fil);
printf("File Closed! \r\n\n");

//-[ Open An Existing Text File, Update Its Content, Read It Back]-
// (1) Open The Existing File For Write (Update)
FR_Status = f_open(&Fil, "STM32.TXT", FA_OPEN_EXISTING |
FA_WRITE);
FR_Status = f_lseek(&Fil, f_size(&Fil)); // Move The File Pointer
To The EOF (End-Of-File)

```

```

    if(FR_Status != FR_OK)
    {
        printf("Error! While Opening (STM32.TXT) File For Update..
\r\n");
        break;
    }
    // (2) Write New Line of Text Data To The File
    FR_Status = f_puts("This New Line Was Added During File
Update!\r\n", &Fil);
    f_close(&Fil);
    memset(RW_Buffer, '\0', sizeof(RW_Buffer)); // Clear The Buffer

    // (3) Read The Contents of The Text File After The Update
    FR_Status = f_open(&Fil, "STM32.TXT", FA_READ); // Open The
File For Read
    f_read(&Fil, RW_Buffer, f_size(&Fil), &RWC);
    printf("Data Read From (STM32.TXT) After Update:\r\n%s",
RW_Buffer);
    f_close(&Fil);

    //-----[ Delete The Text File ]-----
    // Delete The File
    /*
    FR_Status = f_unlink(STM32.TXT);
    if (FR_Status != FR_OK){
        printf("Error! While Deleting The (STM32.TXT) File.. \r\n");
    }
    */

    //-----[ Create folder ]-----

    FR_Status = f_mkdir("/MYFOLDER\0");
    if(FR_Status != FR_OK)
    {
        printf("Error! While Creating/Opening A New Folder, Error Code:
(%i)\r\n", FR_Status);
        break;
    }

    FR_Status = f_open(&Fil, "/MYFOLDER/STM32.TXT",
FA_CREATE_ALWAYS | FA_WRITE);
    if(FR_Status != FR_OK)
    {

```

```

        printf("Error! While Creating/Opening A New Text File in folder,
Error Code: (%i)\r\n", FR_Status);
        break;
    }
    //Write data into the file
    printf("Text File in Folder Created & Opened! Writing Data To The
Text File..\r\n\n");
    // (1) Write Data To The Text File [ Using f_puts() Function ]
    f_puts("Hello! From STM32 To SD Card Over SDIO, Using
f_puts(), file in Folder \n", &Fil);
    // (2) Write Data To The Text File [ Using f_write() Function ]
    strcpy(RW_Buffer, "Hello! From STM32 To SD Card Over SDIO,
Using f_write(), file in Folder\r\n");
    f_write(&Fil, RW_Buffer, strlen(RW_Buffer), &WWC);
    // Close The File
    f_close(&Fil);

} while(0);

//-----[ Test Complete! Unmount The SD Card ]-----
FR_Status = f_mount(NULL, "", 0);
if (FR_Status != FR_OK)
{
    printf("\r\nError! While Un-mounting SD Card, Error Code:
(%i)\r\n", FR_Status);
} else{
    printf("\r\nSD Card Un-mounted Successfully! \r\n");
}

```

Chú ý rằng ta cần thêm function để sử dụng hàm printf() cho việc hiển thị thông báo ra màn hình SWV ITM Data console :

```

/* USER CODE BEGIN 4 */
int _write(int file, char *ptr, int len)
{
    int DataIdx;
    for (DataIdx = 0; DataIdx < len; DataIdx++)
    {
        ITM_SendChar(*ptr++);
    }
    return len;
}
/* USER CODE END 4 */

```

- Bước 9: Setup màn hình SWV ITM Data console.
 - Nhấn Debug -> Debug configurations.

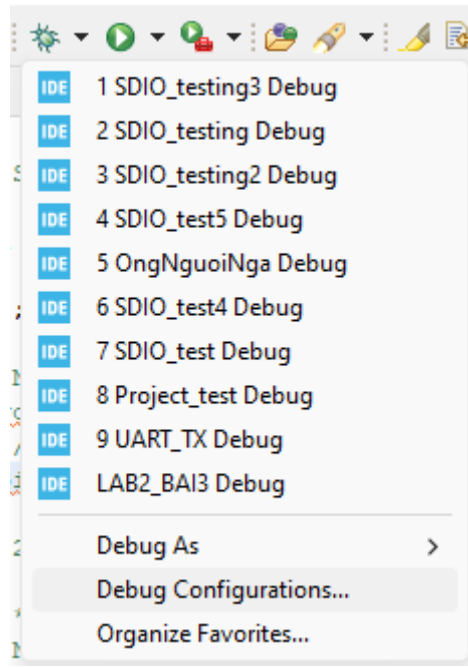


Figure 27:

- Trong cửa sổ Debug configurations -> Debugger -> Enable Serial Wire Viewer (SWV).
- Nhấn Debug.

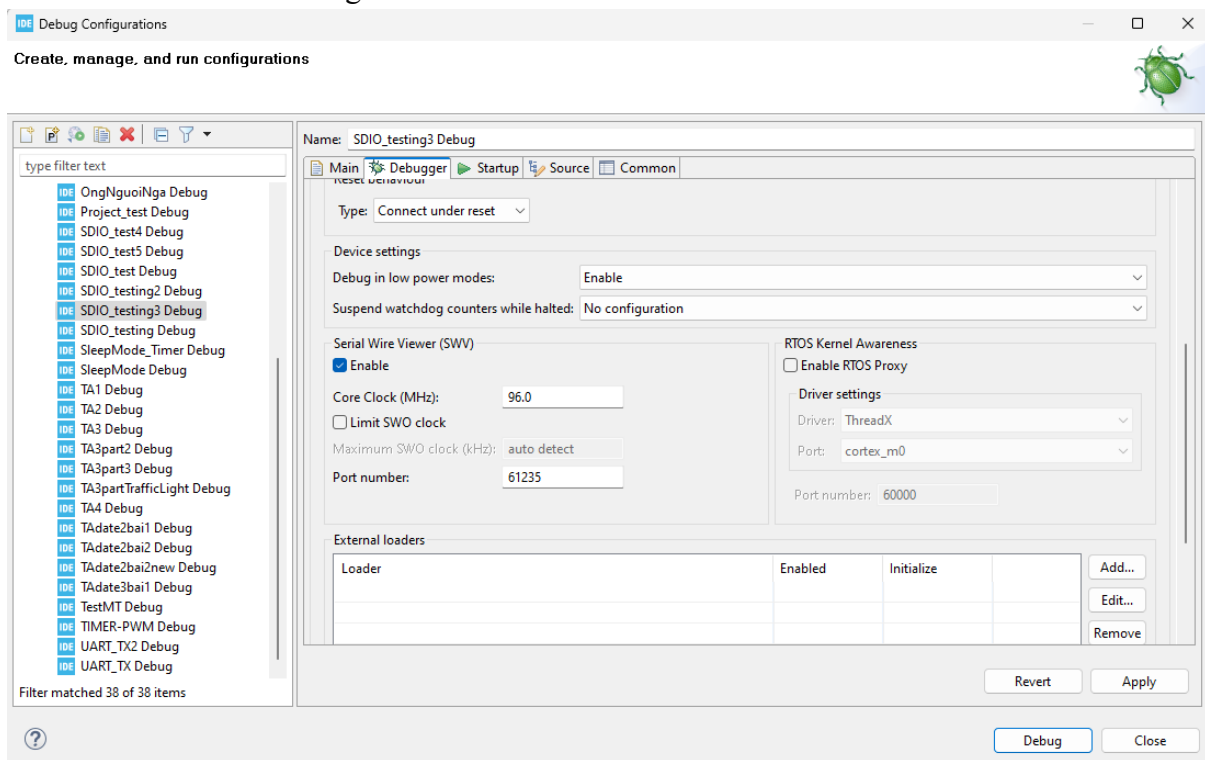


Figure 28:

- Trong giao diện debug, nhấn Window -> Show View -> SWV -> SWV ITM Data Console.

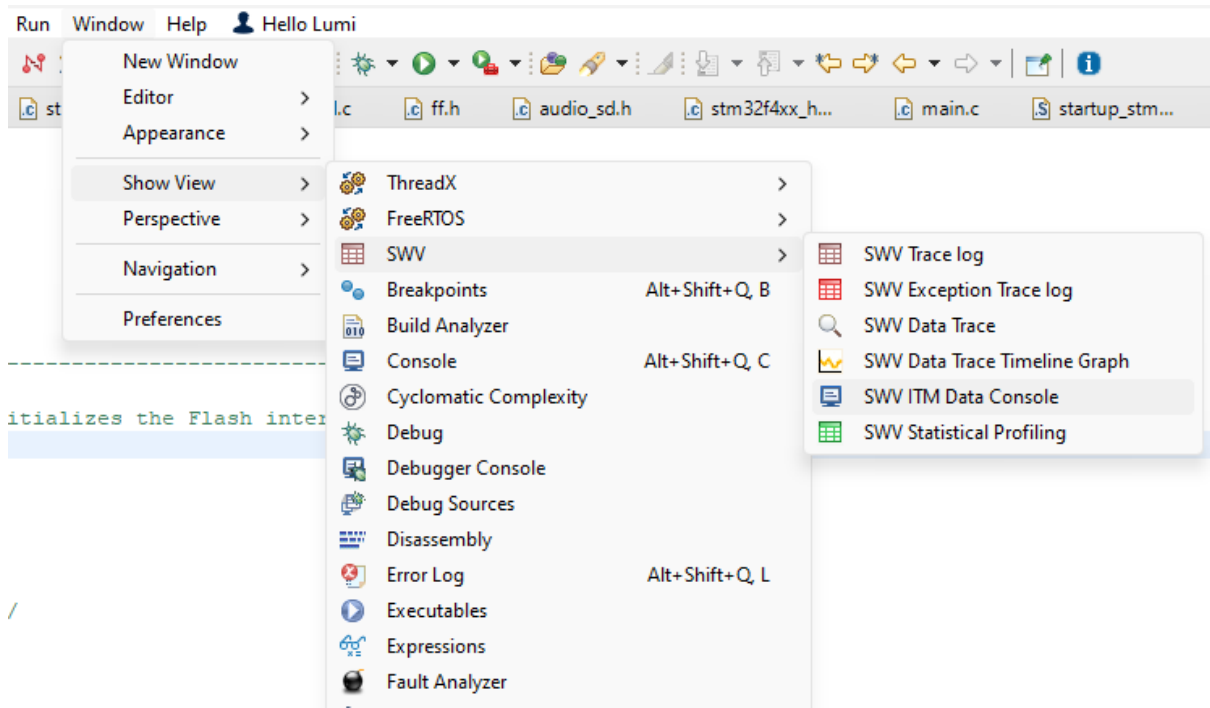


Figure 29:

- Trong cửa sổ SWV settings, trong mục ITM Stimulus Ports ,enable port 0 -> OK.

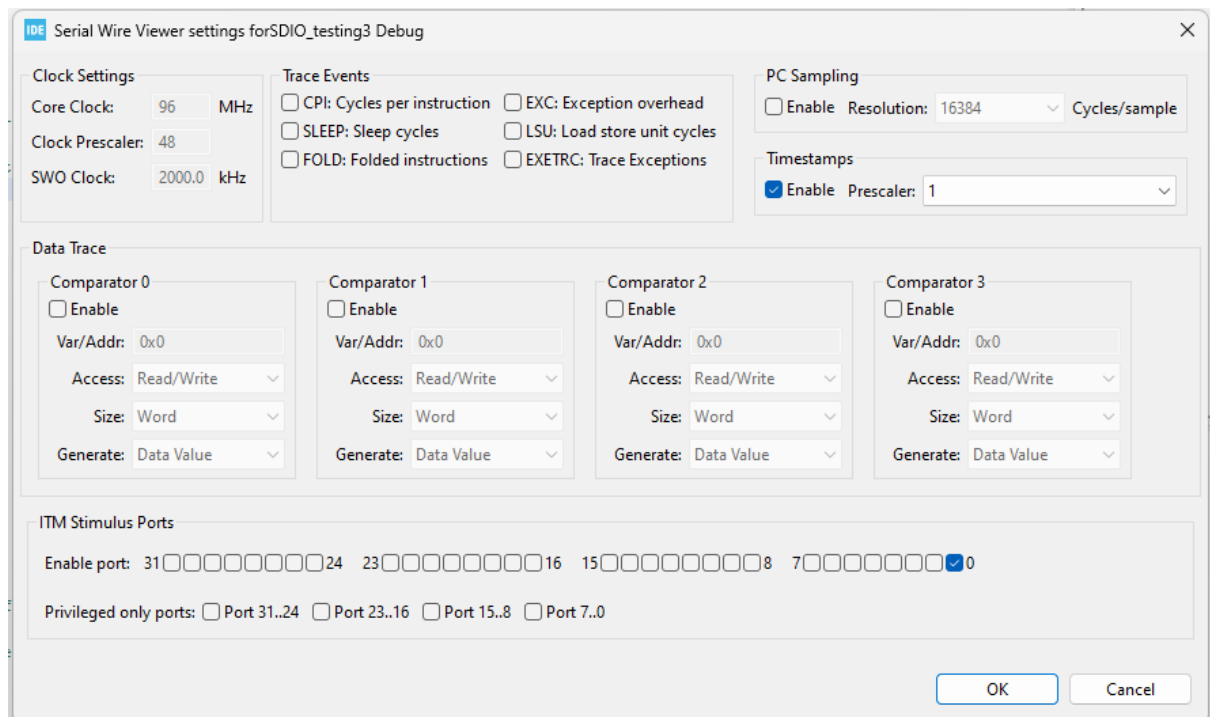


Figure 30:

- Nhấn nút Start Trace.

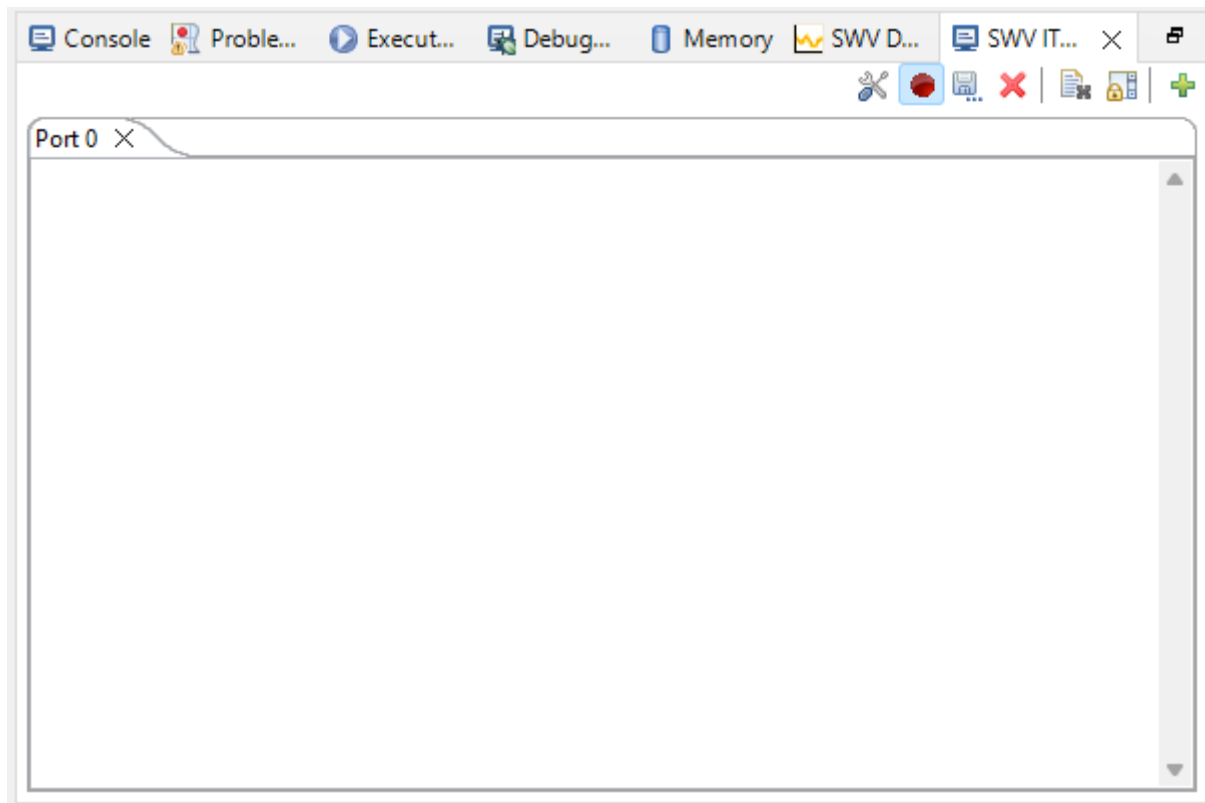


Figure 31:

- Bước 10: Debug chương trình
 - Nhấn Resume (F8) để chương trình thực thi. Bạn có thể dùng Step Into, Step Over, and Step Return để nhảy từng lệnh, nhảy vào hay nhảy ra từng hàm.
 - Quan sát SWV ITM console.
 - Kích chọn bảng Live Expressions. Sau đó kích vào Add new expression, gõ vào tên biến, mảng để theo dõi sự thay đổi giá trị trong quá trình Debug.
 - Sửa lỗi nếu cần thiết.

6. Mã code.

CODE 1: tạo file .txt và mở file để ghi và đọc dữ liệu

```

/*-----*/ /* main.c: Main program body
*/ /*-----*/
/* Includes -----*/
#include "main.h"
#include "fatfs.h"

/* Private variables -----*/

FATFS mynewdiskFatFs; /* File system object for User logical drive */
FIL MyFile; /* File object */
char mynewdiskPath[4]; /* User logical drive path */
uint32_t wbytes; /* File write counts */

```

```

uint8_t wtext[] = "text to write logical disk"; /* File write buffer */
uint8_t receive_arr[50];

/* USER CODE BEGIN 0 */
void writeData()
{
    /* ghi du lieu vao the nho voi ten file "STM32.TXT" voi noi dung file duoc luu trong bien
    wtext*/
    if(f_mount(&mynewdiskFatFs, (TCHAR const*)mynewdiskPath, 0) == FR_OK)
    {
        if(f_open(&MyFile, "STM32.TXT", FA_CREATE_ALWAYS | FA_WRITE)
        == FR_OK)
        {
            if(f_write(&MyFile, wtext, sizeof(wtext), (void *)&wbytes) ==
            FR_OK)
            {
                f_close(&MyFile);
            }
        }
    }
}

void readData()
{
    /* doc du lieu trong the nho co ten file "STM32.TXT" va luu du lieu vao bien receive_arr*/
    if(f_mount(&mynewdiskFatFs, (TCHAR const*)mynewdiskPath, 0) == FR_OK)
    {
        if(f_open(&MyFile, "STM32.TXT", FA_READ) == FR_OK)
        {
            if(f_read(&MyFile, &receive_arr, f_size(&MyFile), (void *)&wbytes)
            == FR_OK)
            {
                f_close(&MyFile);
            }
        }
    }
}

/* USER CODE END 0 */

int main(void)
{

```

```

/* USER CODE BEGIN 2 */
writeData(); // gọi hàm writeData để ghi dữ liệu vào file
HAL_delay(1000); // delay 1 s
readData(); // gọi hàm readData để đọc dữ liệu từ file
/* USER CODE END 2 */
}

```

CODE 2:

```

/*-----*/ /* main.c: Main program body
*/ /*-----*/
/* Includes -----*/
#include "main.h"
#include "fatfs.h"

/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <string.h>
/* USER CODE END Includes */

/* Private variables-----*/
FATFS FatFs;
FIL Fil;
FRESULT FR_Status;
FATFS *FS_Ptr;
UINT RWC, WWC; // Read/Write Word Counter
DWORD FreeClusters;
uint32_t TotalSize, FreeSpace;
char RW_Buffer[200];

/* main() function-----*/
int main(void)
{

    HAL_Delay(1000);
    do
    {
        //-----[ Mount The SD Card ]-----
        FR_Status = f_mount(&FatFs, "", 1);
        if (FR_Status != FR_OK)
        {
            printf("Error! While Mounting SD Card, Error Code: (%i)\r\n", FR_Status);
            break;
        }
    }
}

```

```

printf("SD Card Mounted Successfully! \r\n\n");
//---[ Get & Print The SD Card Size & Free Space]-----
f_getfree("", &FreeClusters, &FS_Ptr);
TotalSize = (uint32_t)((FS_Ptr->n_fatent - 2) * FS_Ptr->csize * 0.5);
FreeSpace = (uint32_t)(FreeClusters * FS_Ptr->csize * 0.5);
printf("Total SD Card Size: %lu Bytes\r\n", TotalSize);
printf("Free SD Card Space: %lu Bytes\r\n\n", FreeSpace);

//--[ Open A Text File For Write & Write Data ]-----
//Open the file

FR_Status = f_open(&Fil, "STM32.TXT", FA_CREATE_ALWAYS | FA_WRITE);
if(FR_Status != FR_OK)
{
    printf("Error! While Creating/Opening A New Text File, Error Code: (%i)\r\n",
FR_Status);
    break;
}
printf("Text File Created & Opened! Writing Data To The Text File..\r\n\n");
// (1) Write Data To The Text File [ Using f_puts() Function ]
f_puts("Hello! From STM32 To SD Card Over SDIO, Using f_puts()\n", &Fil);
// (2) Write Data To The Text File [ Using f_write() Function ]
strcpy(RW_Buffer, "Hello! From STM32 To SD Card Over SDIO, Using f_write()\r\n");
f_write(&Fil, RW_Buffer, strlen(RW_Buffer), &WWC);
// Close The File
f_close(&Fil);

//--[ Open A Text File For Read & Read Its Data]-----

// Open The File

FR_Status = f_open(&Fil, "STM32.TXT", FA_READ);
if(FR_Status != FR_OK)
{
    printf("Error! While Opening (STM32.TXT) File For Read.. \r\n");
    break;
}
// (1) Read The Text File's Data [ Using f_gets() Function ]
f_gets(RW_Buffer, sizeof(RW_Buffer), &Fil);
printf("Data Read From (STM32.TXT) Using f_gets():%s", RW_Buffer);
// (2) Read The Text File's Data [ Using f_read() Function ]
f_read(&Fil, RW_Buffer, f_size(&Fil), &RWC);
printf("Data Read From (STM32.TXT) Using f_read():%s", RW_Buffer);
// Close The File

```

```

f_close(&Fil);
printf("File Closed! \r\n\n");

//-[ Open An Existing Text File, Update Its Content, Read It Back]-
// (1) Open The Existing File For Write (Update)
FR_Status = f_open(&Fil, "STM32.TXT", FA_OPEN_EXISTING | FA_WRITE);
FR_Status = f_lseek(&Fil, f_size(&Fil)); // Move The File Pointer To The EOF (End-Of-
File)
if(FR_Status != FR_OK)
{
    printf("Error! While Opening (STM32.TXT) File For Update.. \r\n");
    break;
}
// (2) Write New Line of Text Data To The File
FR_Status = f_puts("This New Line Was Added During File Update!\r\n", &Fil);
f_close(&Fil);
memset(RW_Buffer, '\0', sizeof(RW_Buffer)); // Clear The Buffer
// (3) Read The Contents of The Text File After The Update
FR_Status = f_open(&Fil, "STM32.TXT", FA_READ); // Open The File For Read
f_read(&Fil, RW_Buffer, f_size(&Fil), &RWC);
printf("Data Read From (STM32.TXT) After Update:\r\n%s", RW_Buffer);
f_close(&Fil);

//-----[ Delete The Text File ]-----
// Delete The File
/*
FR_Status = f_unlink(STM32.TXT);
if (FR_Status != FR_OK){
    printf("Error! While Deleting The (STM32.TXT) File.. \r\n");
}
*/

//-----[ Create folder ]-----

FR_Status = f_mkdir("/MYFOLDER\0");
if(FR_Status != FR_OK)
{
    printf("Error! While Creating/Opening A New Folder, Error Code:
(%i)\r\n", FR_Status);
    break;
}

FR_Status = f_open(&Fil, "/MYFOLDER/STM32.TXT",
FA_CREATE_ALWAYS | FA_WRITE);
if(FR_Status != FR_OK)

```

```

        {
            printf("Error! While Creating/Opening A New Text File in folder,
Error Code: (%i)\r\n", FR_Status);
            break;
        }
        //Write data into the file
        printf("Text File in Folder Created & Opened! Writing Data To The
Text File..\r\n\n");
        // (1) Write Data To The Text File [ Using f_puts() Function ]
        f_puts("Hello! From STM32 To SD Card Over SDIO, Using
f_puts(), file in Folder \n", &Fil);
        // (2) Write Data To The Text File [ Using f_write() Function ]
        strcpy(RW_Buffer, "Hello! From STM32 To SD Card Over SDIO,
Using f_write(), file in Folder\r\n");
        f_write(&Fil, RW_Buffer, strlen(RW_Buffer), &WWC);
        // Close The File
        f_close(&Fil);

    } while(0);

//-----[ Test Complete! Unmount The SD Card ]-----

FR_Status = f_mount(NULL, "", 0);
if (FR_Status != FR_OK)
{
    printf("\r\nError! While Un-mounting SD Card, Error Code: (%i)\r\n", FR_Status);
} else{
    printf("\r\nSD Card Un-mounted Successfully! \r\n");
}
while (1)
{
    // Nothing To Do Here!
}

}

/* -----[ Private function ]-----*/
int _write(int file, char *ptr, int len)
{
    int DataIdx;
    for (DataIdx = 0; DataIdx < len; DataIdx++)
    {
        ITM_SendChar(*ptr++);
    }
}

```

```
    return len;  
}
```

7. Kết quả

CODE 1:

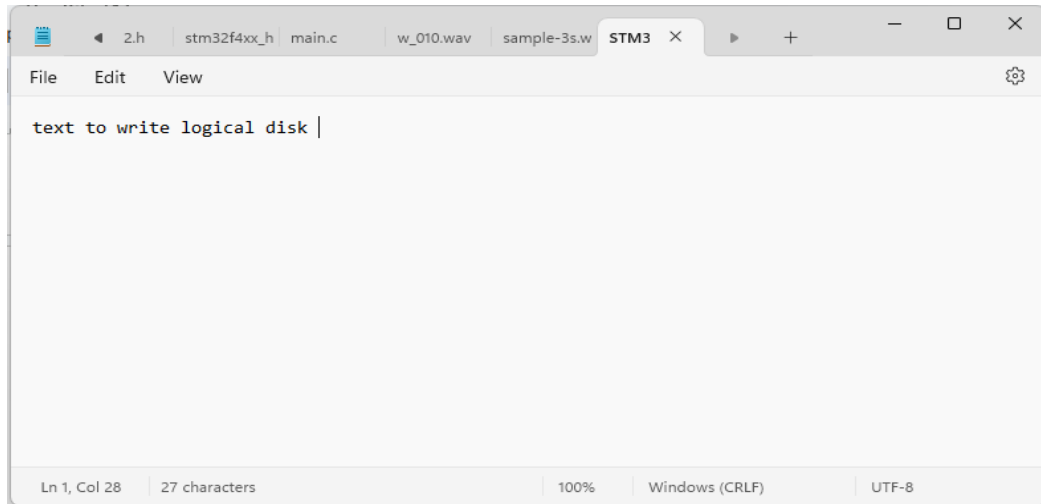


Figure 32:


Name	Date modified	Type	Size
 STM32.TXT		Text Document	1 KB

Figure 33:

(x)= Variables Breakpoints Expressions Registers Live Expressions SFRs		
Expression	Type	Value
receive_arr	uint8_t [50]	[50]
receive_arr[0]	uint8_t	116 't'
receive_arr[1]	uint8_t	101 'e'
receive_arr[2]	uint8_t	120 'x'
receive_arr[3]	uint8_t	116 't'
receive_arr[4]	uint8_t	32 ' '
receive_arr[5]	uint8_t	116 't'
receive_arr[6]	uint8_t	111 'o'
receive_arr[7]	uint8_t	32 ' '
receive_arr[8]	uint8_t	119 'w'
receive_arr[9]	uint8_t	114 'r'
receive_arr[10]	uint8_t	105 'i'
receive_arr[11]	uint8_t	116 't'
receive_arr[12]	uint8_t	101 'e'
receive_arr[13]	uint8_t	32 ' '
receive_arr[14]	uint8_t	108 'l'
receive_arr[15]	uint8_t	111 'o'
receive_arr[16]	uint8_t	103 'g'
receive_arr[17]	uint8_t	105 'i'
receive_arr[18]	uint8_t	99 'c'
receive_arr[19]	uint8_t	97 'a'
receive_arr[20]	uint8_t	108 'l'
receive_arr[21]	uint8_t	32 ' '
receive_arr[22]	uint8_t	100 'd'
receive_arr[23]	uint8_t	105 'i'
receive_arr[24]	uint8_t	115 's'
receive_arr[25]	uint8_t	107 'k'
receive_arr[26]	uint8_t	0 '\000'
receive_arr[27]	uint8_t	0 '\000'
receive_arr[28]	uint8_t	0 '\000'
receive_arr[29]	uint8_t	0 '\000'

Figure 34

CODE2:

```
Port 0 X
SD Card Mounted Successfully!

Total SD Card Size: 15549952 Bytes
Free SD Card Space: 15549760 Bytes

Text File Created & Opened! Writing Data To The Text File..

Data Read From (STM32.TXT) Using f_gets():Hello! From STM32 To SD Card Over
SDIO, Using f_puts()
Data Read From (STM32.TXT) Using f_read():Hello! From STM32 To SD Card Over
SDIO, Using f_write()
File Closed!

Data Read From (STM32.TXT) After Update:
Hello! From STM32 To SD Card Over SDIO, Using f_puts()
Hello! From STM32 To SD Card Over SDIO, Using f_write()
This New Line Was Added During File Update!

SD Card Un-mounted Successfully!
```

Figure 35


```
Console Problems Executables SWV ITM Data Console x Memory
Port 0 x
SD Card Mounted Successfully!

Total SD Card Size: 31154688 Bytes
Free SD Card Space: 31154432 Bytes

Text File Created & Opened! Writing Data To The Text File..

Data Read From (STM32.TXT) Using f_gets():Hello! From STM32 To SD Card Over SDIO, Using f_puts()
Data Read From (STM32.TXT) Using f_read():Hello! From STM32 To SD Card Over SDIO, Using f_write()
File Closed!

Data Read From (STM32.TXT) After Update:
Hello! From STM32 To SD Card Over SDIO, Using f_puts()
Hello! From STM32 To SD Card Over SDIO, Using f_write()
This New Line Was Added During File Update!

Text File in Folder Created & Opened! Writing Data To The Text File..

SD Card Un-mounted Successfully!
```

Figure 36:

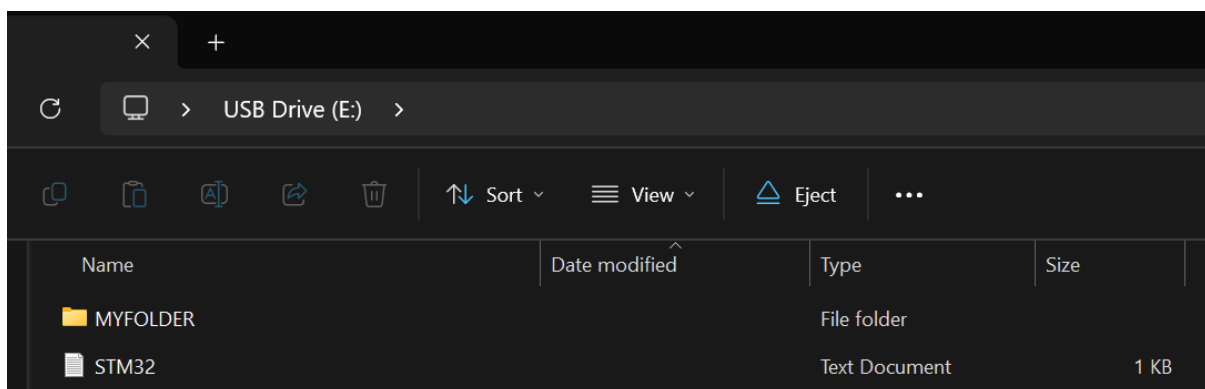


Figure 37:

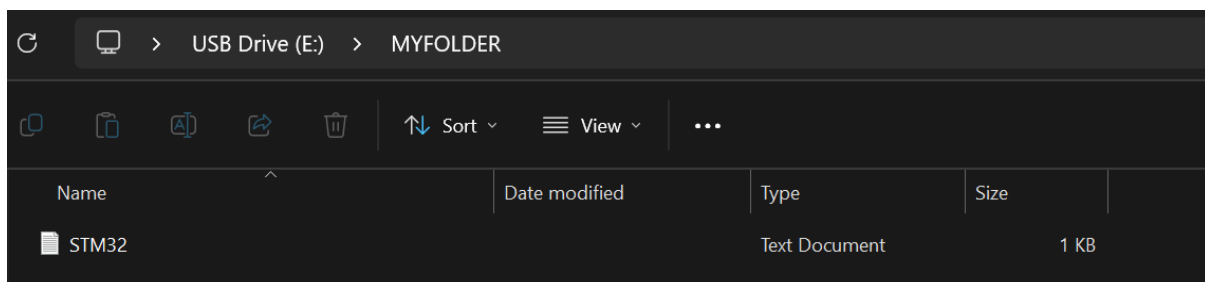


Figure 38:

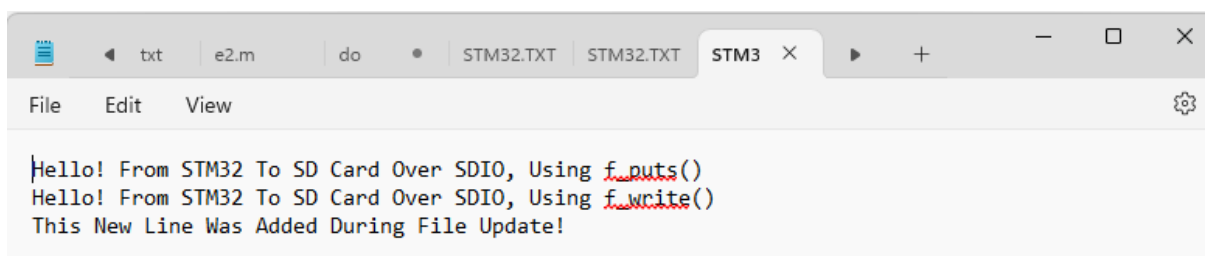


Figure 39:

8. Một số lỗi

- Lỗi in chữ trung quốc:



Figure 40:

Nếu trong console view hiện như thế này, thì đó là có thể là do mình cấu hình tần số trong debug configuration sai. Vì bo mạch STM32F411E chỉ hỗ trợ xung tối đa là 100MHz nên nếu cấu hình trên 100MHz sẽ gặp lỗi. Ở đây mình cấu hình xung là 96MHz.

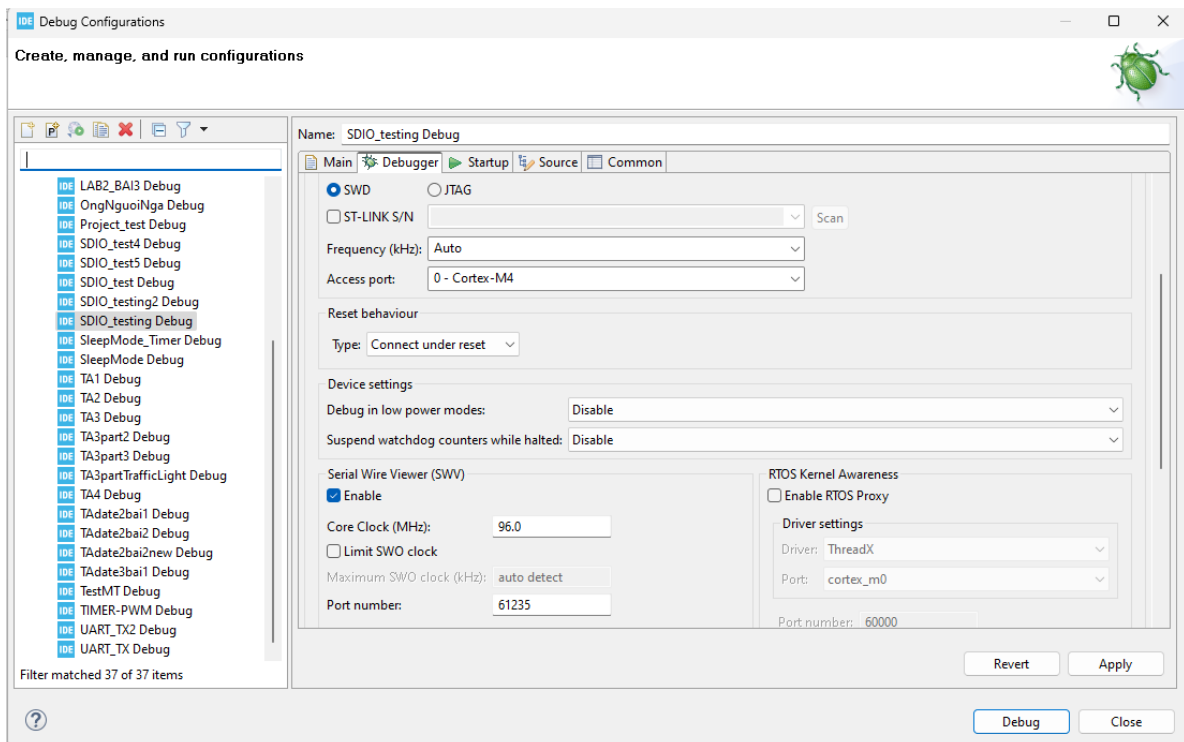


Figure 41:

- **Lỗi FR_NOT_READY (3):**

Sau mỗi thao tác, chương trình sẽ trả về mã kết quả loại enum FRESULT. Khi một hàm API thao tác thành công, chương trình sẽ trả về số 0 (FR_OK).

Expression	Type	Value
(0)= FR_Status	FRESULT	FR_OK
+ Add new expression		

Figure 42:

Nếu gặp lỗi, chương trình trả về giá trị số khác với số 0 và cho biết loại lỗi tương ứng. Bạn có thể tra bảng mã kết quả trả về của thư viện Fatfs bằng cách nhấn FRESULT -> Open Declaration.

```

48  /* USER CODE BEGIN PV */
49  FATFS FatFs;
50  FIL Fil;
51  FRESULT
52  FATFS *
53  UINT RW
54  DWORD F
55  uint32_
56  char RW
57  char Tx
58
59
60  /* USER
61
62  /* Priv
63  void Sy
64  static
65  static
66  /* USER
67

```

Undo
Ctrl+Z

Revert File

Save
Ctrl+S

Open Declaration
F3

Open Type Hierarchy
F4

Open Call Hierarchy
Ctrl+Alt+H

Quick Outline
Ctrl+O

Quick Type Hierarchy
Ctrl+T

Explore Macro Expansion
Ctrl+#

Toggle Source/Header
Ctrl+Tab

Open With
>

Show In
Alt+Shift+W >

```

typedef enum {
    FR_OK = 0, /* (0) Succeeded */
    FR_DISK_ERR, /* (1) A hard error occurred in the low level disk I/O layer */
    FR_INT_ERR, /* (2) Assertion failed */
    FR_NOT_READY, /* (3) The physical drive cannot work */
    FR_NO_FILE, /* (4) Could not find the file */
    FR_NO_PATH, /* (5) Could not find the path */
    FR_INVALID_NAME, /* (6) The path name format is invalid */
    FR_DENIED, /* (7) Access denied due to prohibited access or directory full */
    FR_EXIST, /* (8) Access denied due to prohibited access */
    FR_INVALID_OBJECT, /* (9) The file/directory object is invalid */
    FR_WRITE_PROTECTED, /* (10) The physical drive is write protected */
    FR_INVALID_DRIVE, /* (11) The logical drive number is invalid */
    FR_NOT_ENABLED, /* (12) The volume has no work area */
    FR_NO_FILESYSTEM, /* (13) There is no valid FAT volume */
    FR_MKFS_ABORTED, /* (14) The f_mkfs() aborted due to any problem */
    FR_TIMEOUT, /* (15) Could not get a grant to access the volume within defined period */
    FR_LOCKED, /* (16) The operation is rejected according to the file sharing policy */
    FR_NOT_ENOUGH_CORE, /* (17) LFN working buffer could not be allocated */
    FR_TOO_MANY_OPEN_FILES, /* (18) Number of open files > _FS_LOCK */
    FR_INVALID_PARAMETER /* (19) Given parameter is invalid */
} FRESULT;

```

Figure 43:

Ví dụ sau đây là khi thao tác với hàm `f_mount`, chương trình báo lỗi không thực hiện được và trả về Error Code: (3) tương ứng với lỗi `FR_NOT_READY`. Lỗi này có nghĩa là khả năng module SDIO gặp vấn đề về mặt vật lý (nổi nhâm hoặc lỏng dây, thẻ nhớ không đúng format,...). Ở đây mình cố tình không cắm thẻ nhớ vào module SDIO nên chương trình báo lỗi (3) này là chính xác. Khi gặp lỗi này bạn nên kiểm tra lại phần cứng bo mạch của mình.

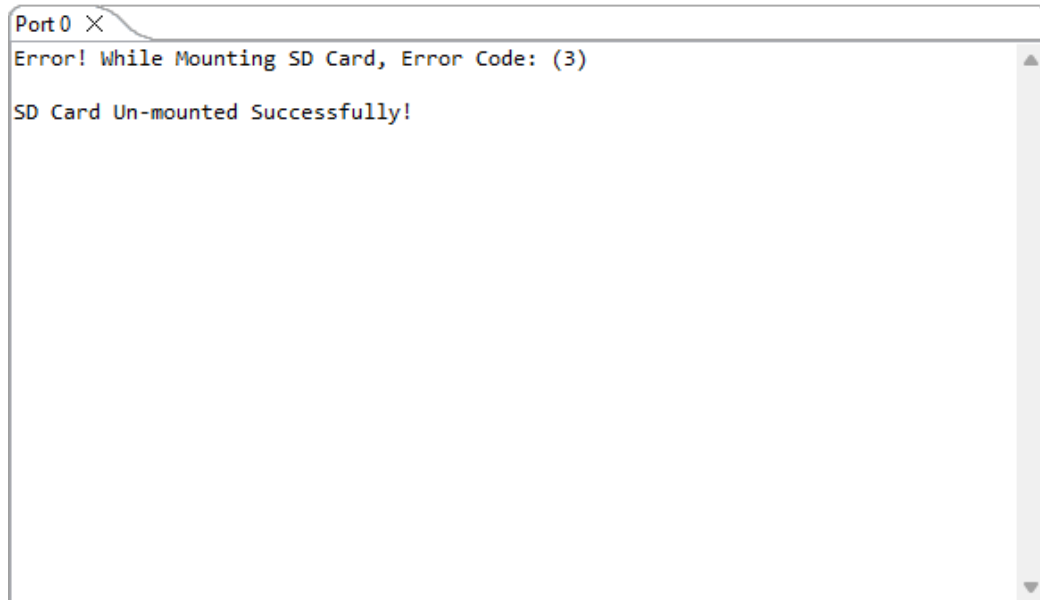


Figure 44:

Lỗi `FR_NOT_READY` cũng báo trong một số trường hợp đặt biệt của module thẻ nhớ SDIO. Bạn cấu hình tất cả các chân GPIO của khối SDIO đều ở chế độ Pull-up trừ `SDIO_CK` thì vẫn để No pull-up, pull-down.

Search Signals							
Search (Ctrl+F)				<input type="checkbox"/> Show only Modified Pins			
Pin Na...	Signal on ...	GPIO outp...	GPIO mode	GPIO Pull-...	Maximum ...	User Label	Modified
PA6	SDIO_CMD	n/a	Alternate ...	Pull-up	Very High		✓
PB15	SDIO_CK	n/a	Alternate ...	No pull-up ...	Very High		✓
PC8	SDIO_D0	n/a	Alternate ...	Pull-up	Very High		✓

Figure 45:

- **Lỗi `FR_DISK_ERR` (1):**
Một lỗi nữa cũng thường gặp là lỗi (1) `FR_DISK_ERR`. Lỗi này xảy ra khi các layer `disk_read`, hàm `disk_write` hoặc `disk_ioctl` gặp vấn đề. Chương trình báo lỗi này là khả năng cao là do chip flash memory của bo mạch xung đột hoặc biến File tương ứng với 1 file trong thẻ nhớ đã và thao tác nhưng

chưa được đóng. Cách giải quyết là format lại thẻ nhớ microSD và xóa bộ nhớ flash bằng phần mềm STM32CubeProgrammer.

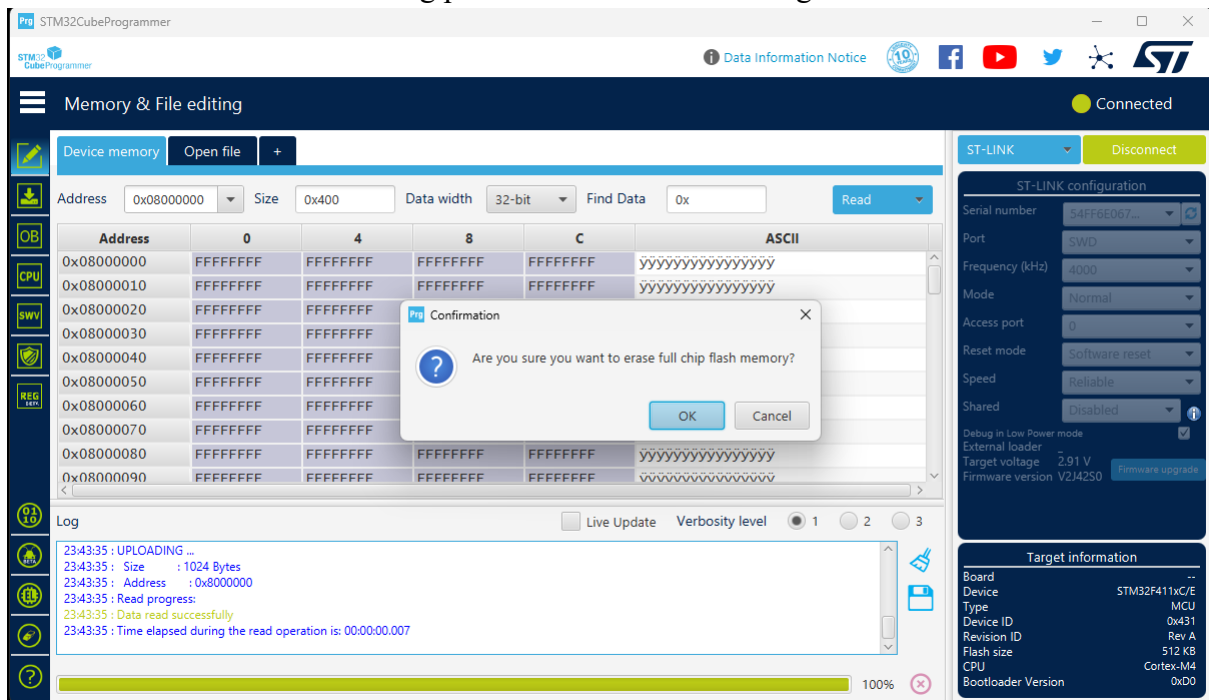


Figure 46:

Cũng có khả năng là do tốc độ đọc thẻ tối đa của thẻ SDIO thấp hơn 24MHz. Bạn có thể làm điều chỉnh bằng cách tăng hệ số CLKDIV nếu bạn gặp lỗi trong khi quá trình gắn thẻ SD.

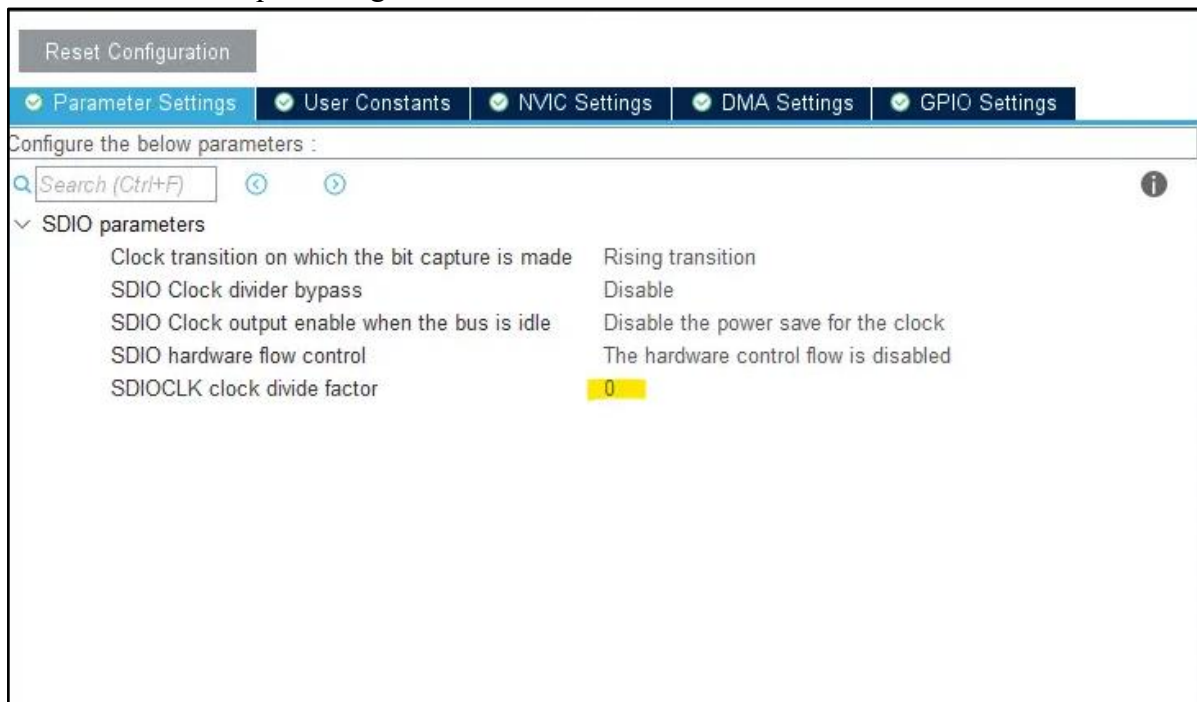


Figure 47:

Tài liệu tham khảo:

<http://elm-chan.org/fsw/ff/>

https://www.st.com/resource/en/user_manual/um1721-developing-applications-on-stm32cube-with-fatfs-stmicroelectronics.pdf
<https://deepbluembedded.com/stm32-sdio-dma-example/>

Chương

21:

https://www.st.com/resource/en/reference_manual/rm0383-stm32f411xce-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
https://www.st.com/resource/en/product_training/stm32l4_peripheral_sdmmc.pdf
[Physical-Layer-Simplified-SpecificationV6.0.pdf \(taterli.com\)](#)