Nhóm tác giả:

Trần Hoàng Minh

Nguyễn Thị Tâm

Võ Văn Bưu

Lương Như Quỳnh

Hướng dẫn và kiểm tra

Nguyễn Huỳnh Nhật Thương

Nguyễn Quang Phương

Contents

A. Audio

1. Tổng quan digital micro

a. Công nghệ làm digital microphoness

- Hai trong số các công nghệ được sử dụng phổ biến nhất trong xây dựng micrô là Microphone MEMS (Micro-Electro-Mechanical Systems) và microphone điện cực electret (Electret Condenser Microphone - ECM).
- Microphone ECM, hay microphone điện cực electret, là một loại microphone phổ biến trong nhiều ứng dụng âm thanh. Nó hoạt động dựa trên nguyên lý điện dung, trong đó một lớp điện môi nằm giữa hai bản cực. Một trong các bản cực, thường được làm từ vật liệu electret, có khả năng tích điện. Khi sóng âm đến, áp lực âm thanh làm cho bản cực di động, gây ra sự thay đổi điện tích giữa hai bản cực. Điều này tạo ra tín hiệu điện tương ứng với âm thanh. ECM có độ nhạy cao và tần số đáp ứng rộng, giúp tái tạo âm thanh với chất lượng tốt, đặc biệt trong các ứng dụng ghi âm chuyên nghiệp, hội thảo và hệ thống âm thanh. Một số module âm thanh phổ biến như MAX4466, MAX9814.

Figure 1,2:

- Microphone MEMS là một công nghệ tiên tiến bao gồm một cảm biến (MEMS) và một mạch tích hợp chuyên dụng (ASIC) trong một gói đơn trên chip silicon. Với kích thước nhỏ gọn, MEMS thường được tích hợp trong các thiết bị như smartphone, tai nghe và thiết bị IoT. Nguyên lý hoạt động của MEMS dựa trên sự thay đổi áp suất do âm thanh tác động lên cấu trúc vi mô bên trong chip. Điều này tạo ra tín hiệu điện, thường đi kèm với mạch khuếch đại tích hợp để cải thiện chất lượng âm thanh. MEMS có ưu điểm là chi phí sản xuất thấp và dễ dàng sản xuất hàng loạt, nhưng chất lượng âm thanh có thể không bằng

ECM. Tuy nhiên, với sự phát triển công nghệ, microphone MEMS ngày càng trở nên phổ biến và cải thiện về độ nhạy và chất lượng âm thanh.

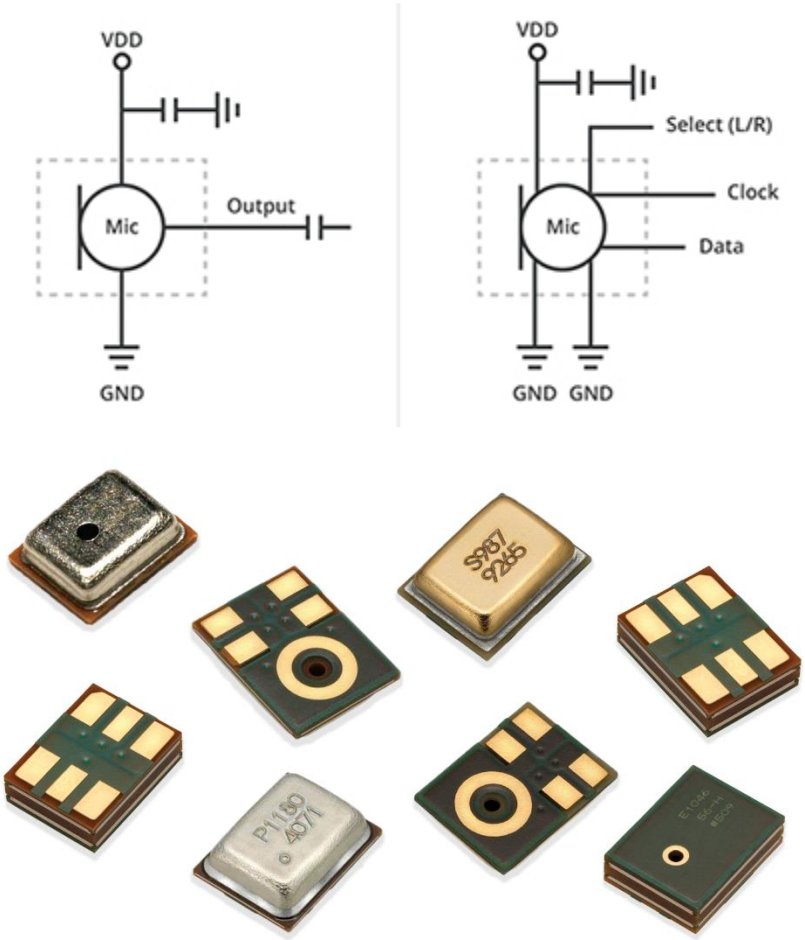


Figure 3,4:

b. MEMS microphone

- Nguyên lý hoạt động của MEMS microphone là phần cảm biến sẽ chuyển đổi áp suất âm thanh biến đổi thành các biến thiên điện dung mà ASIC chuyển đổi thành tín hiệu ở đầu ra là tín hiệu tương tự hoặc tín hiệu số. Sóng âm thanh đi vào microphone qua một lỗ âm thanh ở trên hoặc dưới module, nên sẽ có 2 loại MEMS microphone chính là microphone cổng trên (top port) và microphone cổng dưới (bottom port). Microphone cổng dưới sẽ có nhiều ưu điểm hơn như là buồng sau lớn hơn, cải thiện tỷ lệ tín hiệu trên tạp âm (SNR) và cho phép đáp ứng tần số phẳng (flat frequency response). Microphone cổng trên sẽ dễ hàn PCB và thiết kế vòng đệm hơn.

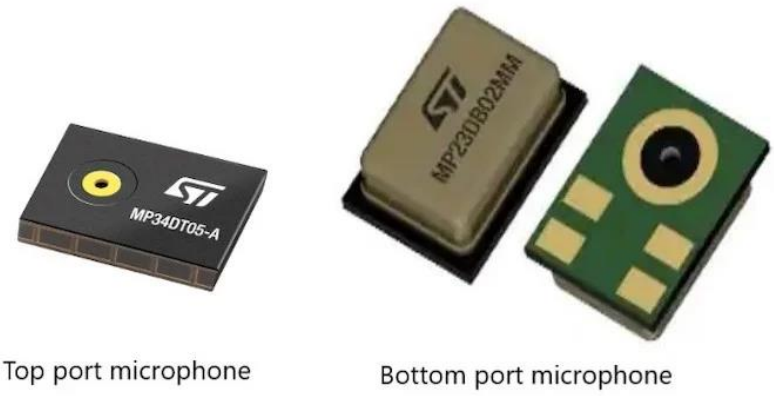


Figure 5,6:

c. Tín hiệu tương tự và tín hiệu số

- Micro MEMS với đầu ra là tín hiệu tương tự có mức tiêu thụ điện năng thấp hơn và cấu trúc ASIC nhỏ gọn hơn. Micro MEMS kỹ thuật số sẽ có chức năng ADC chuyển đổi tín hiệu tương tự thành tín hiệu PDM. Do đó tín hiệu sẽ được truyền đi mạnh mẽ hơn và lọc bớt nhiễu trường điện từ (EMI).

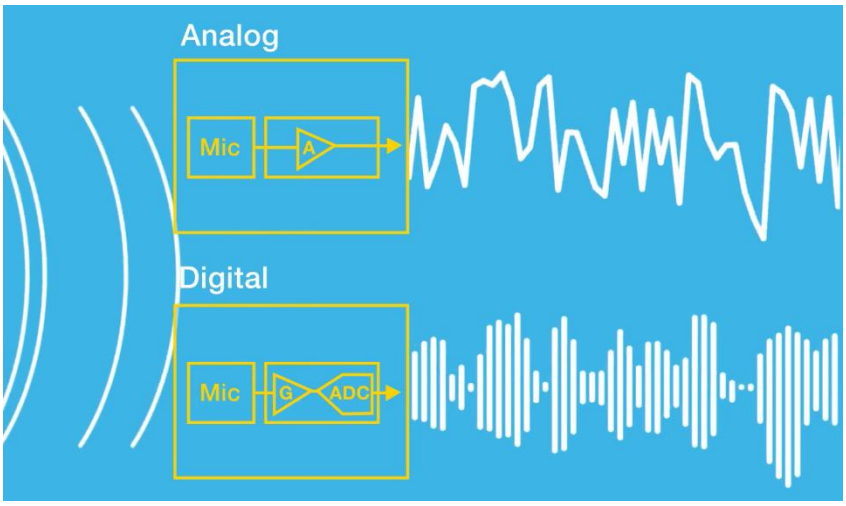


Figure 7:

2. Tập tin WAV

File .wav là một loại tệp âm thanh phổ biến, xuất hiện lần đầu trên hệ điều hành Windows. Phần mở rộng này là viết tắt của "waveform". File .wav hỗ trợ nhiều kích thước âm thanh, tần số lấy mẫu và kênh âm thanh. Đặc điểm của nó là ghi lại các dạng sóng âm thanh tự nhiên mà không nén dữ liệu và có kích thước dữ liệu lớn. Chất lượng của âm thanh được khôi phục từ file .wav phụ thuộc vào kích thước của mẫu và tần số lấy mẫu âm thanh. Tần số lấy mẫu càng cao, chất lượng càng tốt nhưng kéo theo kích thước cũng lớn hơn.:

Tệp WAVE có một đoạn “WAVE” duy nhất bao gồm hai đoạn phụ:

- Đoạn “fmt” - chỉ định định dạng dữ liệu.
- Một đoạn “dữ liệu” - chứa dữ liệu mẫu thực tế.

Một số khái niệm cần nắm:

- **Tần số lấy mẫu:** số lượng mẫu âm thanh ghi lại trong mỗi giây, các tần số thường gặp như 8000, 12000, 16000, 22050, 44100… đơn vị là Hz. Tần số lấy mẫu càng cao thì chất lượng âm thanh càng tốt.
- **Số lượng bit lấy mẫu** (độ chính xác lấy mẫu theo biên độ)**:** là giá trị lấy mẫu, dùng để đo dao động của âm thanh, giá trị càng lớn thì độ phân giải càng cao, âm thanh tạo ra càng mạnh. Chúng ta thường gặp loại 8-bit và 16 bit.
- **Số kênh:** số kênh như kiểu mono (đơn âm) tức là chúng ta chỉ có thể nghe bằng một loa; kiểu stereo (âm thanh nổi) có thể phát ra hai loa cùng lúc.

Tiêu đề của tệp WAV (RIFF) dài 44-byte từ 0 tới 43 và có định dạng như ví dụ sau:

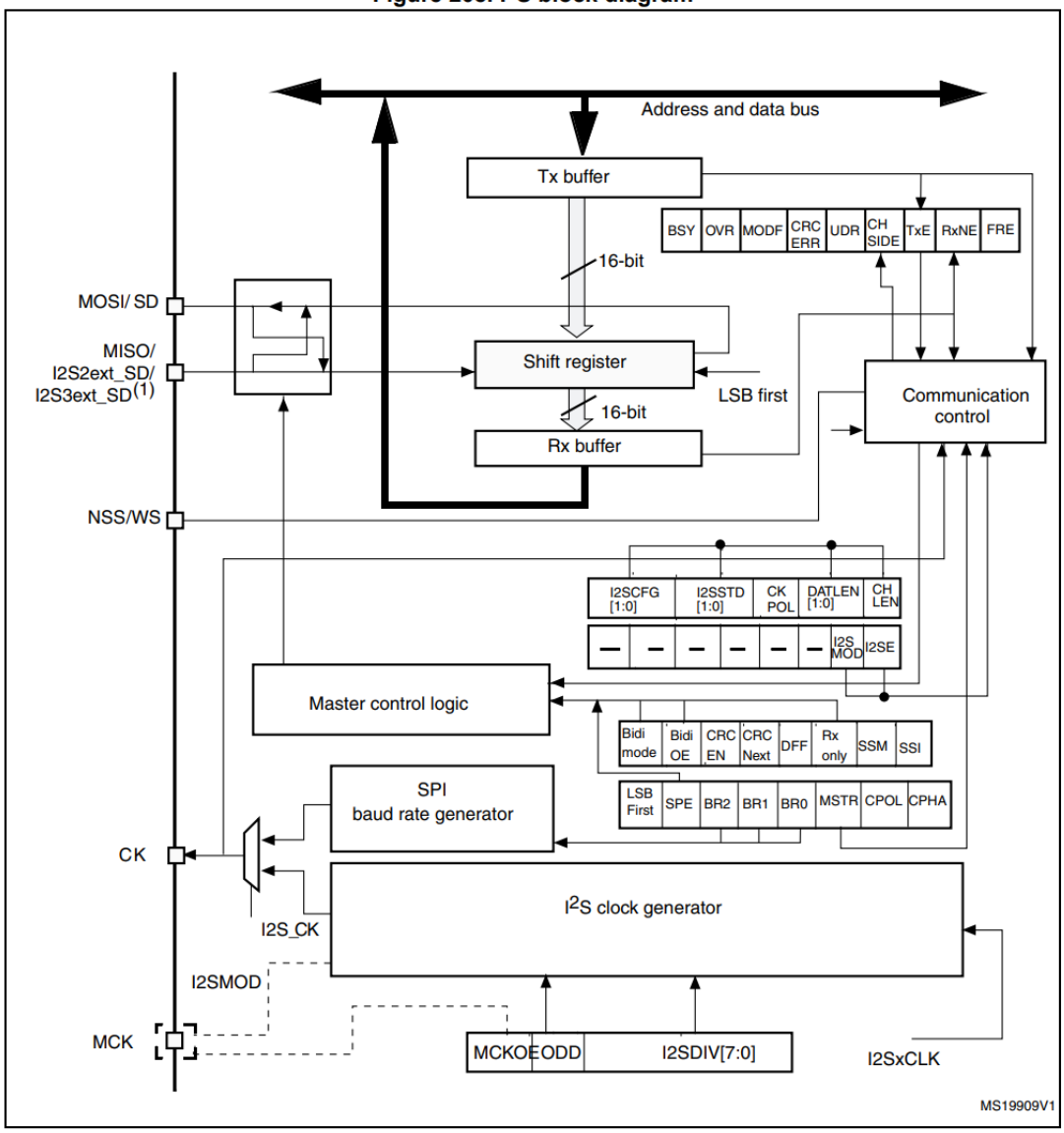
Vị trí	Giá trị mẫu	Mô tả
0 - 3	“RIFF”	Đánh dấu tệp là tệp riff. Mỗi ký tự dài 1 byte.
4 - 7	Kích thước tệp (số nguyên)	Kích thước của toàn bộ tệp - 8 byte, tính bằng byte (số nguyên 32 bit). Thông thường, bạn sẽ điền thông tin này sau khi tạo.
8 - 11	“WAVE”	Tiêu đề loại tệp. Đối với mục đích của chúng tôi, nó luôn bằng “WAVE”.
12 - 15	“fmt "	Định dạng đoạn đánh dấu. Bao gồm dấu null
16 - 19	16	Độ dài của dữ liệu định dạng như được liệt kê ở trên
20 - 21	1	Loại định dạng (1 là PCM) - 2 byte số nguyên
22 - 23	2	Số kênh - Số nguyên 2 byte
24 - 27	44100	Tốc độ mẫu - số nguyên 32 byte. Các giá trị phổ biến là 44100 (CD), 48000 (DAT). Tỷ lệ mẫu = Số lượng mẫu mỗi giây, hoặc Hertz.
28 - 31	176400	(Tỷ lệ mẫu * BitsPerSample * Kênh) / 8.

32 - 33	4	(BitsPerSample * Channels) / 8.1 - 8 bit mono2 - 8 bit stereo/16 bit mono4 - 16 bit stereo
34 - 35	16	Số bit trên mỗi mẫu
36 - 39	“dữ liệu”	“dữ liệu” tiêu đề chunk. Đánh dấu phần đầu của phần dữ liệu.
40 - 43	Kích thước tệp (dữ liệu)	Kích thước của phần dữ liệu.
Các giá trị mẫu được cung cấp ở trên cho nguồn âm thanh nổi 16 bit.		

3. Giao thức I2S

a. Khối I2S trong STM32F411

Figure 208. I²S block diagram



1. I2S2ext_SD and I2S3ext_SD are the extended SD pins that control the I²S full duplex mode.

Figure 8: Cấu trúc khối I2S trong STM32F411.

The SPI could function as an audio I2S interface when the I2S capability is enabled (by setting the I2SMOD bit in the SPI_I2SCFGR register). This interface uses almost the same pins, flags and interrupts as the SPI.

- Trong STM32F411 chúng ta có 5 khối I2S (I2S1-I2S5). Mỗi khối có các chân kết nối như sau:

- SD: Dữ liệu nối tiếp (được ánh xạ trên chân MOSI) để truyền hoặc nhận hai kênh dữ liệu ghép kênh thời gian (chỉ có chế độ half-duplex).
- WS: Chọn từ (được ánh xạ trên chân NSS) là tín hiệu điều khiển dữ liệu đầu ra ở chế độ Master và đầu vào ở chế độ Slave.
- CK: Clock nối tiếp (được ánh xạ trên chân SCK) là đầu ra clock nối tiếp ở chế độ Master và đầu vào clock nối tiếp ở chế độ Slave.
- I2S2ext_SD và I2S3ext_SD: các chân bổ sung (được ánh xạ trên chân MISO) để điều khiển chế độ full duplex.

- Có thể sử dụng một chân bổ sung khi cần master clock cho một số thiết bị âm thanh bên ngoài.
  - MCK: Master Clock (được ánh xạ riêng) được sử dụng khi I2S được cấu hình ở chế độ chính (và khi bit MCKOE trong thanh ghi SPI_I2SPR được đặt), để xuất clock bổ sung này được tạo ở tốc độ tần số được cấu hình trước bằng 256 × FS, trong đó FS là tần số lấy mẫu âm thanh.
  - Trong dự án này chúng ta sẽ sử dụng I2S3 và các bạn có thể sử dụng các khối I2S khác vẫn được. Lưu ý là I2S2 có kết nối trực tiếp trên board vì vậy sẽ có thể tín hiệu từ microphone trên board sẽ làm hư hỏng tín hiệu âm thanh chúng ta ghi, vì vậy không khuyến khích sử dụng I2S2.

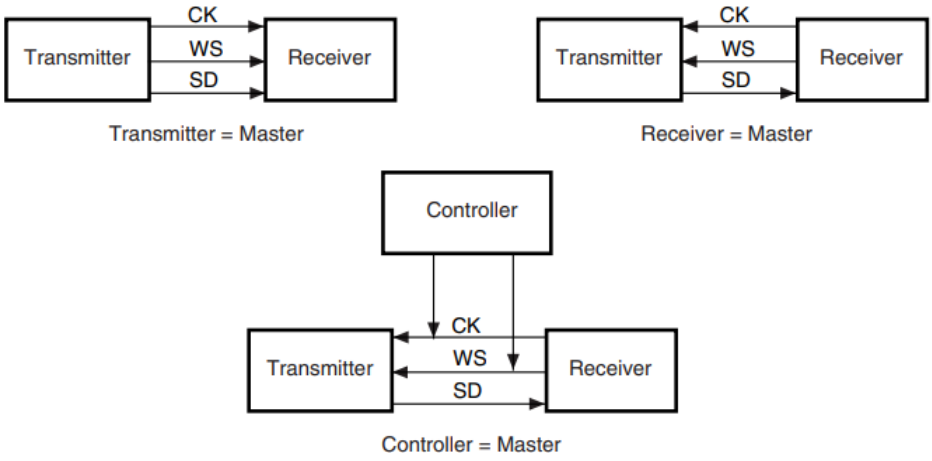


Figure 9: I2S protocol signal description and configuration.

b. Các chế độ hoạt động của I2S:

- Master Mode: Trong chế độ này, thiết bị điều khiển (master) tạo ra các tín hiệu clock (SCK) và word select (WS). Thiết bị này thường là bộ vi điều khiển hoặc DSP
- Slave Mode: Trong chế độ này, thiết bị phụ (slave) nhận các tín hiệu clock và word select từ thiết bị điều khiển. Thiết bị phụ có thể là bộ giải mã âm thanh hoặc bộ chuyển đổi tương tự-số (ADC).

- Transmit Mode: Thiết bị hoạt động ở chế độ này sẽ truyền dữ liệu âm thanh ra ngoài qua đường dây dữ liệu (SD).
- Receive Mode: Thiết bị hoạt động ở chế độ này sẽ nhận dữ liệu âm thanh vào từ đường dây dữ liệu (SD).
- Full Duplex Mode: Trong chế độ này, thiết bị có thể đồng thời truyền và nhận dữ liệu âm thanh. Điều này yêu cầu hai đường dây dữ liệu riêng biệt, một cho truyền và một cho nhận.

c. Data frame:

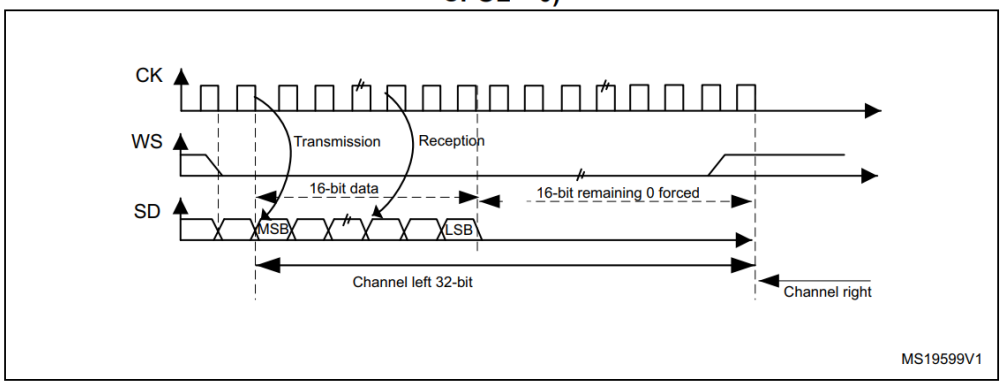
- Dữ liệu âm thanh chỉ được truyền trên một four-line bus và thường được ghép kênh theo thời gian trên hai kênh: Kênh phải và kênh trái.
- Tuy nhiên chỉ có 1 thành ghi 16-bit cho việc truyền và nhận dữ liệu, vì vậy phần mềm phải ghi vào thanh ghi dữ liệu giá trị thích hợp tương ứng với phía kênh được xem xét hoặc đọc dữ liệu từ thanh ghi dữ liệu và xác định kênh tương ứng bằng cách kiểm tra bit CHSIDE trong thanh ghi SPI_SR.
- Kênh bên trái sẽ luôn được gửi trước, sau đó kênh bên phải (CHSIDE) không có ý nghĩa đối với giao thức PCM).
- Có 3 khung dữ liệu và gói được sử dụng trong phương thức giao tiếp I2S. Và định dạng dữ liệu sẽ được gửi như sau:
- Dữ liệu 16-bit được đóng gói trong khung 16 bit.
- Dữ liệu 16-bit được đóng gói trong khung 32 bit.
- Dữ liệu 24-bit được đóng gói trong khung 32 bit.
- Dữ liệu 32-bit được đóng gói trong khung 32 bit.
- Khi sử dụng định dạng 16-bit được đóng gói trong khung 32-bit thì 16 bit đầu tiên (MSB) là các bit có nghĩa còn 16 LSB bit thì buộc về 0 mà không cần bất kì hành động nào của phần mềm hay là yêu cầu từ DMA nào (chỉ một thao tác động hoặc ghi).
- Khung dữ liệu 24-bit và 32-bit cần hai thao tác đọc và ghi từ CPU tới hoặc từ SPI_DR hoặc hai thao tác này từ DMA nếu DMA được ưu tiên cho cho ứng dụng.
- Đối với khung dữ liệu 24-bit cụ thể, 8-bit không quan trọng được mở rộng thành 32-bit với 0 bit (theo phần cứng). Đối với tất cả các định dạng dữ liệu và tiêu chuẩn truyền thông, bit quan trọng nhất luôn được gửi trước (MSB trước). Giao diện I2S hỗ trợ bốn tiêu chuẩn âm thanh, có thể cấu hình bằng cách sử dụng các bit I2SSTD [1:0] và PCMSYNC trong thanh ghi SPI_I2SCFGR.

d. I2S Philips standard

- Đối với tiêu chuẩn này, tín hiệu WS được sử dụng để cho biết kênh nào đang được truyền đi. Nó được kích hoạt một chu kỳ xung nhịp CK trước khi có bit đầu tiên (MSB)

- Data frame của I2S Philips standard (16-bit được đóng gói trong khung 32 bit) được thể hiện ở hình:

Figure 214. I²S Philips standard (16-bit extended to 32-bit packet frame with CPOL = 0)



When 16-bit data frame extended to 32-bit channel frame is selected during the I²S configuration phase, only one access to SPI_DR is required. The 16 remaining bits are forced by hardware to 0x0000 to extend the data to 32-bit format.

Figure 10: I^2 S Philips standard.

e. Clock generator

- Tốc độ bit I2S (I2S bitrate) xác định luồng dữ liệu trên dây dữ liệu I2S và tần số của I2S clock.

I2S bitrate = số bit của mỗi kênh (channel) × số lượng kênh (channel) × tần số lấy mẫu. ***(I2S bitrate = number of bits per channel × number of channels × sampling audio frequency)***

Ví dụ với âm thanh 16-bit, sử dụng hai kênh trái phải (left and right channels), tốc độ bit I2S là:

$$\text{I 2S bitrate} = 16 \times 2 \times \text{FS}$$

Lưu ý nếu độ rộng của gói là 32-bit thì I2S bitrate = 32 × 2 × FS.

f. DMA controller overview

- Direct memory access (DMA) được sử dụng để cung cấp truyền dữ liệu tốc độ cao giữa các thiết bị ngoại vi và bộ nhớ và giữa bộ nhớ và bộ nhớ. Dữ liệu có thể được di chuyển nhanh chóng bằng DMA mà không cần bất kỳ hành động nào của CPU. Điều này giúp giải phóng tài nguyên CPU cho các hoạt động khác.
- Trong dự án này, chúng tôi sử dụng chế độ Peripheral-to-memory, do đó mỗi khi có yêu cầu từ thiết bị ngoại vi, luồng sẽ khởi tạo một lần truyền từ nguồn để điền vào FIFO. Và chế độ Increment được bật ở nửa từ, do đó địa chỉ của lần truyền tiếp theo là địa chỉ của lần truyền trước đó được tăng thêm 2 trong thanh ghi DMA_SxCR.

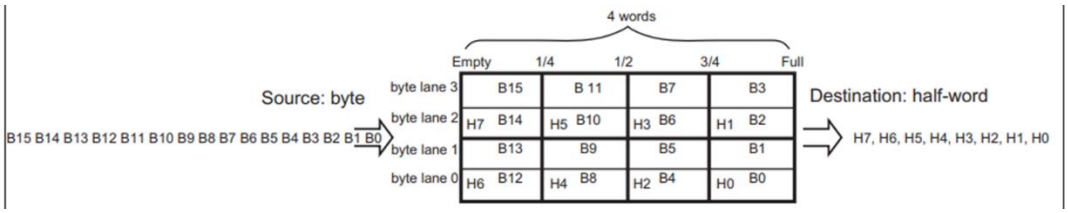


Figure 11: data location in FIFO with half- word mode.

- Trong dự án này sử dụng circular mode để xử lý circular buffer và luồng dữ liệu liên tục, do đó khi buffer được ghi hết, dữ liệu cần truyền sẽ được tự động ghi đè lên các giá trị ban đầu được lập trình trong giai đoạn cấu hình luồng và các yêu cầu DMA tiếp tục được phục vụ.

4. DMA với I2S

- DMA hoạt động theo cách tương tự như đối với chế độ SPI. Không có sự khác biệt trên I2S. Chỉ có tính năng CRC là không khả dụng ở chế độ I2S do không có hệ thống bảo vệ truyền dữ liệu.
- I2S sử dụng cùng một thanh ghi SPI để truyền dữ liệu (SPI_DR) ở chế độ rộng 16 bit.
- Để hoạt động ở tốc độ tối đa, I2S cần được cung cấp dữ liệu để truyền và dữ liệu nhận được trên bộ đệm Rx phải được đọc để tránh tràn. Để tạo điều kiện thuận lợi cho việc truyền, I2S có khả năng DMA triển khai giao thức yêu cầu/xác nhận đơn giản. Các yêu cầu riêng biệt phải được gửi đến bộ đệm Tx và Rx.

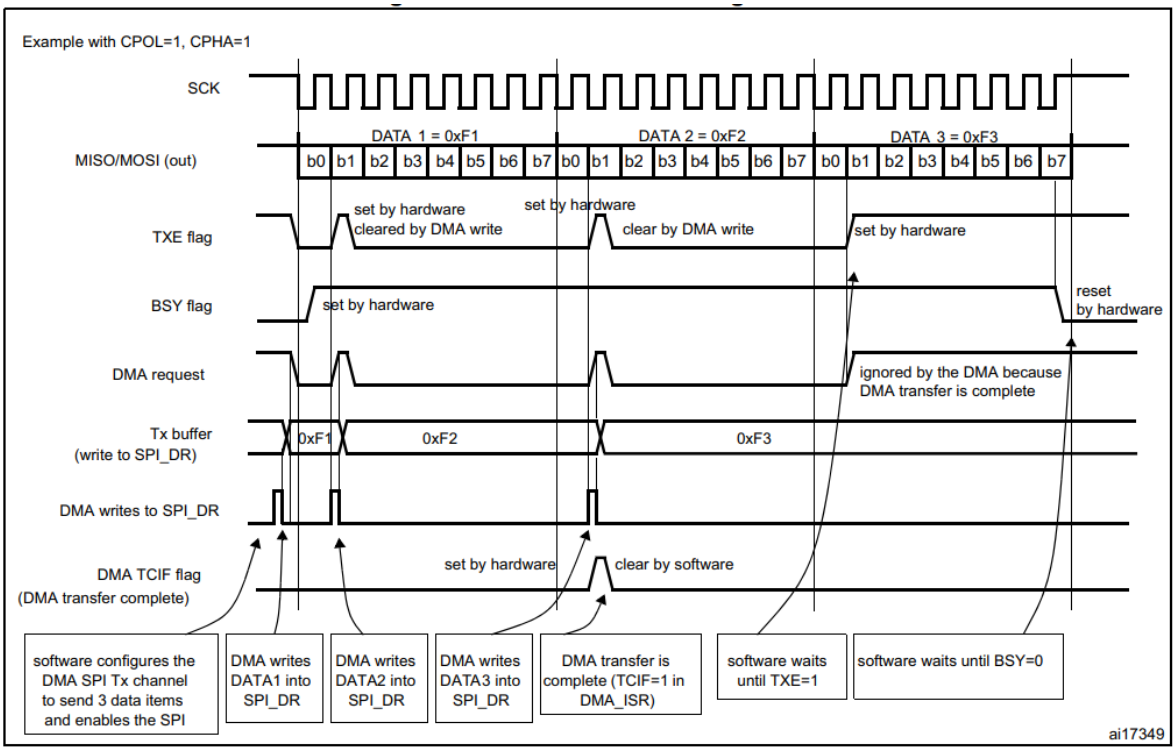


Figure 12: Transmission using DMA.

- Trong quá trình truyền, một yêu cầu DMA được gửi mỗi khi TXE được đặt thành 1. Sau đó, DMA ghi vào thanh ghi SPI_DR (điều này xóa TXE flag).(Show in Figure)

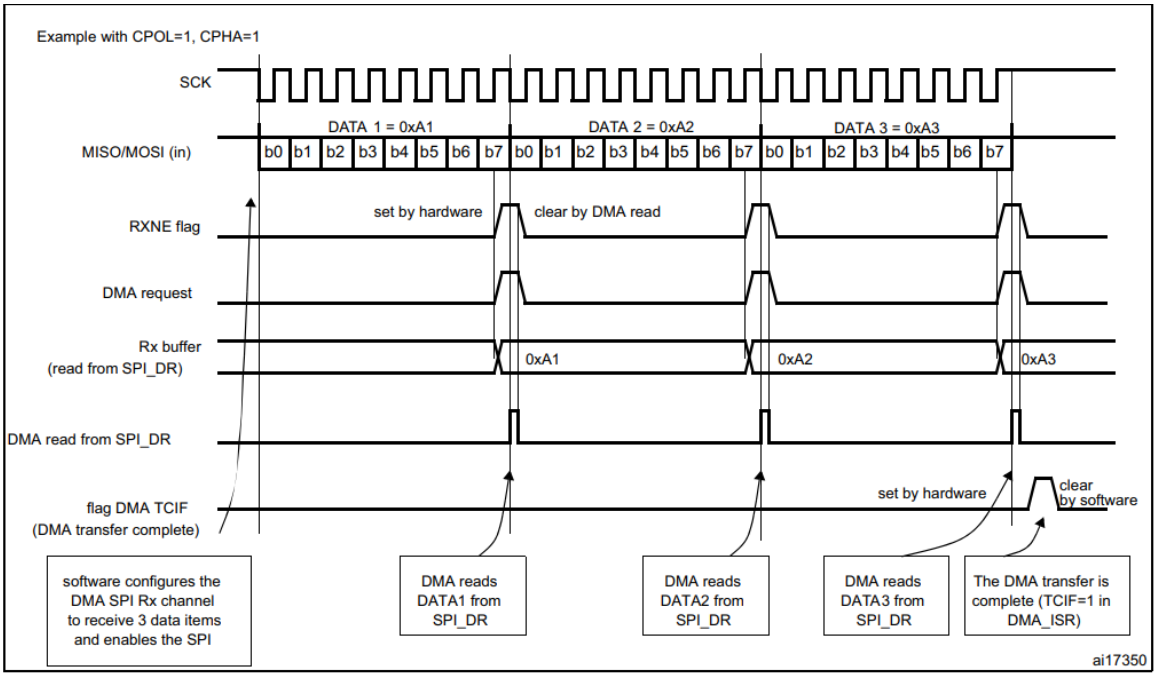


Figure 13: Reception using DMA.

- Trong quá trình tiếp nhận, một yêu cầu DMA được gửi mỗi khi RXNE được đặt thành 1. Sau đó, DMA đọc thanh ghi SPI_DR (điều này xóa cờ RXNE flag). (Show in figure)
- Khi I2S chỉ được sử dụng để truyền dữ liệu, có thể chỉ bật kênh DMA Tx SPI. Trong trường hợp này, the OVR flag được đặt vì dữ liệu nhận được không được đọc.
- Khi I2S chỉ được sử dụng để nhận dữ liệu, có thể chỉ bật kênh SPI Rx DMA. Ở chế độ truyền, khi DMA đã ghi tất cả dữ liệu cần truyền (TCIF flag được đặt trong thanh ghi DMA_ISR), có thể theo dõi cờ BSY để đảm bảo rằng giao tiếp I2S đã hoàn tất.
- Với dự án thu âm với giao thức I2S ta dùng Bộ điều khiển DMA để xử lý việc truyền dữ liệu âm thanh từ thiết bị ngoại vi I2S sang bộ nhớ đệm. Sau đó, CPU có thể xử lý dữ liệu theo từng khối lớn hơn thay vì xử lý từng mẫu riêng lẻ. Giúp đảm bảo luồng dữ liệu âm thanh dựa trên I2S mà không cần sử dụng quá nhiều CPU, cải thiện hiệu suất và hiệu quả.

5. Thực hành đọc âm thanh xem kết quả bộ đệm debug xem

a. Chuẩn bị phần cứng và sơ đồ nối dây.

- **Digital microphone:**

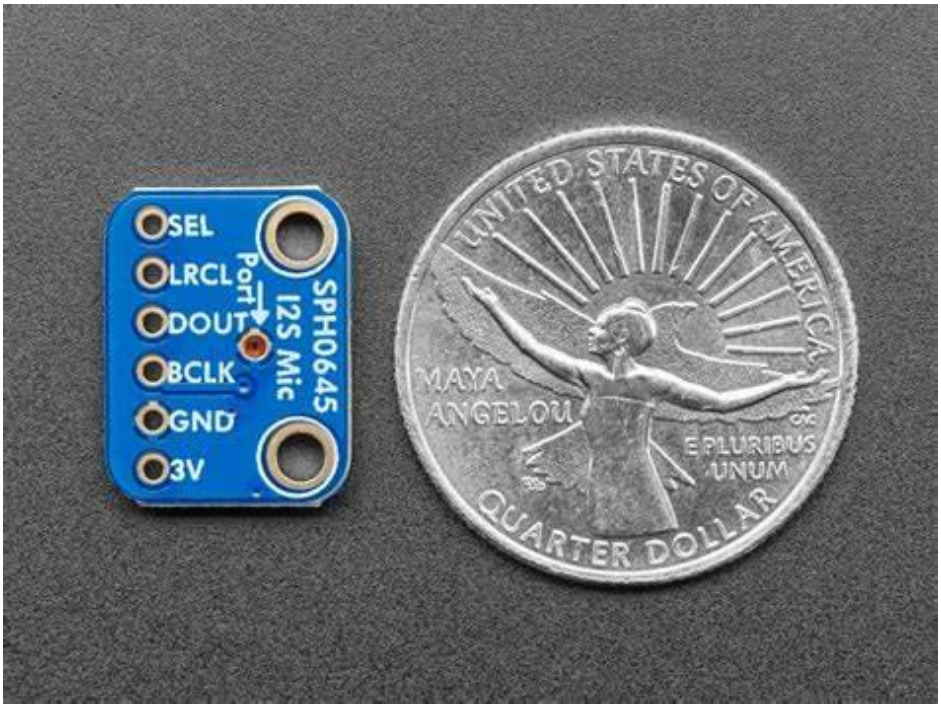


Figure 14: SPH0445 MEMS microphone.

Table 1: Một số thông số kĩ thuật của SPH0645

Name	Microphone SPH0645
Type	MEMS Microphones
Frequency Range	1.024 MHz to 4096 MHz
Operating Supply Current	600 uA
Operating Supply Voltage	1.8 V
Output Type	Digital, I2S
Voltage Supply	1.6 V - 3.6 V

- 3V - đây là nguồn điện. Về mặt kỹ thuật, nó có thể được cấp nguồn từ mức thấp tới 1,6V đến 3,6V nhưng bạn cần đảm bảo mức logic của mình khớp.
- GND - nguồn và đất dữ liệu.
- BCLK - xung nhịp bit, còn được gọi là xung nhịp dữ liệu - đến từ I2S chính để báo cho micrô biết đã đến lúc truyền dữ liệu.
- DOUT - dữ liệu đầu ra từ micrô.
- LRCLK - xung nhịp trái/phải, còn được gọi là WS (chọn từ), điều này báo cho micrô biết khi nào bắt đầu truyền. Khi LRCLK ở mức thấp, kênh trái sẽ truyền. Khi LRCLK ở mức cao, kênh phải sẽ truyền.

- SEL - chân chọn kênh. Theo mặc định, chân này ở mức thấp, do đó nó sẽ truyền trên kênh đơn âm bên trái. Nếu bạn kết nối chân này với điện áp logic cao, micrô sẽ ngay lập tức bắt đầu truyền trên kênh phải.

Trong dự án này chúng tôi dùng digital microphone SPH0645 và I2S3 ở chế độ Half Duplex Mater để chỉ nhận dữ liệu từ microphone. Và sơ đồ chân và kết nối chân được thể hiện như hình vẽ:

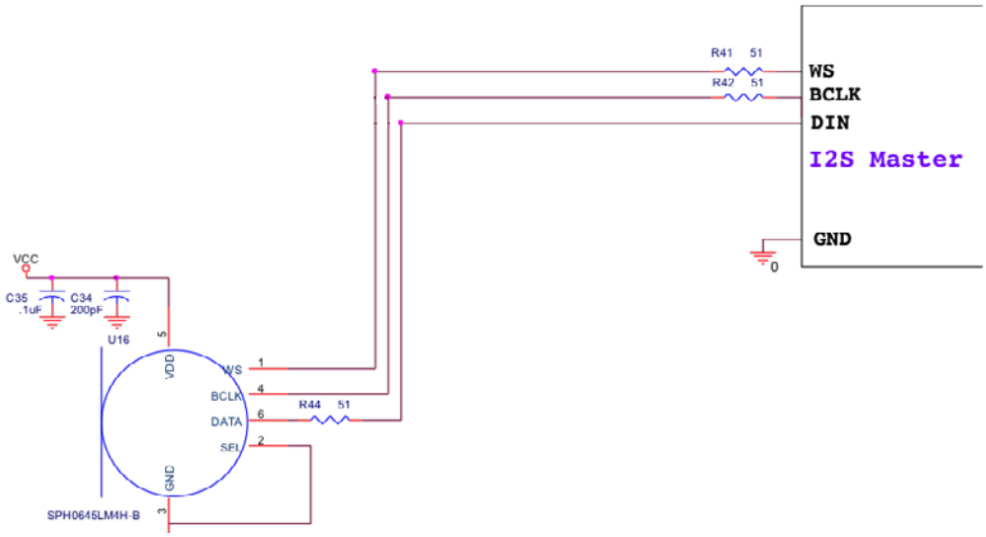


Figure 15: Sơ đồ kết nối chân giữa digital microphone với các chân của khối I2S.

Chân LRCLK, BCLK, DOUT của micrphone nối lần lượt với chân WS, Clock, and SD của I2S3. Ngoài ra chân SEL được kết nối với GND. (Được thể hiện ở Figure 13).

- **Sơ đồ kết nối giữa các module và vi điều khiển.**
  - Chương trình sẽ mặc định các chân I2S3 của stm32F411 ứng với module microphone như sau:

PA4	I2S3_WS
PC10	I2S3_CK
PC12	I2S3_SD

  - Ta sẽ kết nối các chân của khối I2S3 tương ứng với các chân của module giao tiếp thẻ nhớ và microphone SPH0645.

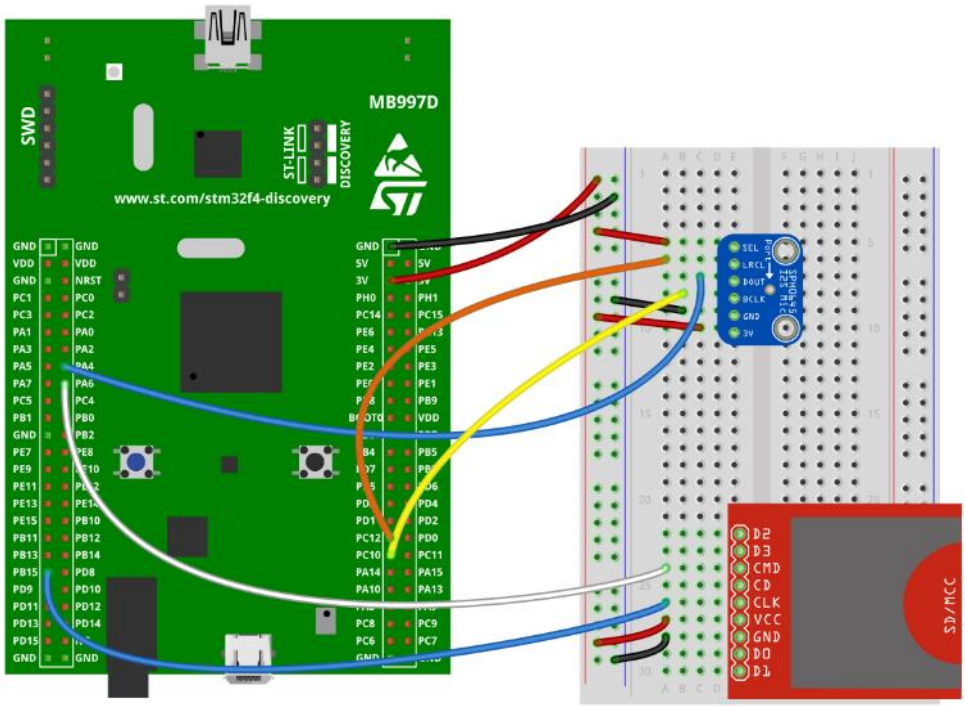


Figure 16: Sơ đồ nối dây giữa vi điều khiển STM32F411 và moude giao tiếp thẻ nhớ, microphone SPH0645.

b. Cấu hình và lập trình cho vi điều khiển trên phần mềm STM32CubeIDE

Bước 1: Khởi động phần mềm STM32CubeIDE.

Kích chọn dòng STM32F411VETx và chọn “Next”.

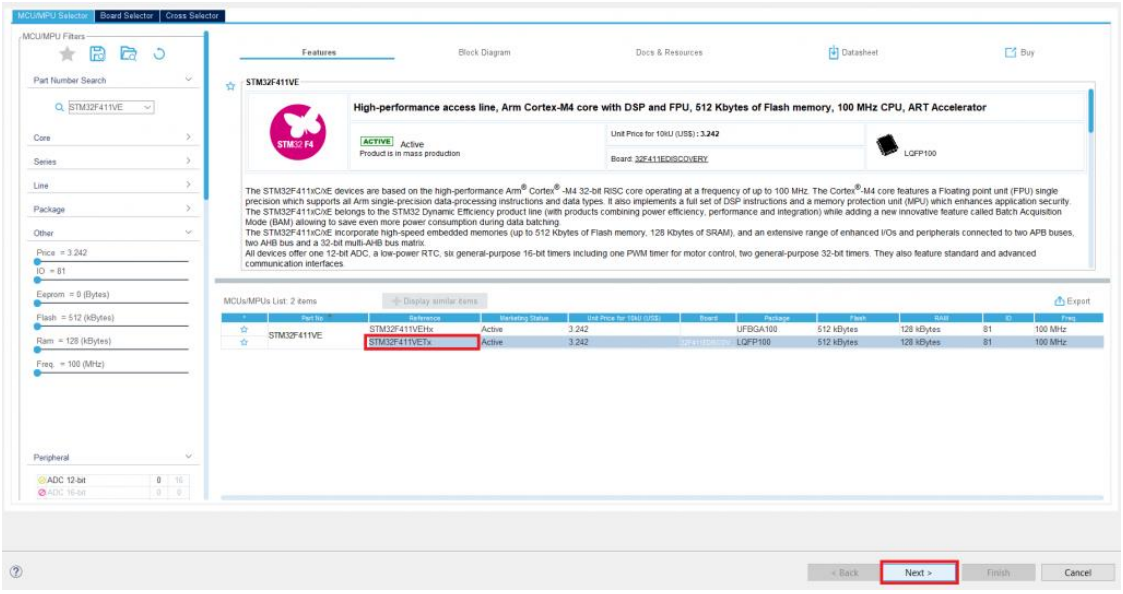


Figure 17: Giao diện ở màn hình chính của phần mền STM32 CubeIDE.

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

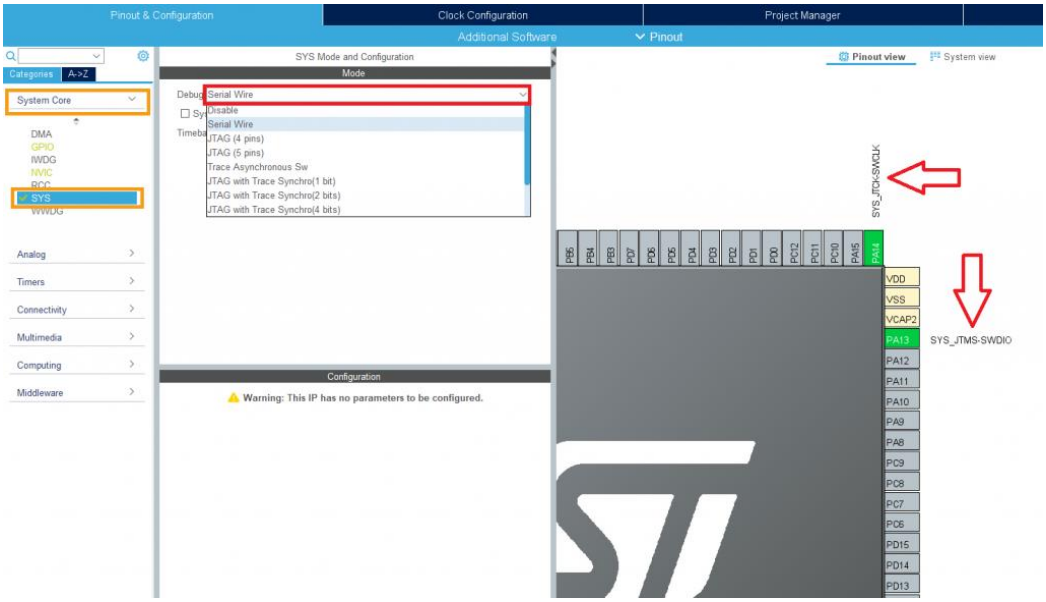


Figure 18: Cấu để nạp chương trình vào vi điều khiển.

Bước 3: Cấu hình cho khối I2S và DMA

- Ở mục Multimedia, chọn I2S3 -> Mode -> Haf-Duplex Mater

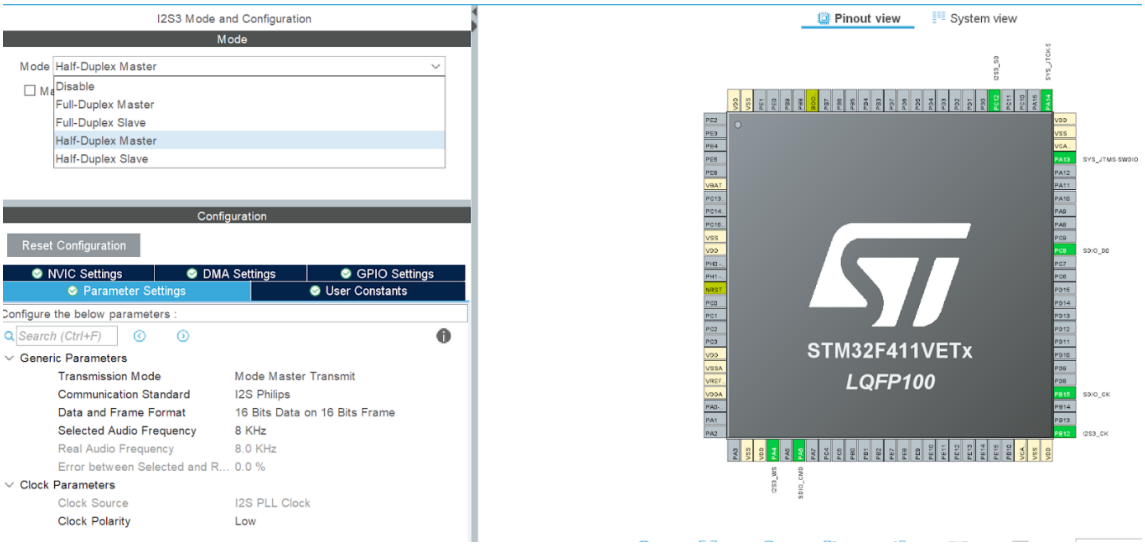


Figure 19: Chọn chế độ làm việc Half-Duplex Mater cho I2S3.

- Ở mục Parameter Settings, ta chọn Mode Master Receive và chuẩn giao tiếp I2S Philips với khung dữ liệu 16 Bit trên khung truyền 32 Bit. Và chúng ta cũng thiết lập tần số lấy mẫu là ở 32 kHz.

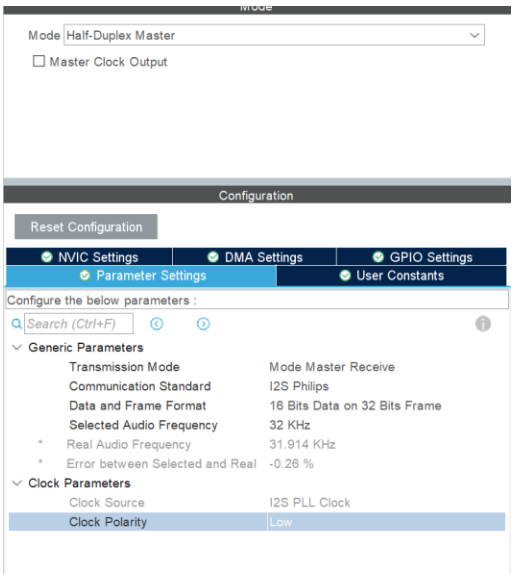


Figure 20: Thông số cấu hình cho khối I2S3.

- Vào mục DMA Settings -> Add -> SPI3_RX và ở mục Mode -> Circular, ở mục Data Width -> Half Word.

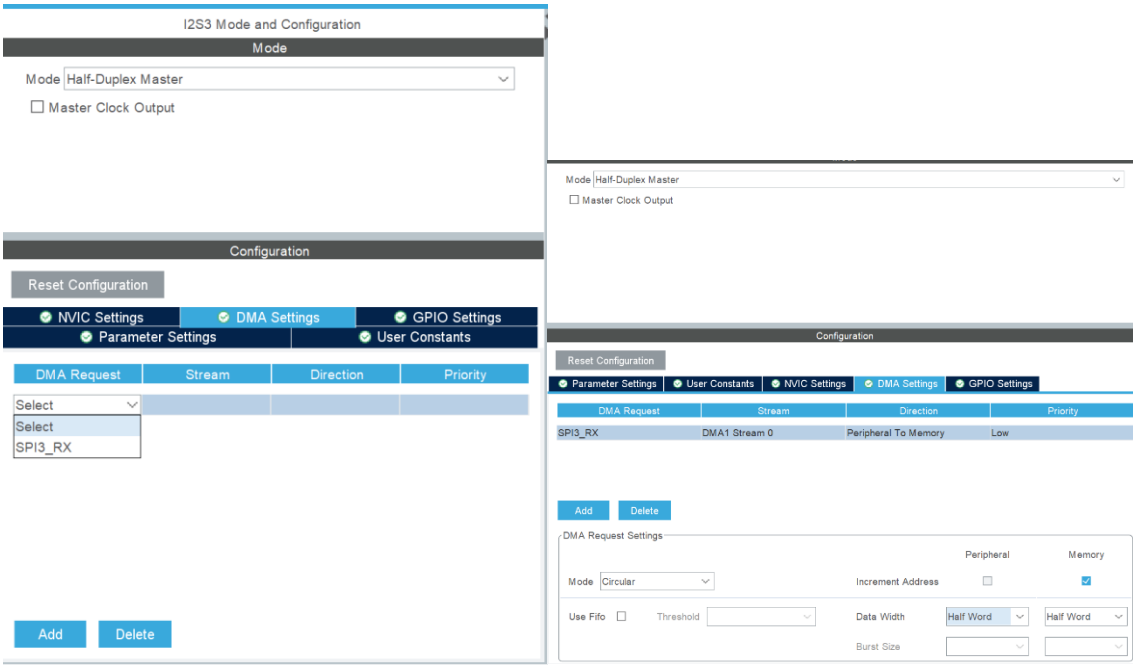


Figure 21: Thêm và cấu hình cho DMA điều khiển khối I2S3.

Bước 4: Thiết lập thông số ở mục Clock Configuration

SMT32F411 có tần số clock tối ta là 100 MHz thì các bạn có thể cài đặt, nhưng nếu trong quá trình trình ghi dữ liệu vào thẻ nhớ gặp tình trạng xuất hiện các dòng chữ

tiếng Trung thì có thể hạ giá trị tần số của Clock xuống. Và ở đây dùng clock với tần số 80 MHz.

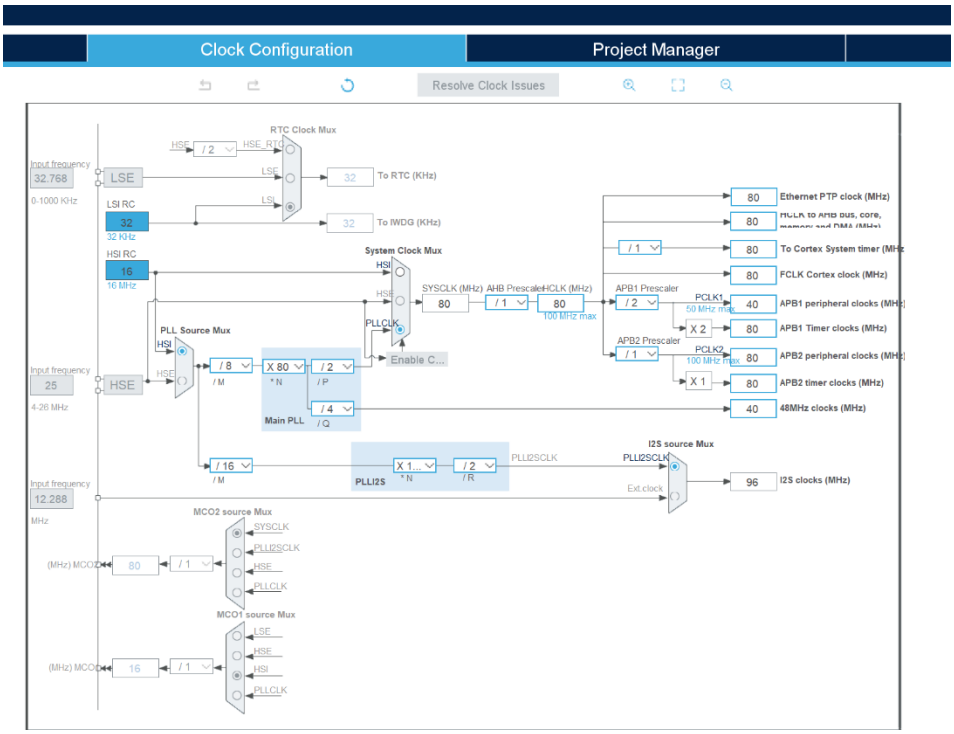


Figure 22: Thông số cấu hình cho clock của vi điều khiển.

Bước 5: Sinh code bằng cách nhấn tổ hợp phím Alt+K

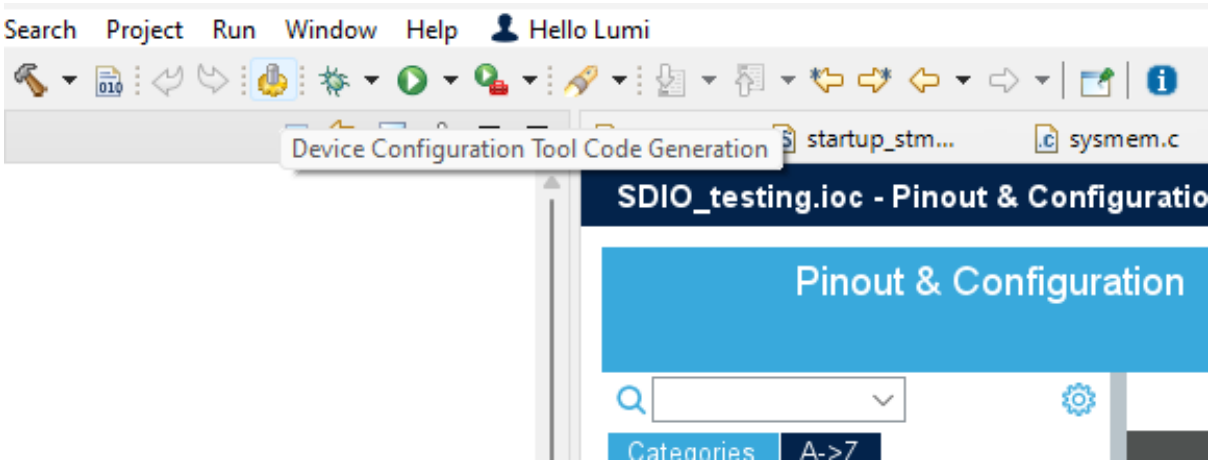


Figure 23: Thao tác trên thanh công cụ của phần mền để sinh code.

Bước 6: Setup màn hình SWV ITM Data console.

- Nhấn Debug -> Debug configurations.

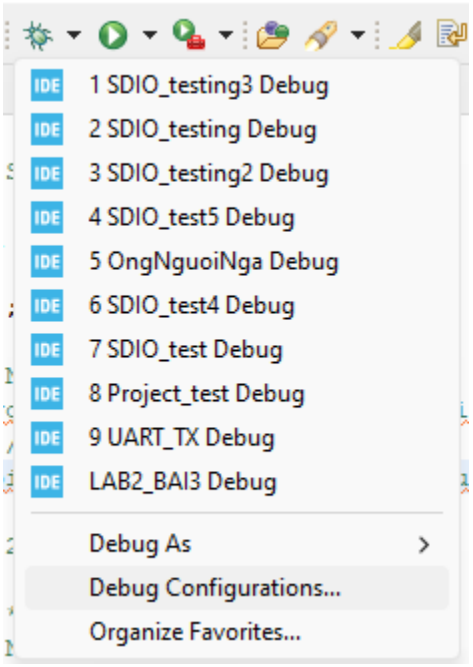


Figure 24: Thao tác chọn Debug Configurations ở trên thanh công cụ.

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

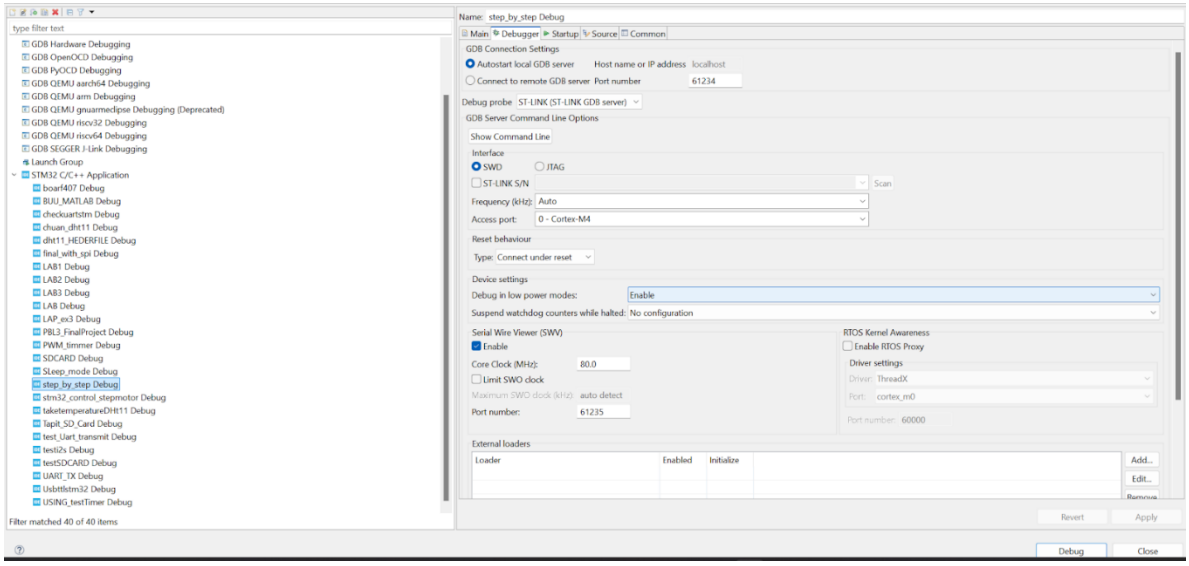


Figure 25: Enable Serial Wire Viewer(SWW) để có thể mở các của sổ của mục SWW.

Bước 7: Code

/* USER CODE BEGIN Includes */

#include "stdio.h"

```
/* USER CODE END Includes */


/* USER CODE BEGIN PV */
#define WAV_WRITE_SAMPLE_COUNT 50 // buffer
int16_t data_i2s[WAV_WRITE_SAMPLE_COUNT];/*array to store received audio data */
volatile int16_t sample_i2s;
/* USER CODE END PV */
int main(void)
{
 /* USER CODE BEGIN 2 */
 HAL_I2S_Receive_DMA(&hi2s3,(uint16_t*)data_i2s, sizeof(data_i2s));
 /* USER CODE END 2 */
 while (1)
  {
  }
}


/* 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;
}
void HAL_I2S_RxCpltCallback(I2S_HandleTypeDef *hi2s)
```

```
{
        sample_i2s = data_i2s[0];
}
/* USER CODE END 4 */
```

Bước 8: Cấu hình đồ thị lấy mẫu âm thanh

- Window -> Show view -> SWV -> SWV Data Trace Timeline Graph

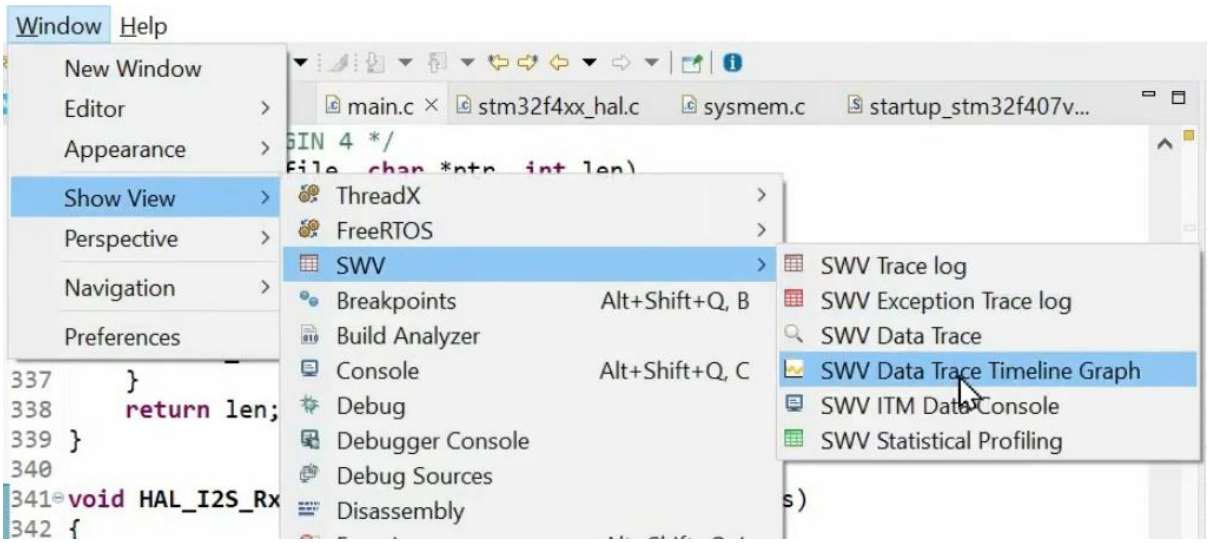


Figure 26: Cách mở của số SWV Data Trace Timeline Grahp để theo dõi tín hiệu thu được ở dạng đồ thị.

- Trong cửa sổ SWV settings, trong mục ITM Stimulus Ports, enable port 0. Trong mục Data Trace, enable Comparator 0, nhập tên biến sample_i2s -> OK.

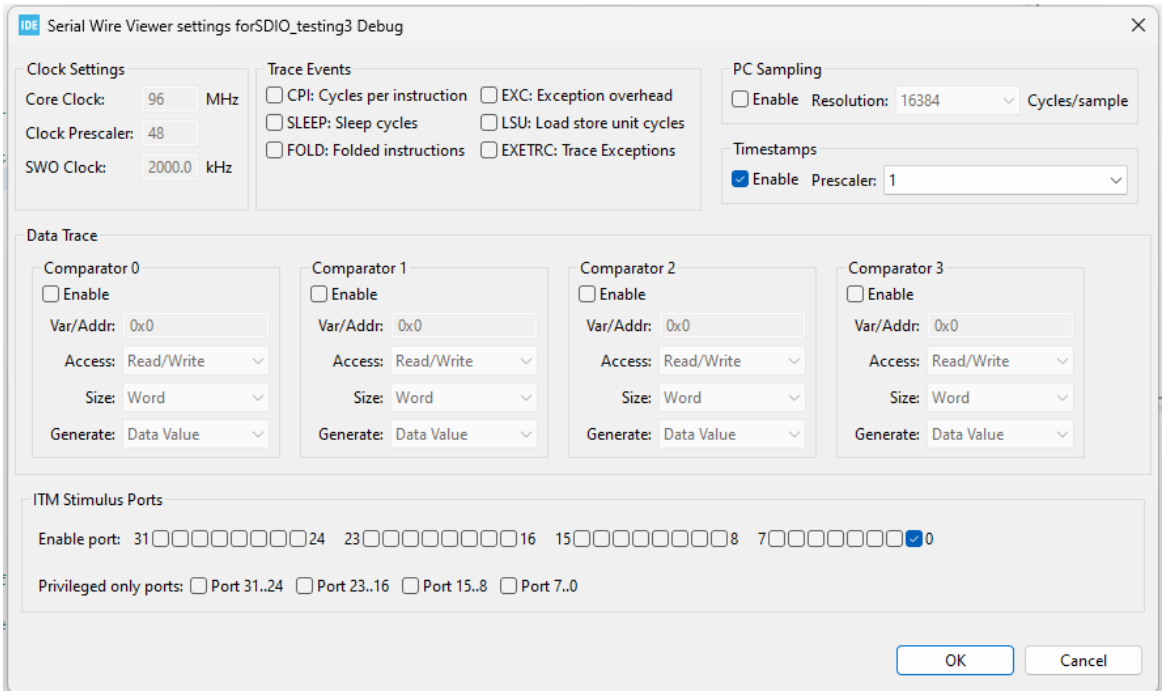


Figure 27: Enable port 0

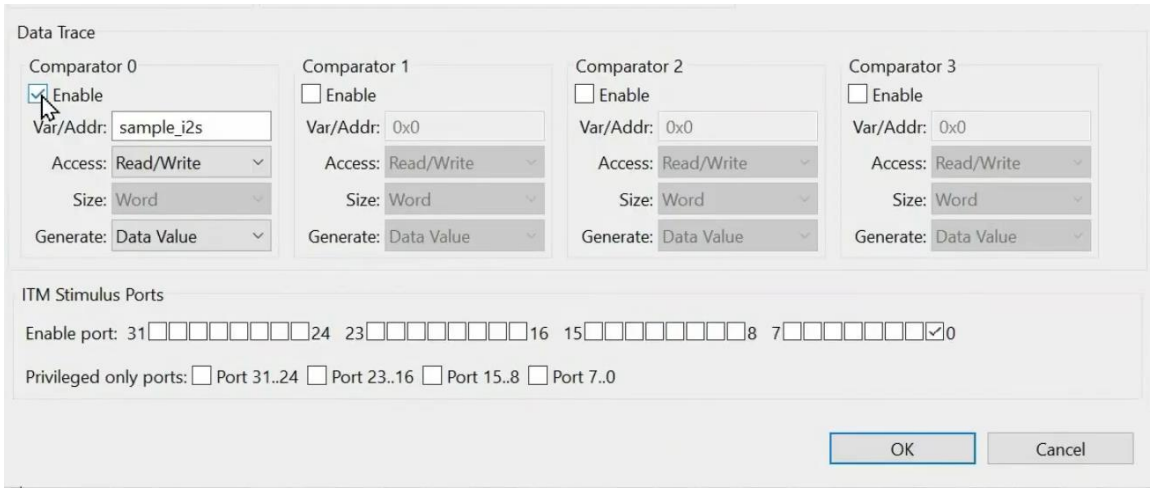


Figure 28: Enable và nhập tên biến cần theo dõi trên của số SWV Data Trace Timeline Grahp.

- Nhấn nút Start Trace

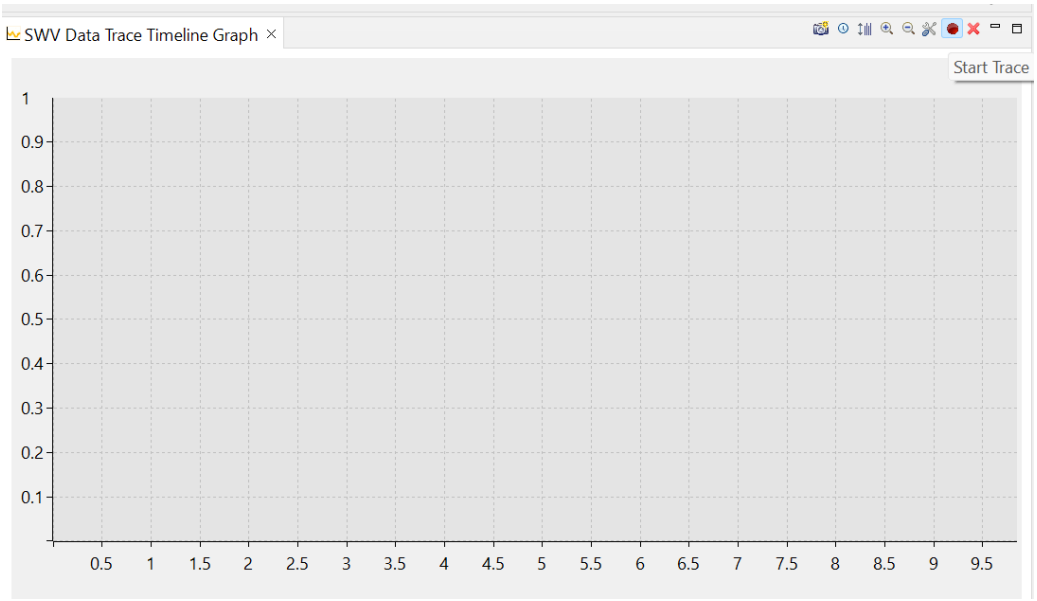


Figure 29: Vị trí của nút Start Trace trên của số SWV Data Trace Timeline Grahp.

Bước 9: 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 Data Trace console.

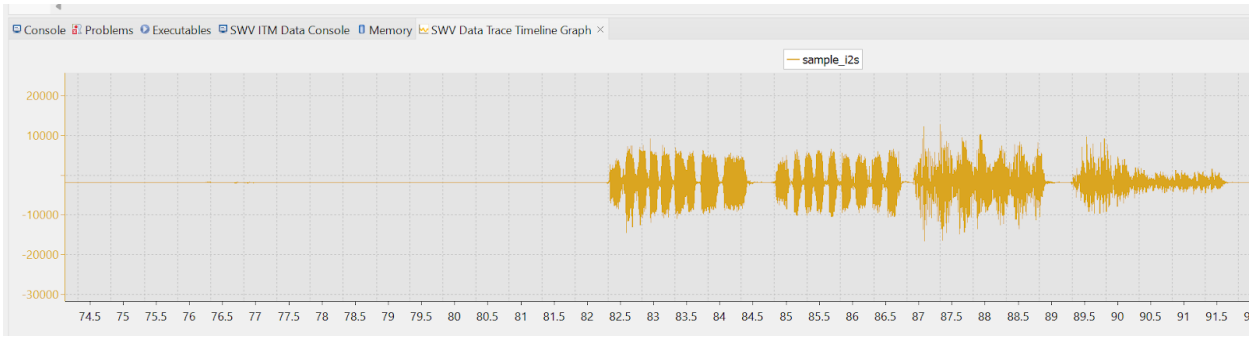
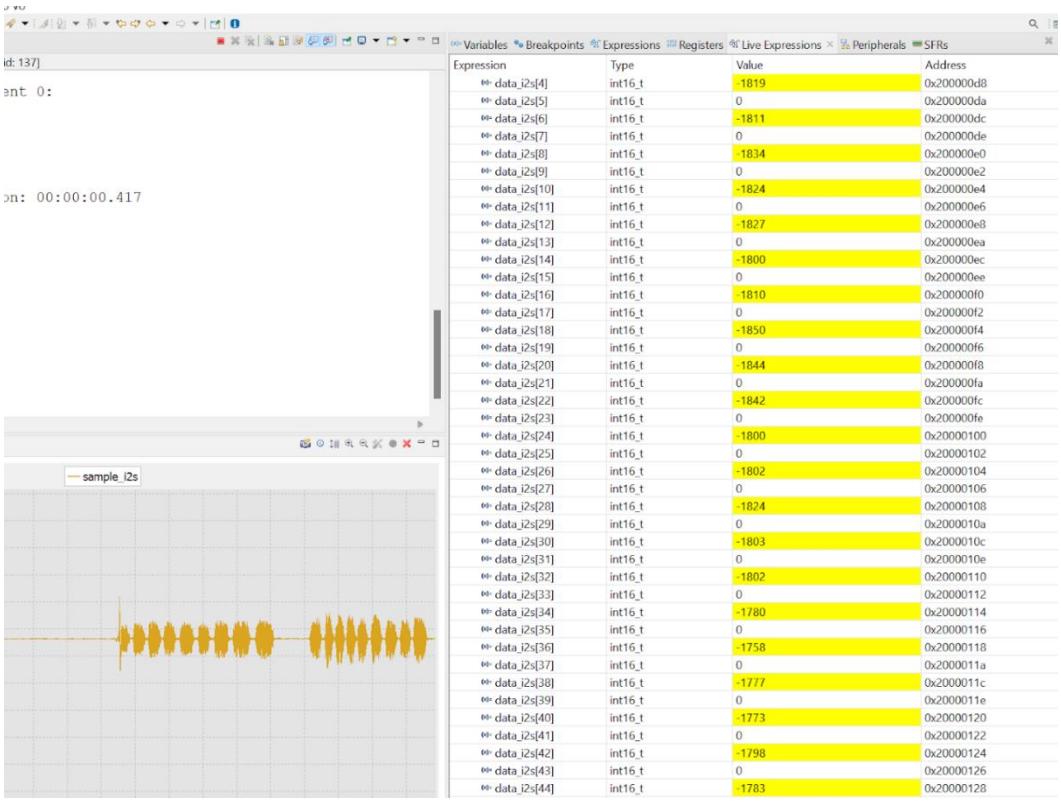


Figure 30: Kết quả dạng sóng thu được từ microphone.

- Ở nhấp vào Live Expression, gõ tên buffer data_i2s và theo dõi các giá trị được ghi vào buffer.



Expression	Type	Value	Address
data_i2s[4]	int16_t	-1819	0x200000d8
data_i2s[5]	int16_t	0	0x200000da
data_i2s[6]	int16_t	-1811	0x200000dc
data_i2s[7]	int16_t	0	0x200000de
data_i2s[8]	int16_t	-1834	0x200000e0
data_i2s[9]	int16_t	0	0x200000e2
data_i2s[10]	int16_t	-1824	0x200000e4
data_i2s[11]	int16_t	0	0x200000e6
data_i2s[12]	int16_t	-1827	0x200000e8
data_i2s[13]	int16_t	0	0x200000ea
data_i2s[14]	int16_t	-1800	0x200000ec
data_i2s[15]	int16_t	0	0x200000ee
data_i2s[16]	int16_t	-1810	0x200000f0
data_i2s[17]	int16_t	0	0x200000f2
data_i2s[18]	int16_t	-1850	0x200000f4
data_i2s[19]	int16_t	0	0x200000f6
data_i2s[20]	int16_t	-1844	0x200000f8
data_i2s[21]	int16_t	0	0x200000fa
data_i2s[22]	int16_t	-1842	0x200000fc
data_i2s[23]	int16_t	0	0x200000fe
data_i2s[24]	int16_t	-1800	0x20000100
data_i2s[25]	int16_t	0	0x20000102
data_i2s[26]	int16_t	-1802	0x20000104
data_i2s[27]	int16_t	0	0x20000106
data_i2s[28]	int16_t	-1824	0x20000108
data_i2s[29]	int16_t	0	0x2000010a
data_i2s[30]	int16_t	-1803	0x2000010c
data_i2s[31]	int16_t	0	0x2000010e
data_i2s[32]	int16_t	-1802	0x20000110
data_i2s[33]	int16_t	0	0x20000112
data_i2s[34]	int16_t	-1780	0x20000114
data_i2s[35]	int16_t	0	0x20000116
data_i2s[36]	int16_t	-1758	0x20000118
data_i2s[37]	int16_t	0	0x2000011a
data_i2s[38]	int16_t	-1777	0x2000011c
data_i2s[39]	int16_t	0	0x2000011e
data_i2s[40]	int16_t	-1773	0x20000120
data_i2s[41]	int16_t	0	0x20000122
data_i2s[42]	int16_t	-1798	0x20000124
data_i2s[43]	int16_t	0	0x20000126
data_i2s[44]	int16_t	-1783	0x20000128

Figure 31: Dữ liệu thu được từ microphone được lưu trên buffer được quan sát trên của sổ Live Expressions

6. Thực hành đọc âm thanh lưu thẻ nhớ SD.

a. Chuẩn bị phần cứng và sơ đồ nối dây

- Ngoài microphone đã được kết nối như ở phần **6** thì để lưu dữ liệu thu được vào thẻ nhớ SD ta cần sử dụng thêm 1 module giao tiếp với thẻ nhớ theo chuẩn SDIO.

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

Figure 32: 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 33: Đầu đọc thẻ nhớ.

- Ngoài ra các bạn nên format lại thẻ microSD để thẻ ở đúng định dạng FAT32.

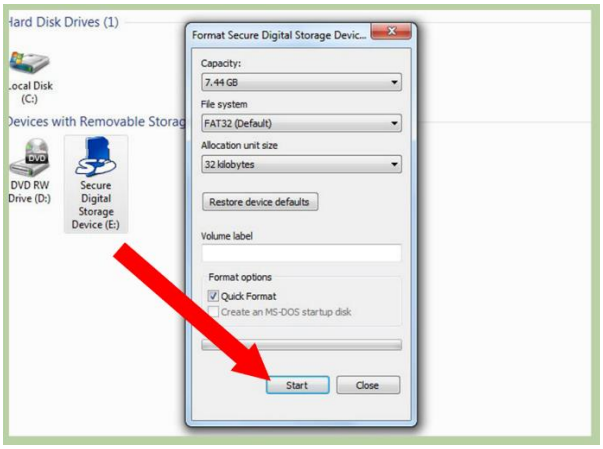


Figure 34: Reset format của thẻ nhớ SD

- Dưới đây chúng ta sẽ xét tới chế độ 4-bit nhưng trong dự án này ta chỉ dùng SDIO ở chế độ 1 bit, không có gì thay đổi ở kết nối phần cứng. Đầu tiên các bạn nối các chân của SDIO vào thẻ nhớ như sau:

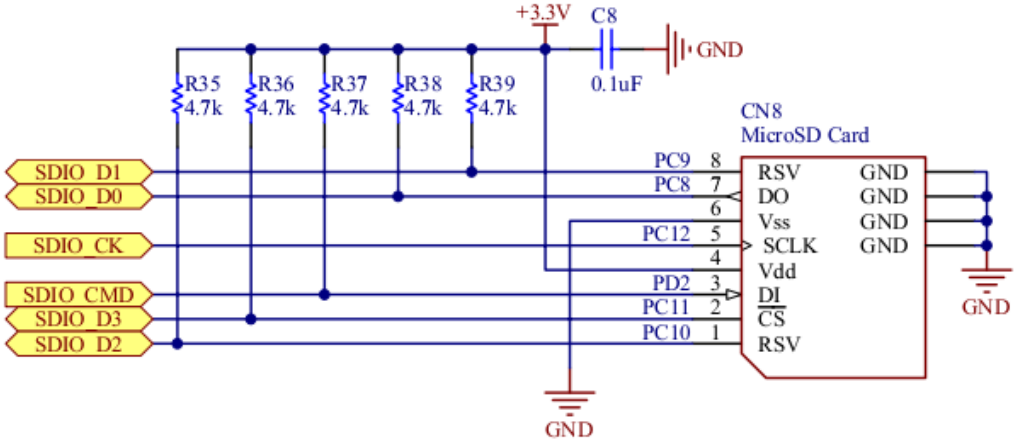


Figure 35: Sơ đồ kết nối chân giữa moudle giao tiếp với thẻ nhớ và khối SDIO

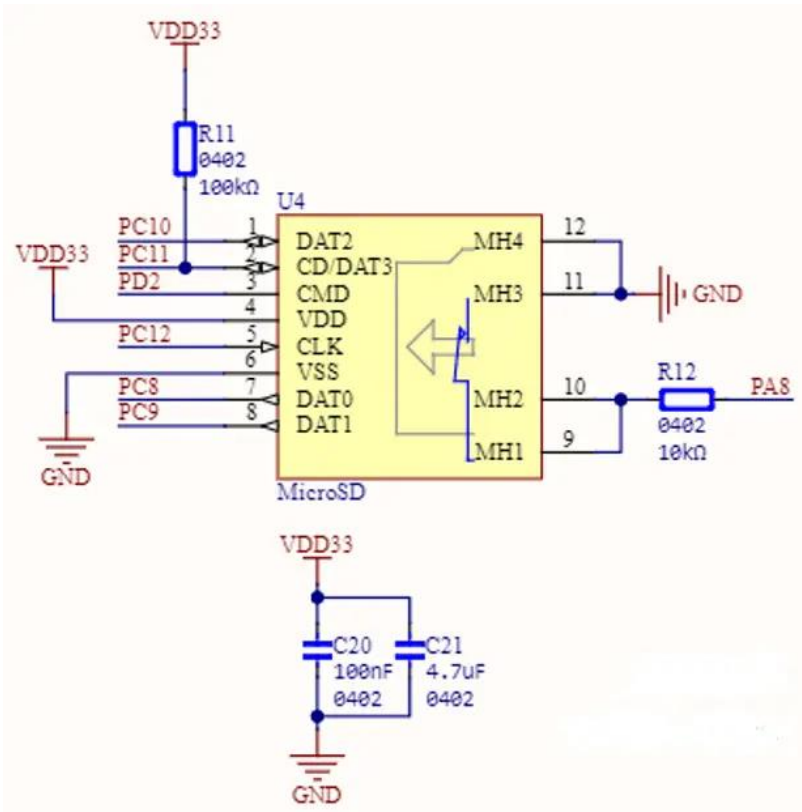


Figure 36: Sơ đồ chân của khối SDIO kết nối với ngoại vi trên STM32F411.

- **Ý nghĩa của các chân:**
  1. 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

2. Chân CMD (Command) chọn gửi lệnh hay Data
3. Chân GND và 3.3(V) là chân cấp nguồn
4. CLK là chân cấp xung clock
5. Sơ đồ kết nối giữa các module và vi điều khiển.

- 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

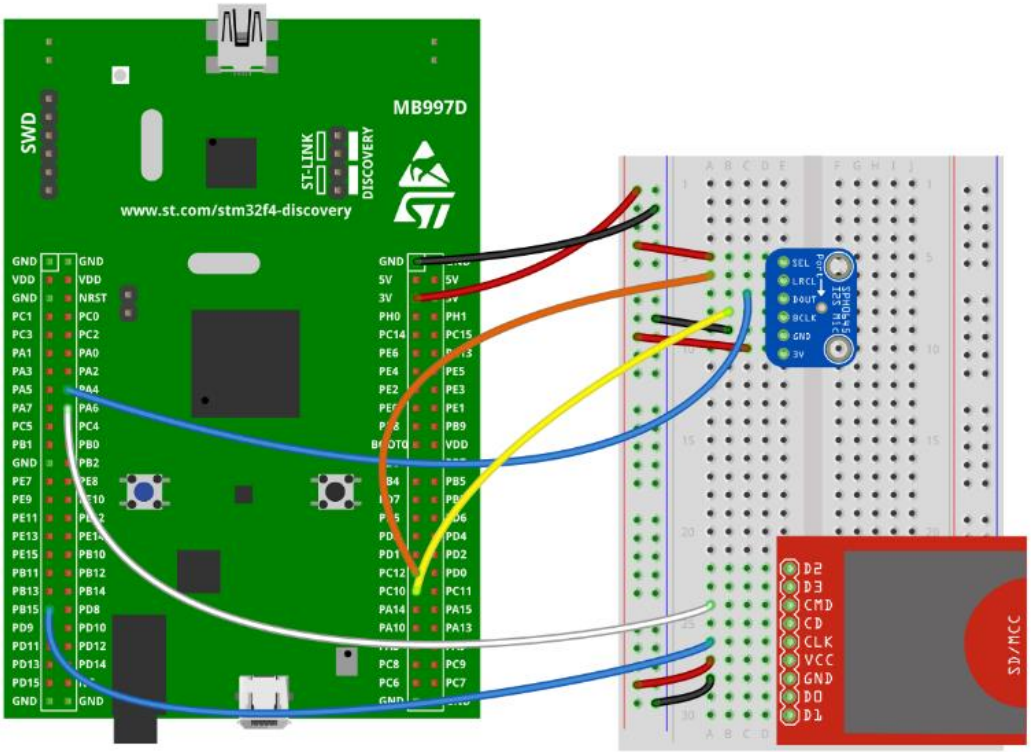


Figure 37: Sơ đồ nối dây giữa vi điều khiển STM32F411 và moude giao tiếp thẻ nhớ, microphone SPH0645.

b. Cấu hình và lập trình cho vi điều khiển trên phần mềm STM32CubeIDE

Ta vẫn sẽ giữ nguyên cấu hình cho khối I2S3 và các cấu hình khác như ở mục 6. Và Chỉ cần cấu hình cho khối SDIO và thư viện FASTFS và nút ấn để điều khiển.

Bước 1: Cấu hình cho khối SDIO

Ở mục Connectivity chọn SDIO -> Mode -> SD 1 bit and set up mục Parameter Settings.

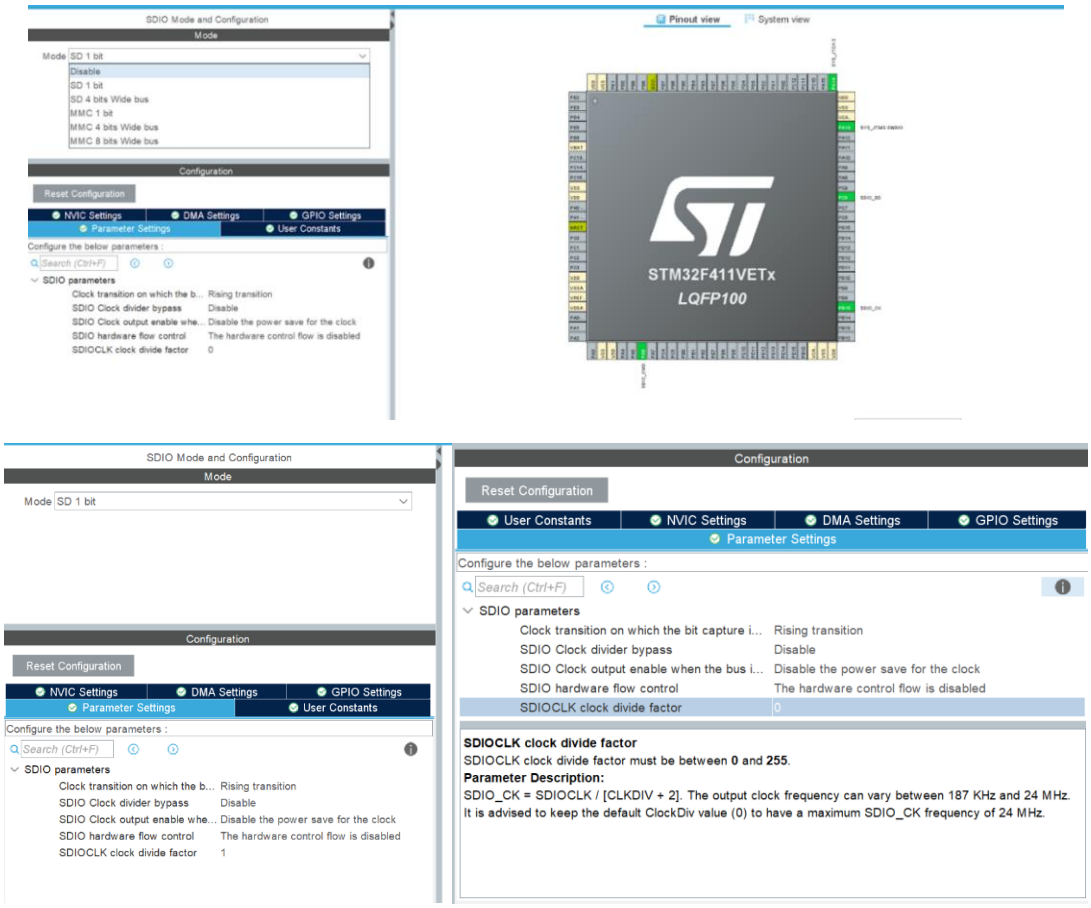


Figure 38: Cấu hình các thông số cho khối SDIO trên Cube IDE.

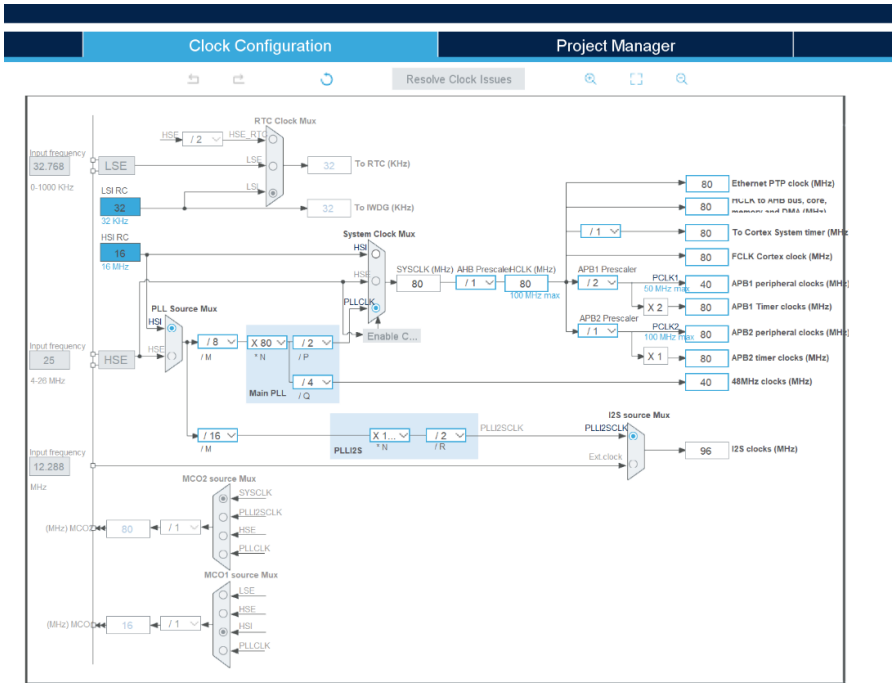


Figure 39: Cấu hình clock làm việc cho vi xử lí và clock cho các ngoại vi.

- Lưu ý ở mục SDIOCLK clock divider factor, chúng ta có thể chọn số chia cho SDIOCLK clock divide factor để lấy tần số đọc và ghi thẻ nhớ. 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 có thể dẫn đến tình trạng file ghi hiện các dòng chữ TRung Quốc do tần số ghi vào thẻ nhớ cao. Hoặc là sẽ bị lỗi kết nối với thẻ SD, khi đó ta có thể tăng giá trị của số chia clock này lên để tránh lỗi này. Ở project này, ta dùng SDIOCLK là 40, để là 1 có nghĩa tần số của SDIO_CK là: 40 / (2+1) = 13.3 MHz.

Bước 2: Thiết lập thông số để dùng thư viện FATFS.

- Ở mục Middleware and Software Packs -> FATFS->FATFS Mode and Configuration
- Ở mục Mode -> Tick into SD Card.

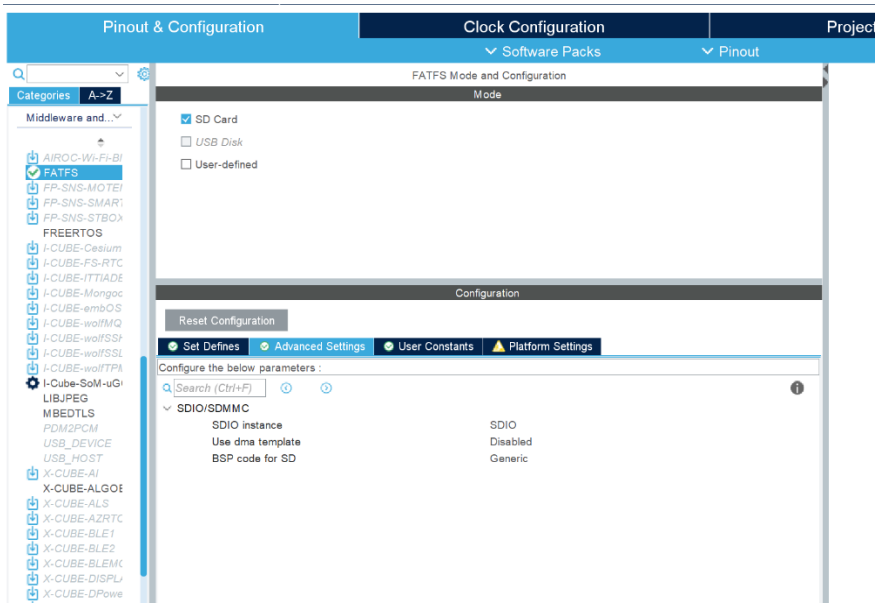


Figure 40: Chọn giao thức SD cho thư viện FASTFS.

- 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 41: Thông báo khi không có cấu chân Detect SDIO.

- Chọn PD0 at mode GPIO input (hoặc chân nào bật 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)

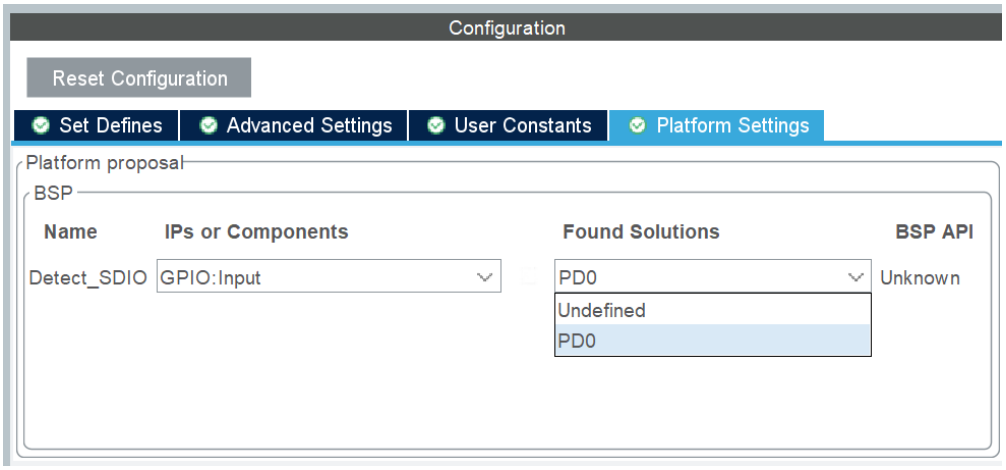


Figure 42: Chọn chân làm chân Detect SDIO.

Bước 3: Thiết lập nút ấn để điều khiển bắt đầu và dừng thu âm.

- Đặt chân A0 ở mode PIO_EXTI0 và gán nhãn B1[Blue Push Button].
- Ở mục System core, chọn NVIC và tick Enable of EXIT 0-line interrupt và đặt mức ưu tiên là 1.

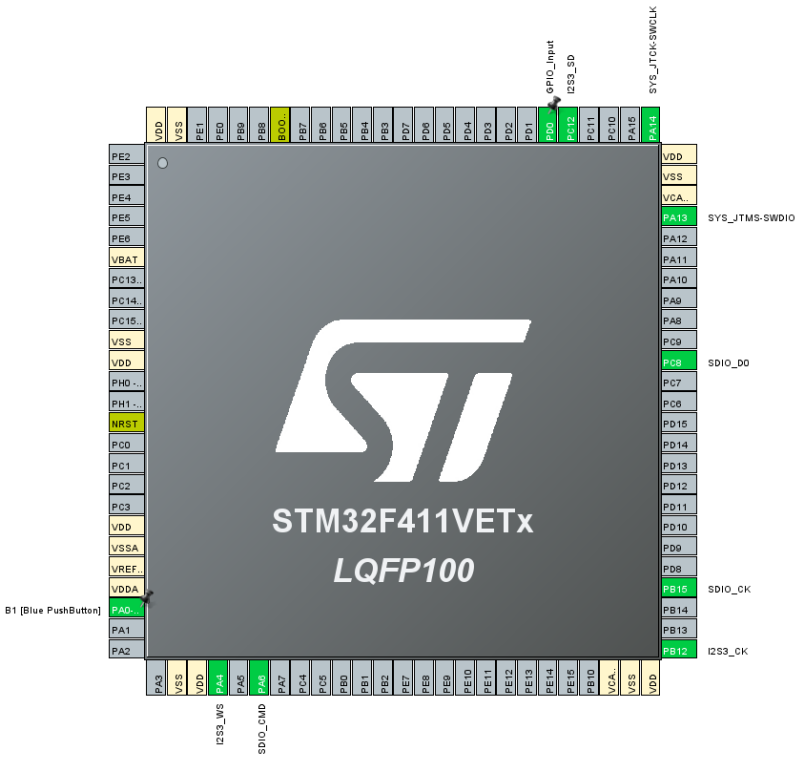


Figure 43: Board mạch STM32F411 sau khi đã cấu hình xong trên STM32 CubeIDE.

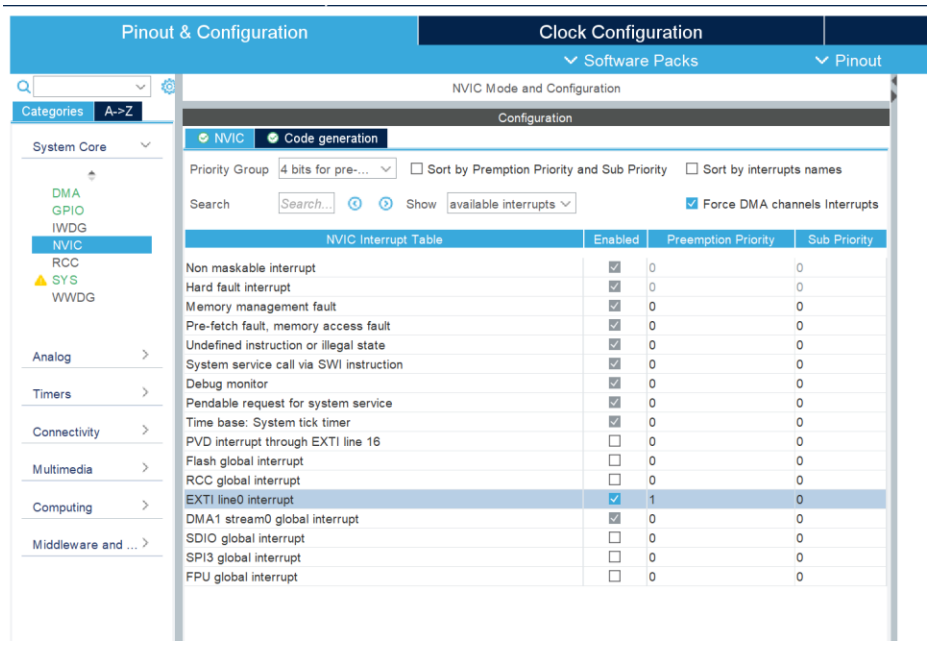


Figure 44: Đặt mức ưu tiên cho EXIT 0-line interrupt trong mục NVIC.

Bước 8: Sinh code bằng cách nhấn tổ hợp phím Alt+K.

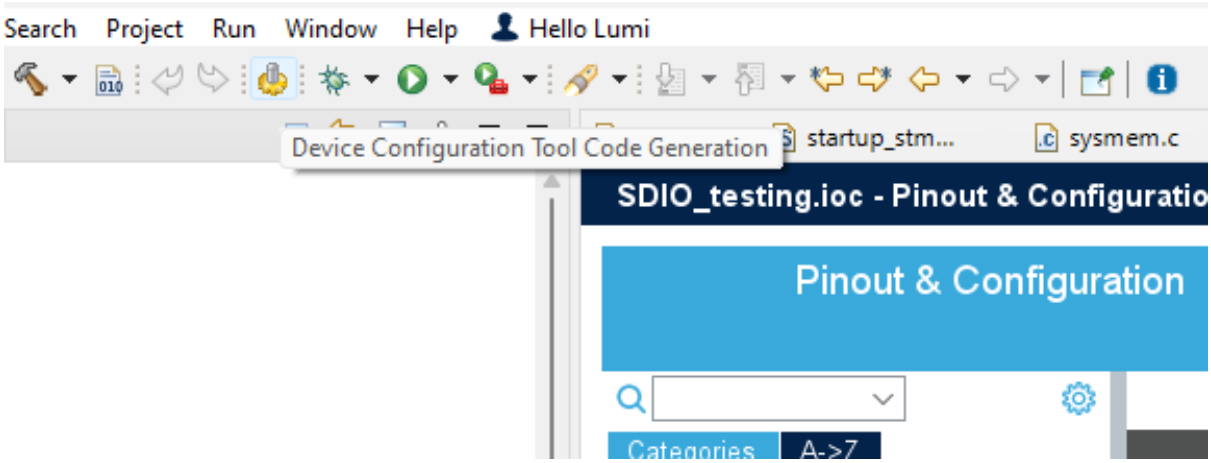


Figure 45: Thao tác trên thanh công cụ của phần mền để sinh code.

Bước 4: Lập trình bằng phần mềm STM32Cube IDE.

- Code để lưu dữ liệu từ microphone thông qua I2S vào SD card ở định dạng .wav.

1. *Tạo file audio_sd.h.*

Trong thư mục Core, kích chuột phải vào thư mục Inc và chọn New -> Header File và tạo file audio_sd.h.

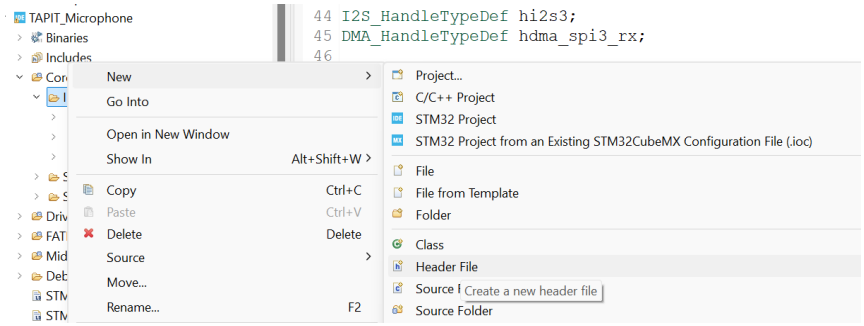


Figure 46: Thao tác tạo Header File mới.

Figure 43:

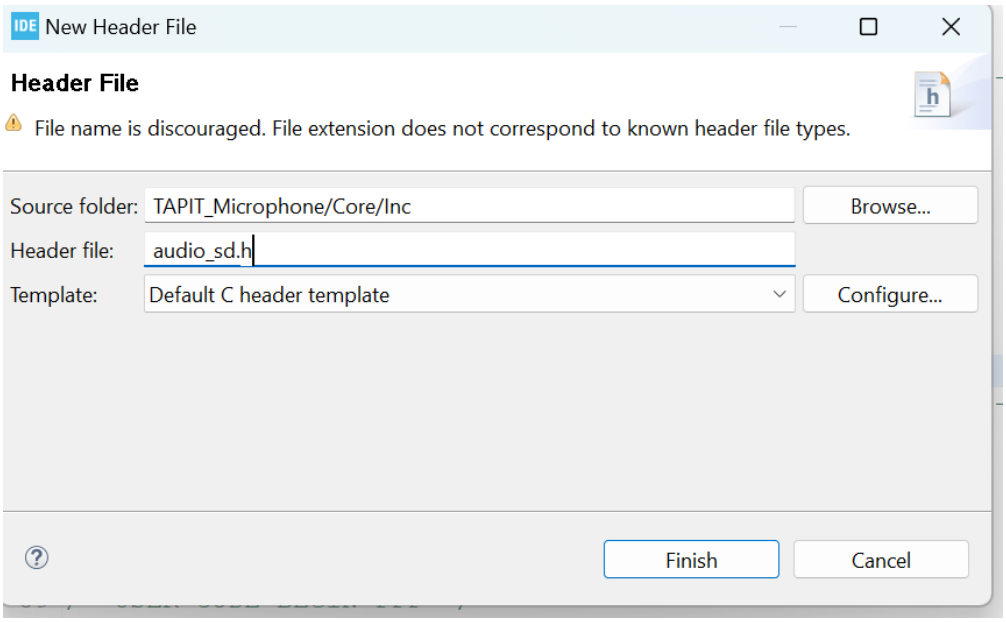


Figure 47: Đặt tên cho Header File mới.

#ifndef INC_AUDIO_SD_H_

#define INC_AUDIO_SD_H_

#include "stdio.h"

#include <stdint.h>

#define WAV_WRITE_SAMPLE_COUNT 2048 // buffer

void sd_card_init ();

void start_recording (uint32_t frequency);

void write2wave_file (uint8_t *ta, uint16_t data_size);

void stop_recording ();

#endif /* INC_AUDIO_SD_H_ */

- WAV_WRITE_SAMPLE_COUNT 2048 là bộ đệm được định nghĩa với giá trị 2048. Đây là số lượng mẫu âm thanh sẽ được ghi vào file WAV trong mỗi lần ghi. Con số này cho thấy kích thước của buffer dùng để lưu trữ dữ liệu âm thanh trước khi ghi vào thẻ SD.
- Khai báo 3 hàm tĩnh **start_recording (), write2wave_file (), stop_recording** ().

2. *Tạo file audio_sd.c*

Trong thư mục Core, kích chuột phải vào thư mục Src và chọn New -> Source File và tạo file audio_sd.c.

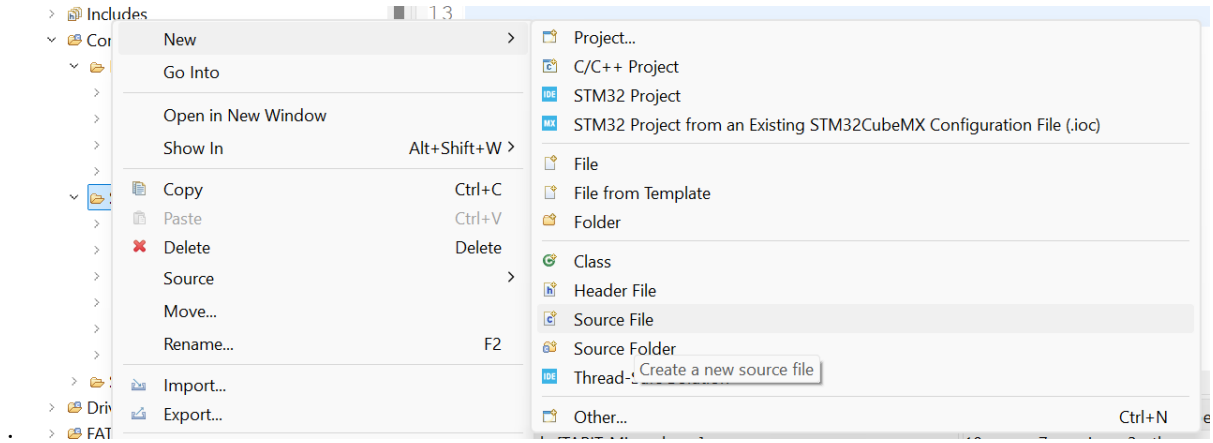


Figure 48: Thao tác tạo Header File mới.

#include "audio_sd.h"

#include "stdio.h"

#include "fatfs.h"

static uint8_t wav_file_header[44]={0x52, 0x49, 0x46, 0x46, 0xa4, 0xa9, 0x03, 0x00, 0x57 ,0x41, 0x56, 0x45, 0x66, 0x6d, 0x74, 0x20, 0x10, 0x00, 0x00, 0x00, 0x01, 0x00, 0x02, 0x00, 0x80, 0x7d, 0x00, 0x00, 0x00, 0xf4, 0x01, 0x00, 0x04, 0x00, 0x10, 0x00, 0x64, 0x61, 0x74, 0x61, 0x80, 0xa9, 0x03, 0x00};

static FRESULT sd_result;

static FATFS sdCard;

static FIL wavFile;

static uint32_t wav_file_size;

```
static uint8_t first_time = 0;
void sd_card_init()
{
	//	mounting an sd card
	sd_result = f_mount(&sdCard,SDPath, 1);
	if(sd_result != 0)
	{
		printf("error in mounting an sd card: %d \n", sd_result);
		while(1);
	}
	else
	{
		printf("succeeded in mounting an sd card \n");
	}
}
void start_recording(uint32_t frequency)
{
	static char file_name[] = "w_000.wav";
	static uint8_t file_counter = 10;
	int file_number_digits = file_counter;
	uint32_t byte_rate = frequency * 2 * 2;
	wav_file_header[24] = (uint8_t)frequency;
	wav_file_header[25] = (uint8_t)(frequency >> 8);
	wav_file_header[26] = (uint8_t)(frequency >> 16);
	wav_file_header[27] = (uint8_t)(frequency >> 24);
	wav_file_header[28] = (uint8_t)byte_rate;
	wav_file_header[29] = (uint8_t)(byte_rate >> 8);
	wav_file_header[30] = (uint8_t)(byte_rate >> 16);
```

```
        wav_file_header[31] = (uint8_t)(byte_rate >> 24);
        // defining a wave file name
        file_name[4] = file_number_digits%10 + 48;
        file_number_digits /= 10;
        file_name[3] = file_number_digits%10 + 48;
        file_number_digits /= 10;
        file_name[2] = file_number_digits%10 + 48;
        printf("file name %s \n", file_name);
        file_counter++;
        // creating a file
        sd_result = f_open(&wavFile ,file_name, FA_WRITE|FA_CREATE_ALWAYS);
        if(sd_result != 0)
        {
                printf("error in creating a file: %d \n", sd_result);
                while(1);
        }
        else
        {
                printf("succeeded in opening a file \n");
        }
        wav_file_size = 0;
}
void write2wave_file(uint8_t *data, uint16_t data_size)
{
        uint32_t temp_number;
        printf("w\n");
        if(first_time == 0)
        {
```

```
            for(int i = 0; i < 44; i++)
            {
                *(data + i) = wav_file_header[i];
            }
            first_time = 1;
        }
        sd_result = f_write(&wavFile,(void *)data, data_size,(UINT*)&temp_number);
        if(sd_result != 0)
        {
            printf("error in writing to the file: %d \n", sd_result);
            while(1);
        }
        wav_file_size += data_size;
}
void stop_recording()
{
        uint16_t temp_number;
        // updating data size sector
        wav_file_size -= 8;
        wav_file_header[4] = (uint8_t)wav_file_size;
        wav_file_header[5] = (uint8_t)(wav_file_size >> 8);
        wav_file_header[6] = (uint8_t)(wav_file_size >> 16);
        wav_file_header[7] = (uint8_t)(wav_file_size >> 24);
        wav_file_size -= 36;
        wav_file_header[40] = (uint8_t)wav_file_size;
        wav_file_header[41] = (uint8_t)(wav_file_size >> 8);
        wav_file_header[42] = (uint8_t)(wav_file_size >> 16);
        wav_file_header[43] = (uint8_t)(wav_file_size >> 24);
```

```
        // moving to the beginning of the file to update the file format
        f_lseek(&wavFile, 0);
        f_write(&wavFile,(void                                    *)wav_file_header,
sizeof(wav_file_header),(UINT*)&temp_number);
        if(sd_result != 0)
        {
                printf("error in updating the first sector: %d \n", sd_result);
                while(1);
        }
        f_close(&wavFile);
        first_time = 0;
        printf("closed the file \n");
}
```

Chúng ta cần một hàm để khai báo các giá trị tiêu đề của tệp .wav mà chúng ta cần ghi âm thanh vào. Ở đây mình khai báo hàm có 44-byte mã ascii và bạn có thể tham khảo.

Vị trí	Mô tả	Giá trị mẫu	Giá trị byte
0 - 3	“RIFF”		{0x52, 0x49, 0x46, 0x46}
4 - 7	Kích thước tệp (số nguyên)		{data_section size + 36}
8 - 11	“WAVE”		{0x57 ,0x41, 0x56, 0x45}
12 - 15	“fmt "		{0x66, 0x6d, 0x74, 0x20}
16 - 19	Độ dài của dữ liệu	16	{0x10, 0x00, 0x00, 0x00}
20 - 21	Loại định dạng (1 là PCM) - 2 byte số nguyên	1	{0x01 0x00}

22 - 23	Số kênh - Số nguyên 2 byte	2	{0x02 0x00}
24 - 27	Tốc độ mẫu - số nguyên 32 byte. Các giá trị phổ biến là 44100 (CD), 48000 (DAT). Tỷ lệ mẫu = Số lượng mẫu mỗi giây, hoặc Hertz.	32 kHz	{0x80, 0x7d, 0x00, 0x00}
28 - 31	(Tỷ lệ mẫu * BitsPerSample * Kênh) / 8.	19200	{0x00, 0xf4, 0x01, 0x00 }
32 - 33	(BitsPerSample * Channels) / 8.1 - 8 bit mono2 - 8 bit stereo/16 bit mono4 - 16 bit stereo	4	{0x04, 0x00}
34 - 35	Số bit trên mỗi mẫu	16	{0x10, 0x00}
36 - 39	“dữ liệu” tiêu đề chunk. Đánh dấu phần đầu của phần dữ liệu.		{0x64, 0x61, 0x74, 0x61}
40 - 43	Kích thước của phần dữ liệu.		

File .wav mình sử dụng là loại tệp âm thanh 16bit, tần số lấy mẫu là 32kHz và chế độ stereo.

Hàm sd_card_init() có nhiệm vụ khởi tạo và gắn thẻ SD

Hàm start_recording() có nhiệm vụ bắt đầu quá trình ghi âm bằng cách mở tệp để ghi với quyền FA_WRITE|FA_CREATE_ALWAYS

Chúng tôi sẽ khai báo tên của file wav mà tôi sẽ lưu trữ âm thanh vào “w_000.wav” và khai báo biến tĩnh file_counter và tăng số đếm mỗi lần gọi để khởi tạo các file tiếp theo“w_001.wav”, “w_002.wav”, tránh tình trạng dữ liệu bị ghi đè lên những file wav trước đó.

Trong hàm start_recording(), chúng ta có biến frequency tần số là một đại lượng biến thiên. Chính vì thế chúng ta cập nhật các phần tử bằng cách dịch bit của biến frequency sang phải để trích xuất tất cả giá trị byte của tần số mà mic thu âm được gán vào trong phần tử 24 đến 27, .

Ta có tốc độ bit = tần số lấy mẫu x BitPerSecond x số kênh, mà trong bài toán chọn bps = số kênh = 2. Cho nên byte_rate = frequency * 2 * 2. Tương tự như như biến frequency, ta cũng dịch biến byte_rate để cập nhật thông số trong các phần tử từ 28 đến 31.

Hàm write2wave_file() có chức năng ghi dữ liệu âm thanh vào tệp WAV ghi dữ liệu vào tệp bằng cách sử dụng f_write. Ngoài ra hàm còn cập nhật kích thước tệp. Nếu ghi dữ liệu lần đầu (first_time = 0) thì sao chép phần tiêu đề của wav đã được khai báo sẵn vào header file WAV.

Hàm stop_recording() có chức năng kết thúc ghi âm thanh bằng cách đóng tệp và reset biến first_time. Đồng thời hàm còn quay lại cập nhật kích thước dữ liệu ở header 4 đến 7 và 30 đến 43.

3. Code cho hàm main

Include the headers file: audio_sd.h để ghi dữ liệu âm thanh nhận được vào SD card và stdio.h cho hàm printf() để hiển thị thông báo ra console window.

```
/* USER CODE BEGIN Includes */

#include "stdio.h"

#include "audio_sd.h"

/* USER CODE END Includes */
```

Tạo Object của class FATFS and và khai báo các biến cần thiết :

```
/* USER CODE BEGIN PV */

FATFS fatfs;

FIL myfile;

FRESULT fresult;

int16_t data_i2s[WAV_WRITE_SAMPLE_COUNT];/*array to store received audio data */

volatile int16_t sample_i2s;

volatile uint8_t button_flag, start_stop_recording;

volatile uint8_t  half_i2s, full_i2s;

/* USER CODE END PV */
```

Code tạo file . txt để kiểm tra kết nối SD card.

```
 /* USER CODE BEGIN 2 */

 if(BSP_SD_Init() == MSD_OK)

 {

      fresult = f_mount(&fatfs,"",1);
```

```
fresult=f_open(&myfile,"sdio23.txt",FA_CREATE_ALWAYS|FA_WRITE);
      f_printf(&myfile, "Create file to test SD card connection ");
      f_close(&myfile);
 }
 /* USER CODE END 2 */
```

Code cho hàm main để nhấn nút ấn là bắt đầu ghi âm và nhấn nút ấn lần 2 sẽ dừng ghi âm. Và trong quá trình DMA điều khiển khối I2S lưu dữ liệu âm thanh thu được vào mảng đã khai báo, nếu một nữa mảng đã được lưu thì sẽ vào ngắt **HAL_I2S_RxHalfCplt Callback** để thay đổi giá trị của biến half_i2s = 1; và sau đó ghi nữa mảng dữ liệu này vào thẻ SD. Trong khi vi xử lí điều khiển hoạt động ghi vào thẻ nhớ, thì DMA vẫn tiếp điều khiển khối I2S nhận dữ liệu âm thanh và lưu vào nữa còn lại của mảng. Khi nữa còn lại đã được lưu đầy, tức là mảng dữ liệu được khai báo đã được lưu đây, chương trình sẽ vào ngắt HAL**_I2S_RxCplt Callback** để thay đổi giá trị của biến full_i2s = 1; và sau đó ghi nữa mảng dữ liệu này vào thẻ SD và giống như trên thì DMA vẫn tiếp điều khiển khối I2S nhận dữ liệu âm thanh và lưu vào nữa còn lại của mảng để tránh tình trạng thất thoát tín hiệu. Với cách nhận và ghi như trên tức là DMA và vi xử lí hoạt động cùng lúc giúp việc ghi âm hiệu quả mà không bị thất thoát tín hiệu. Sơ đồ thuật toán được thể hiện ở hình:

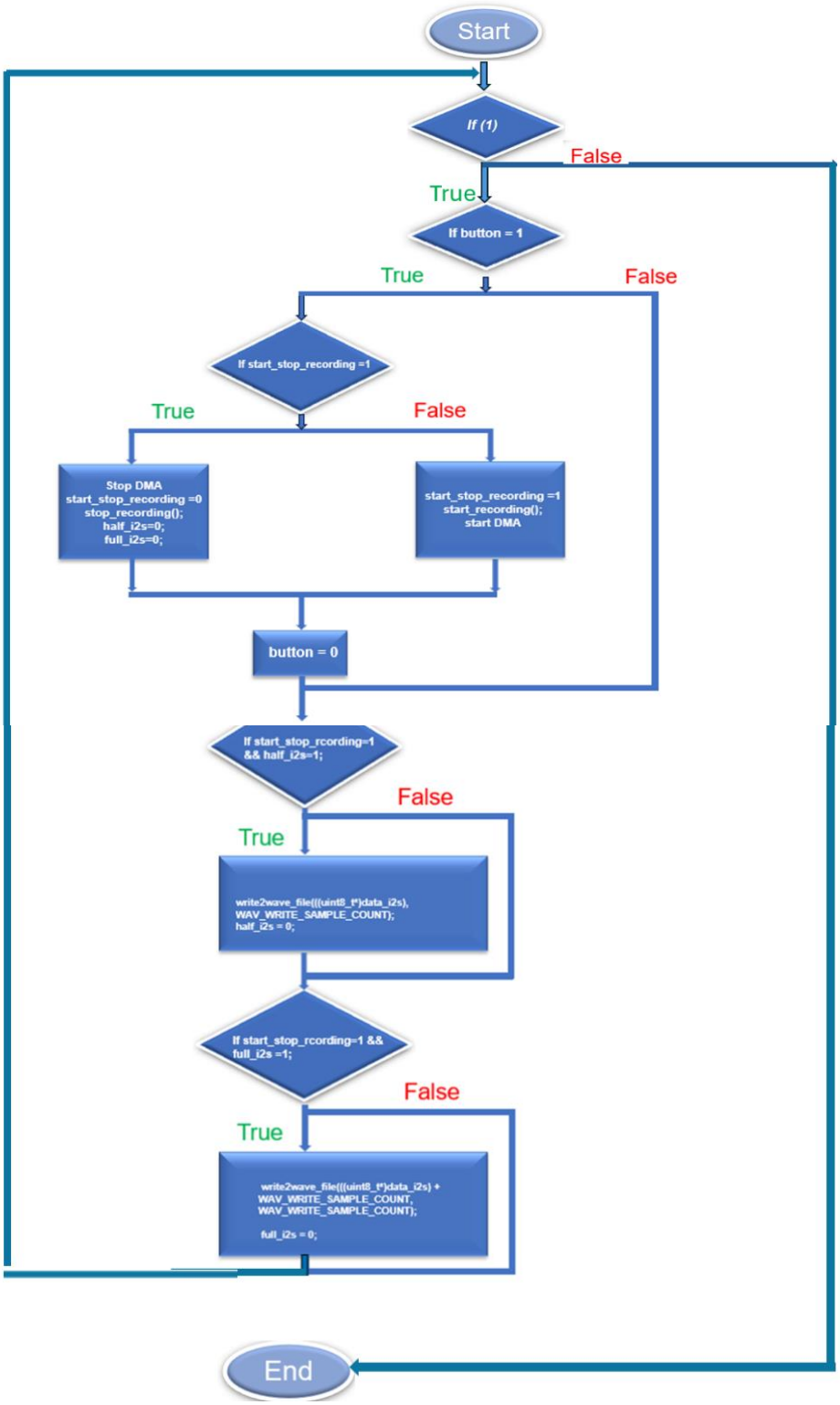


Figure 49: Lưu đồ thuật toán của hàm main.

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
        if(button_flag)
        {
                if(start_stop_recording)
                {
                        HAL_I2S_DMAStop(&hi2s3);
                        start_stop_recording = 0;
                        stop_recording();
                        half_i2s = 0;
                        full_i2s = 0;
                        printf("stop recording \n");
                }
                else
                {
                        start_stop_recording = 1;
                        start_recording(I2S_AUDIOFREQ_32K);
                       printf("start_recording %dand %d\n", half_i2s, full_i2s);
                  HAL_I2S_Receive_DMA(&hi2s3,(uint16_t*)data_i2s,
sizeof(data_i2s)/2);
                }
                button_flag = 0;
        }
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
if(start_stop_recording == 1 && half_i2s == 1)
```

```
            {
            write2wave_file(((uint8_t*)data_i2s),  WAV_WRITE_SAMPLE_COUNT);
            half_i2s = 0;
            }
if(start_stop_recording == 1 && full_i2s == 1)
            {
            write2wave_file(((uint8_t*)data_i2s+ WAV_WRITE_SAMPLE_COUNT),
WAV_WRITE_SAMPLE_COUNT);
             full_i2s = 0;
            }
 }
 /* USER CODE END 3 */
```

Code để dùng hàm prinft() hiện thị thông báo ra console window và các hàm callback:

```
/* 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;
}
void HAL_I2S_RxCpltCallback(I2S_HandleTypeDef *hi2s)
{
        full_i2s = 1;
}
void HAL_I2S_RxHalfCpltCallback(I2S_HandleTypeDef *hi2s)
{
```

```
        half_i2s = 1;
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        if(GPIO_Pin == B1_Pin)
                {
                        button_flag = 1;
                }
// code chong nhieu
HAL_Delay(50);
uint32_t tpre = HAL_GetTick();
while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)== GPIO_PIN_RESET)
 {
         if(HAL_GetTick()-tpre >=3000)
         {
          break;
         }
 }
 HAL_Delay(50);
__HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_Pin);
}
/* USER CODE END 4 */
```

Bước 5: Setup màn hình SWV ITM Data console.

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

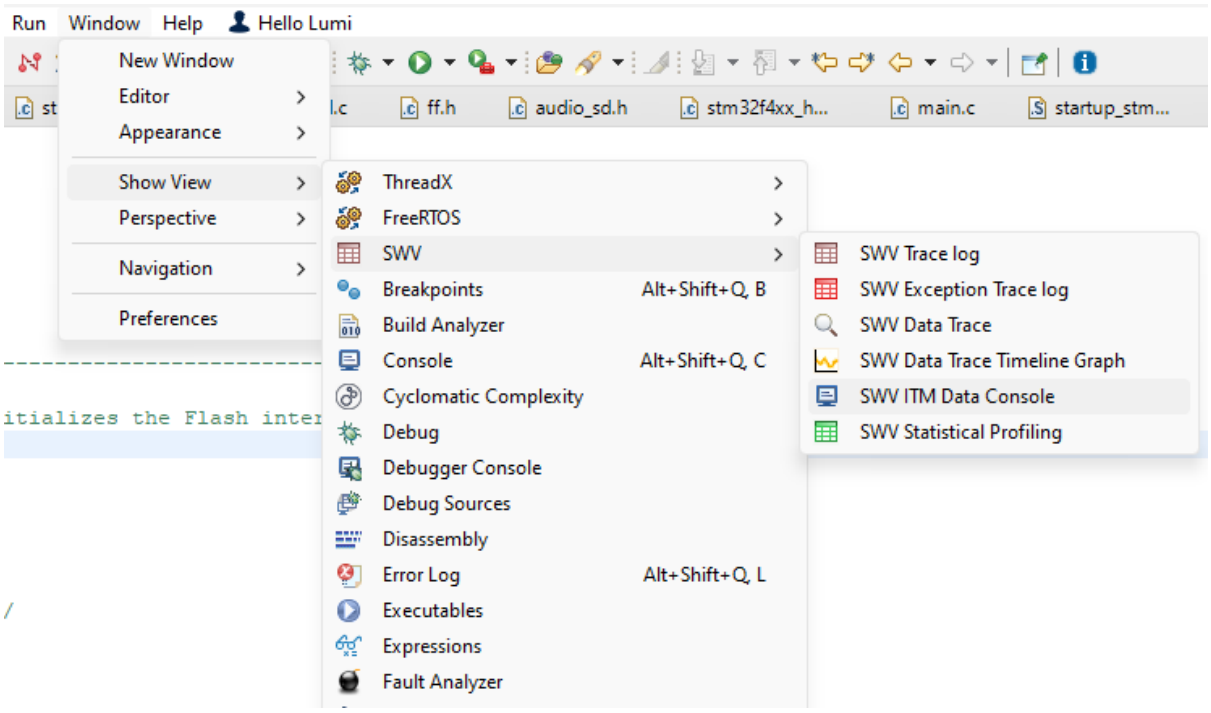


Figure 50: Cách mở của sổ SWV ITM Data Console để xem kết quả của các câu lệnh dùng hàm printf ().

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

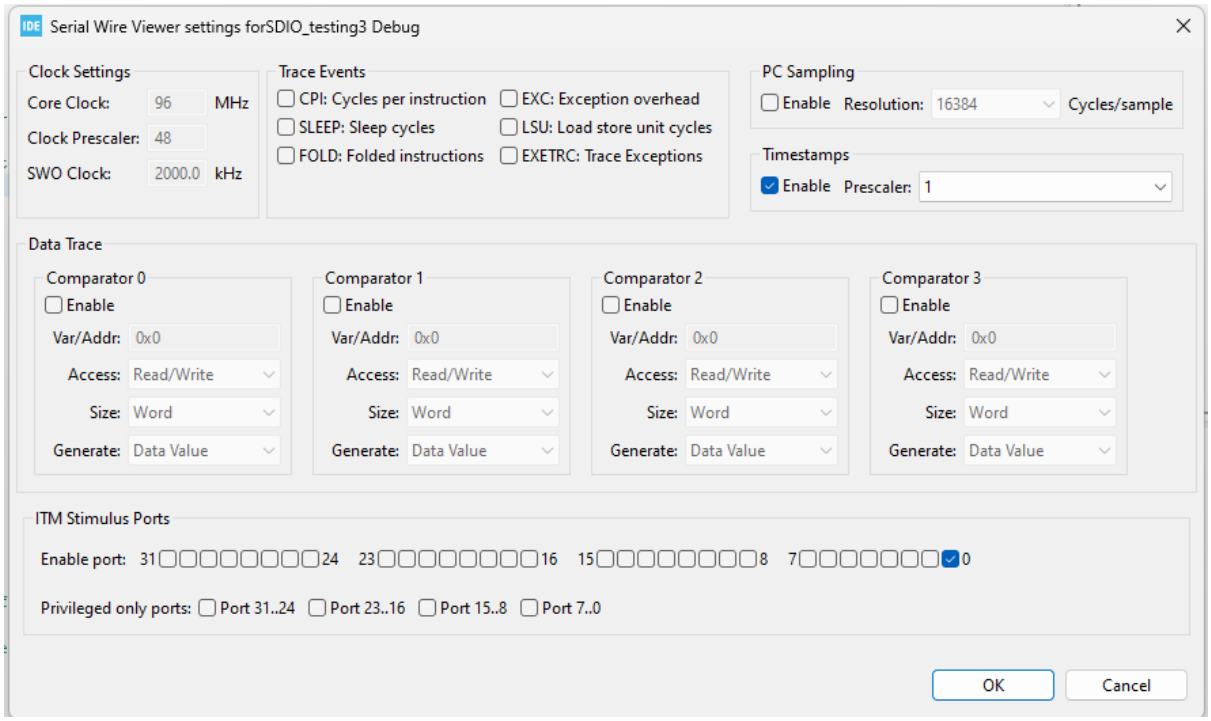


Figure 51: Enable port 0.

- Nhấn nút Start Trace

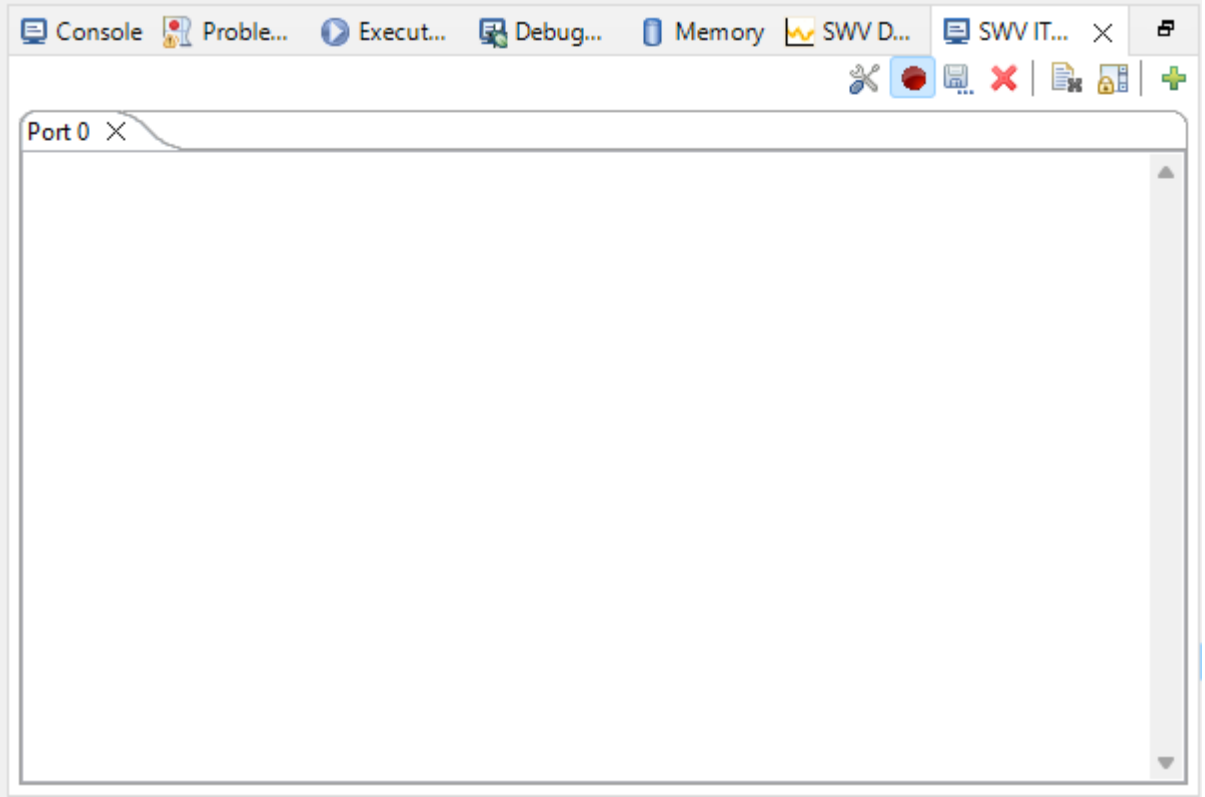


Figure 52: Vị trị nút Start Trace trên cửa sổ SWV ITM Data Console.

Bước 6: 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

⇨ **Kết quả:**

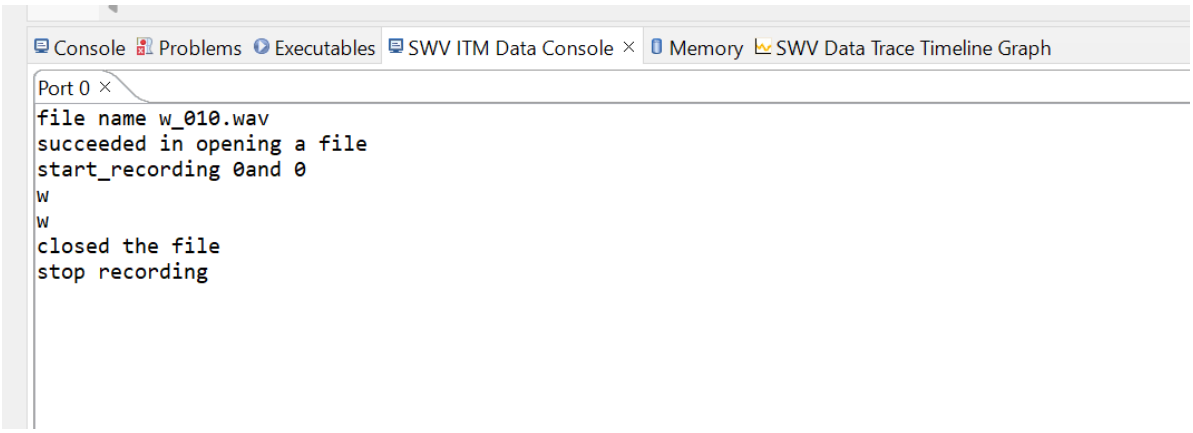


Figure 53:Kết quả hiện thị trên cửa sổ SWV ITM Data Console

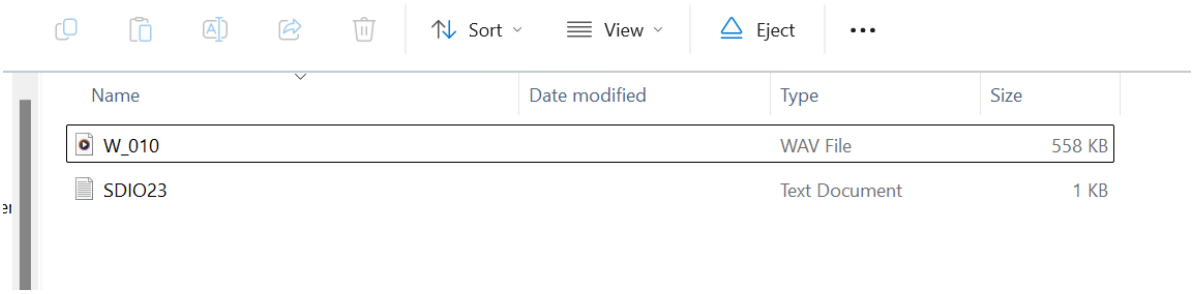


Figure 54: File được tạo trên thẻ nhớ SD.

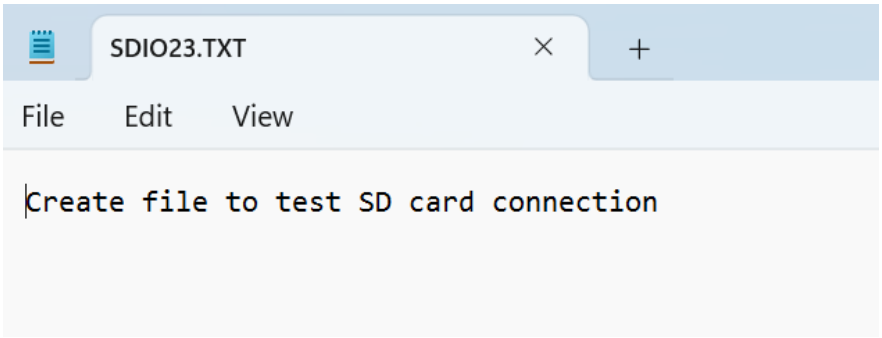


Figure 55: Nội dùng của file SDIO23.TXT.

7. Một số lỗi

- **Lỗi in chữ Trung Quốc.**

Figure 56: Lỗi các dòng chữ trung quốc hiển thị trên cửa sổ SWV ITM Data Console

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 80 MHz.

- **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).

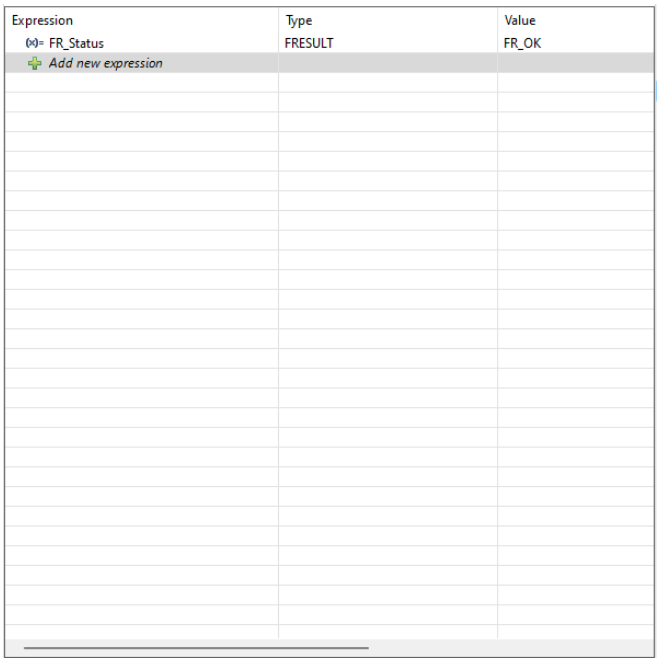
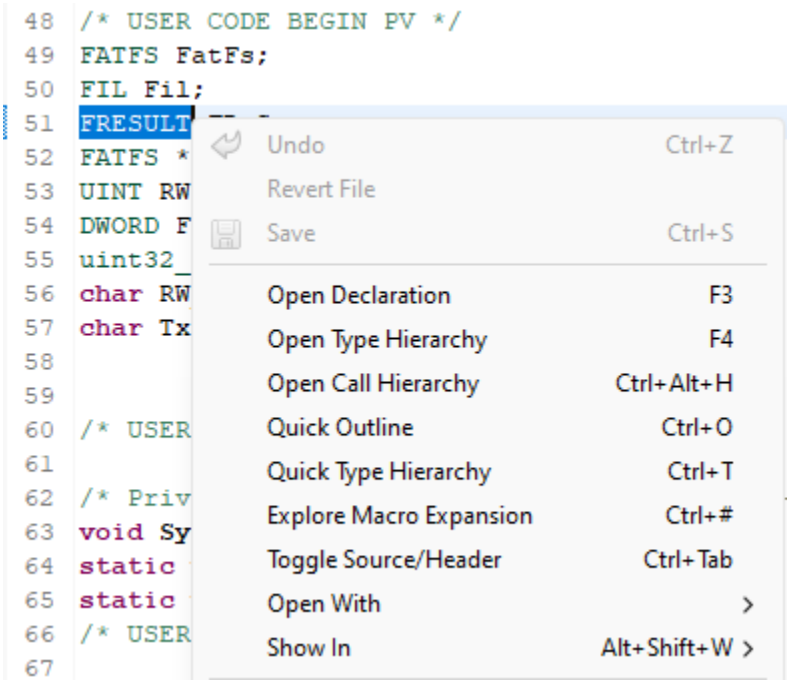


Figure 57: Giá trị được lưu trong biến FR_Status.

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.



```
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 58:Các kết quả trả về của một biến kiểu FRESULT.

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 lí (nối nhầm 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.

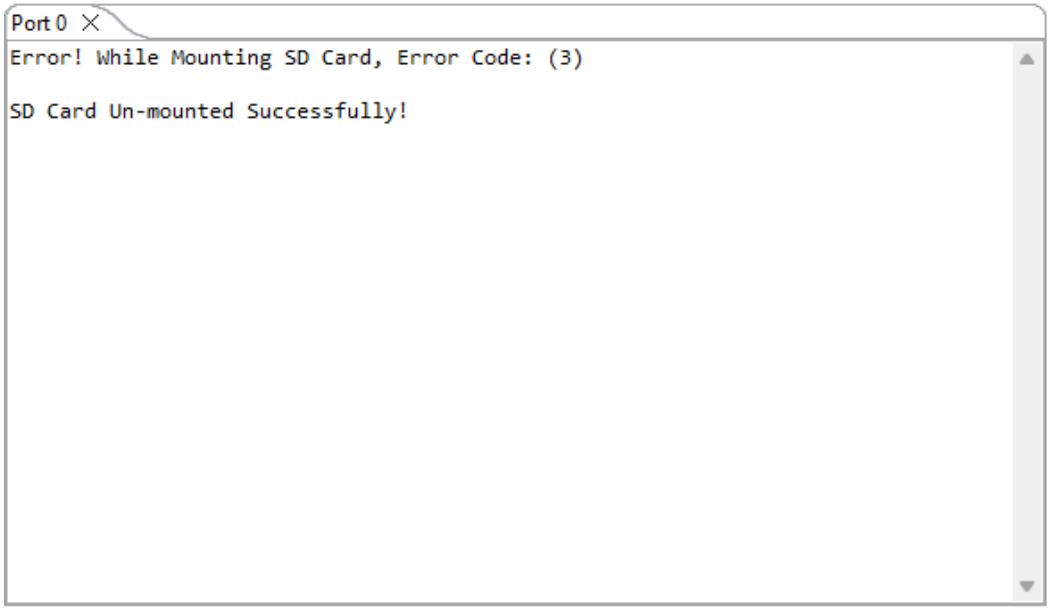


Figure 59: Thông báo lỗi số 3-FR_NOT_READY.

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.

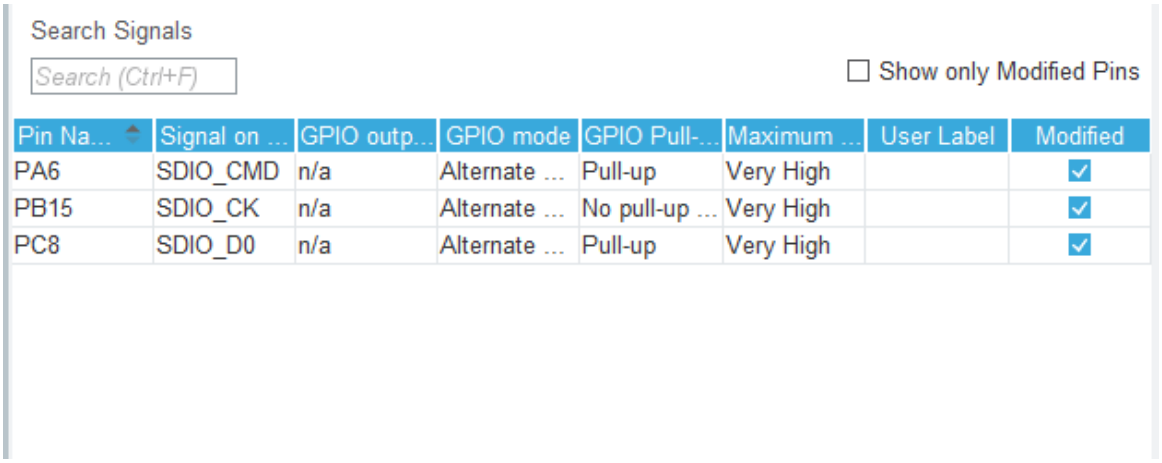


Figure 60: Thay đổi về chế độ Pull-up cho các chân SDIO trừ chân SDIO_CK.

- **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 Fil 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 xoá bộ nhớ flash bằng phần mềm STM32CubeProgrammer.

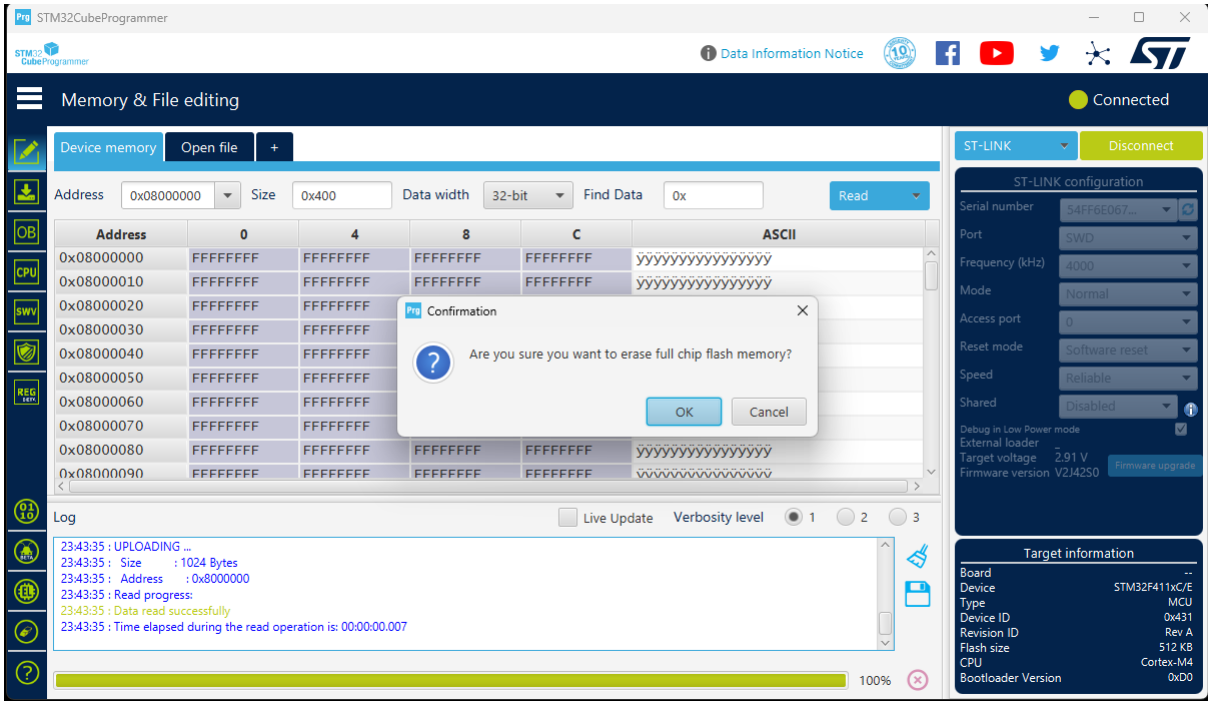


Figure 61: Xóa bộ nhớ của vi điều khiển bằng phần mềm STM32CubeProgrammer.

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 tần hệ số CLKDIV nếu bạn gặp lỗi trong khi quá trình gắn thẻ SD.

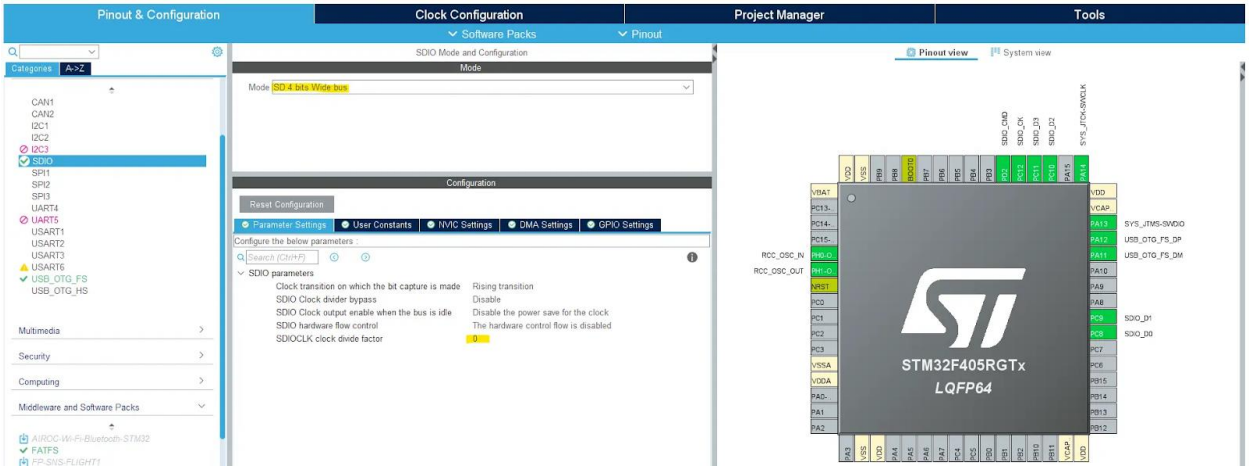


Figure 62: Thay đổi tần số clock của SDIO bằng SDIOCLLK clock divide factor.

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)

https://rdsic.edu.vn/blog/hoa/nhung-ung-dung-va-loi-ich-cua-i2s-protocol-ban-khong -the-bo-qua-vi-cb.html