Ha Noi University of Science and Technology School of Electronic and Telecommunication

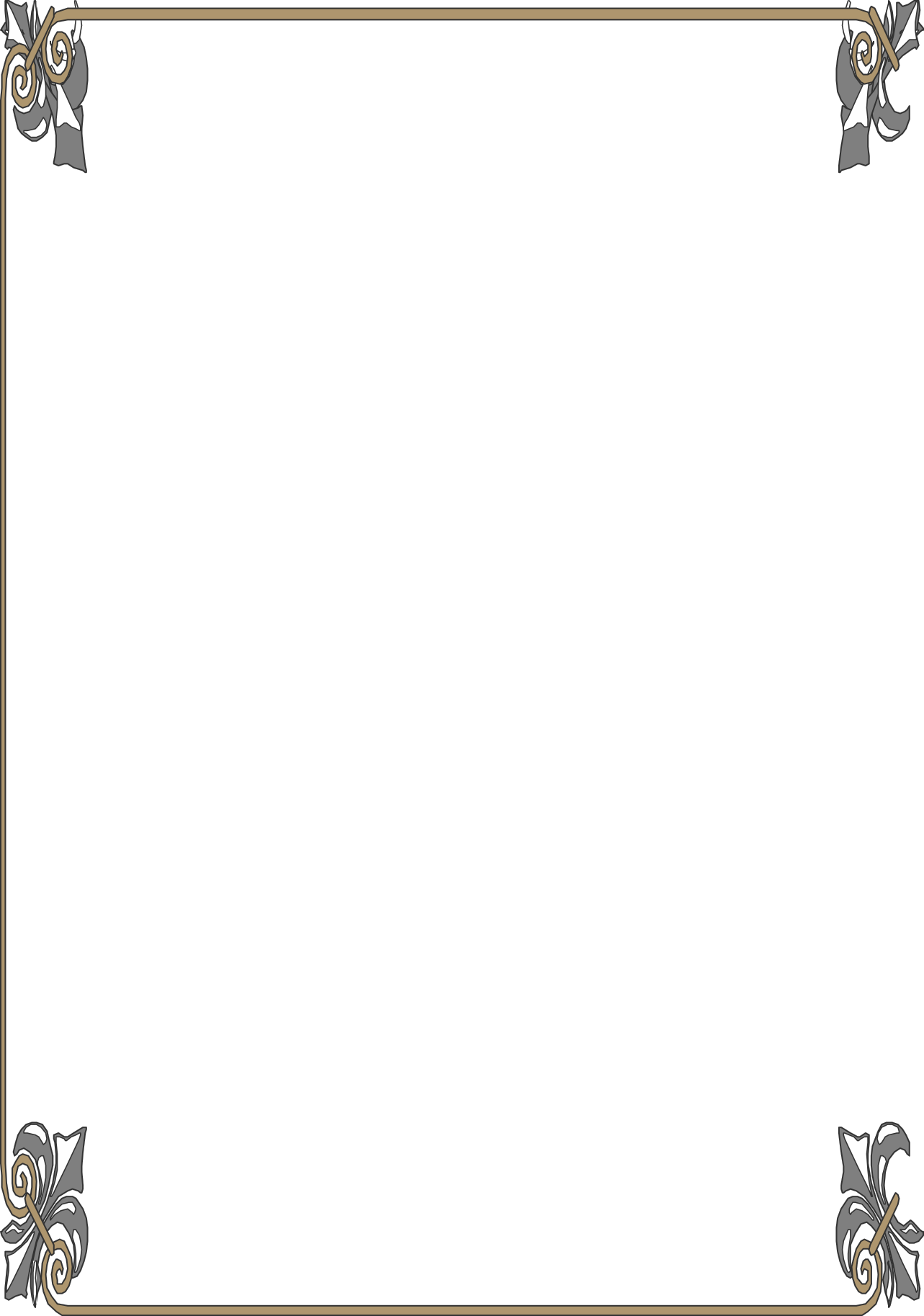
..…oOo…..

**Engineer Graduated Thesis**

## Project:

**Audio Recorder and**

**Playback with STM32F4**



## Student: Trần Văn Hà

**Student ID: 20141325**

**Class: ĐT 04-K59**

**Instructor: Dr. Lê Dũng**

**Ha Noi, 06/2019**

Ha Noi University of Science and Technology School of Electronic and Telecommunication

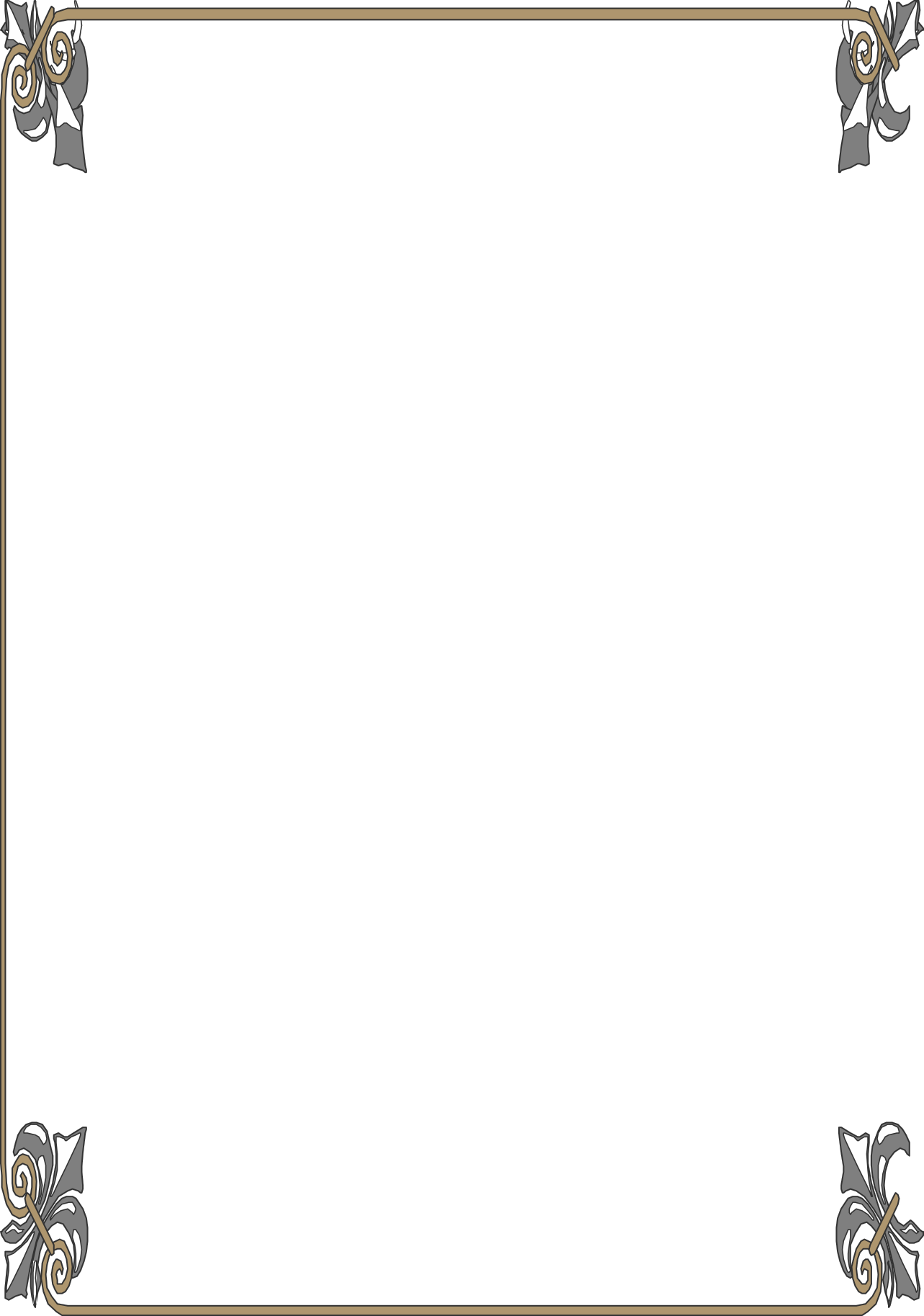
..…oOo…..

**Engineer Graduated Thesis**

## Project:

**Audio Recorder and**

**Playback with STM32F4**



## Student: Trần Văn Hà

**Class: ĐT 04-K59**

**Instructor: Dr. Lê Dũng Examiner:………………………………………..**

**Ha Noi, 06/2019**

**Đánh giá quyển đồ án tốt nghiệp (Dùng cho giảng viên hướng dẫn)**

Giảng viên đánh giá:……...**Dr. Lê Dũng**………………………. Họ và tên Sinh viên:.……..**Trần Văn Hà**……………………… MSSV:……..……………..**20141325**…………………………… Tên đồ án: **Audio recorder and playback with STM32F4**

***Chọn các mức điểm phù hợp cho sinh viên trình bày theo các tiêu chí dưới đây:***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Có sự kết hợp giữa lý thuyết và thực hành (20)** | | | | | | |
| 1 | Nêu rõ tính cấp thiết và quan trọng của đề tài, các vấn đề và các giả  thuyết (bao gồm mục đích và tính phù hợp) cũng như phạm vi ứng dụng của đồ án | 1 | 2 | 3 | 4 | 5 |
| 2 | Cập nhật kết quả nghiên cứu gần đây nhất (trong nước/quốc tế) | 1 | 2 | 3 | 4 | 5 |
| 3 | Nêu rõ và chi tiết phương pháp nghiên cứu/giải quyết vấn đề | 1 | 2 | 3 | 4 | 5 |
| 4 | Có kết quả mô phỏng/thưc nghiệm và trình bày rõ ràng kết quả đạt được | 1 | 2 | 3 | 4 | 5 |
| **Có khả năng phân tích và đánh giá kết quả (15)** | | | | | | |
| 5 | Kế hoạch làm việc rõ ràng bao gồm mục tiêu và phương pháp thực hiện dựa trên kết quả nghiên cứu lý thuyết một cách có hệ thống | 1 | 2 | 3 | 4 | 5 |
| 6 | Kết quả được trình bày một cách logic và dễ hiểu, tất cả kết quả đều được phân tích và đánh giá thỏa đáng. | 1 | 2 | 3 | 4 | 5 |
| 7 | Trong phần kết luận, tác giả chỉ rõ sự khác biệt (nếu có) giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận  để đề xuất hướng giải quyết có thể thực hiện trong tương lai. | 1 | 2 | 3 | 4 | 5 |
| **Kỹ năng viết (10)** | | | | | | |
| 8 | Đồ án trình bày đúng mẫu quy định với cấu trúc các chương logic và đẹp mắt (bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy v.v), có mở đầu chương và kết luận  chương, có liệt kê tài liệu tham khảo và có trích dẫn đúng quy định | 1 | 2 | 3 | 4 | 5 |
| 9 | Kỹ năng viết xuất sắc (cấu trúc câu chuẩn, văn phong khoa học, lập luận logic và có cơ sở, từ vựng sử dụng phù hợp v.v.) | 1 | 2 | 3 | 4 | 5 |
| **Thành tựu nghiên cứu khoa học (5) (chọn 1 trong 3 trường hợp)** | | | | | | |
| 10a | Có bài báo khoa học được đăng hoặc chấp nhận đăng/đạt giải SVNC khoa học giải 3 cấp Viện trở lên/các giải thưởng khoa học (quốc tế/trong nước) từ giải 3 trở lên/ Có đăng ký bằng phát minh  sáng chế | | 5 | | | |
| 10b | Được báo cáo tại hội đồng cấp Viện trong hội nghị sinh viên nghiên cứu khoa học nhưng không đạt giải từ giải 3 trở lên/Đạt giải khuyến khích trong các kỳ thi quốc gia và quốc tế khác về  chuyên ngành như TI contest. | | 2 | | | |
| 10c | Không có thành tích về nghiên cứu khoa học | | 0 | | | |
| **Điểm tổng** | | | **/50** | | | |
| **Điểm tổng quy đổi về thang 10** | | |  | | | |

***3. Nhận xét thêm của Thầy/Cô (****giảng viên hướng dẫn nhận xét về thái độ và tinh thần làm việc của sinh viên****)***

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

Ngày /06/2019 Người nhận xét

(Ký và ghi rõ họ tên)

## Đánh giá quyển đồ án tốt nghiệp (Dùng cho cán bộ phản biện)

Giảng viên đánh giá:……………………………………………………………… Họ và tên Sinh viên**:**……….**Trần Văn Hà**…………………................................ MSSV:……………………..**20141325**…………..………………………………. Tên đồ án: **Audio Recorder and Playback with STM32F4**

***Chọn các mức điểm phù hợp cho sinh viên trình bày theo các tiêu chí dưới đây: Rất kém (1); Kém (2); Đạt (3); Giỏi (4); Xuất sắc (5)***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Có sự kết hợp giữa lý thuyết và thực hành (20)** | | | | | | | |
| 1 | Nêu rõ tính cấp thiết và quan trọng của đề tài, các vấn đề và các giả thuyết (bao gồm mục đích và tính phù hợp) cũng như phạm vi ứng  dụng của đồ án | | 1 | 2 | 3 | 4 | 5 |
| 2 | Cập nhật kết quả nghiên cứu gần đây nhất (trong nước/quốc tế) | | 1 | 2 | 3 | 4 | 5 |
| 3 | Nêu rõ và chi tiết phương pháp nghiên cứu/giải quyết vấn đề | | 1 | 2 | 3 | 4 | 5 |
| 4 | Có kết quả mô phỏng/thưc nghiệm và trình bày rõ ràng kết quả đạt được | | 1 | 2 | 3 | 4 | 5 |
| **Có khả năng phân tích và đánh giá kết quả (15)** | | | | | | | |
| 5 | Kế hoạch làm việc rõ ràng bao gồm mục tiêu và phương pháp thực  hiện dựa trên kết quả nghiên cứu lý thuyết một cách có hệ thống | | 1 | 2 | 3 | 4 | 5 |
| 6 | Kết quả được trình bày một cách logic và dễ hiểu, tất cả kết quả đều được phân tích và đánh giá thỏa đáng. | | 1 | 2 | 3 | 4 | 5 |
| 7 | Trong phần kết luận, tác giả chỉ rõ sự khác biệt (nếu có) giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận để đề xuất hướng giải quyết có thể thực hiện trong tương lai. | | 1 | 2 | 3 | 4 | 5 |
| **Kỹ năng viết (10)** | | | | | | | |
| 8 | Đồ án trình bày đúng mẫu quy định với cấu trúc các chương logic và đẹp mắt (bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy v.v), có mở đầu chương và kết luận  chương, có liệt kê tài liệu tham khảo và có trích dẫn đúng quy định | | 1 | 2 | 3 | 4 | 5 |
| 9 | Kỹ năng viết xuất sắc (cấu trúc câu chuẩn, văn phong khoa học, lập luận logic và có cơ sở, từ vựng sử dụng phù hợp v.v.) | | 1 | 2 | 3 | 4 | 5 |
| **Thành tựu nghiên cứu khoa học (5) (chọn 1 trong 3 trường hợp)** | | | | | | | |
| 10a | | Có bài báo khoa học được đăng hoặc chấp nhận đăng/đạt giải SVNC khoa học giải 3 cấp Viện trở lên/các giải thưởng khoa học (quốc tế/trong nước) từ giải 3 trở lên/ Có đăng ký bằng phát minh sáng chế | 5 | | | | |
| 10b | | Được báo cáo tại hội đồng cấp Viện trong hội nghị sinh viên nghiên cứu khoa học nhưng không đạt giải từ giải 3 trở lên/Đạt giải khuyến khích trong các kỳ thi quốc gia và quốc tế khác về  chuyên ngành như TI contest. | 2 | | | | |
| 10c | | Không có thành tích về nghiên cứu khoa học | 0 | | | | |
| **Điểm tổng** | | | **/50** | | | | |
| **Điểm tổng quy đổi về thang 10** | | |  | | | | |

***3. Nhận xét thêm của Thầy/Cô***

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

.......................................................................................................................................................

Ngày …/06/2019 Người nhận xét (Ký, ghi rõ họ tên)

# Lời nói đầu

Trước tiên, em xin bày tỏ lòng biết ơn sâu sắc đến giảng viên hướng dẫn của mình, Tiến sĩ Lê Dũng, Bộ môn Điện Tử và Kỹ Thuật Máy Tính, Viện Điện Tử Viễn Thông, Trường Đại Học Bách Khoa Hà Nội. Trong suốt thời gian em thực hiện đồ án, Thầy đã tận tình hướng dẫn, đưa ra hướng phát triển cũng như cách giải quyết các khó khắn để em hoàn thành đồ án đúng hạn.

Tôi cũng xin bày tỏ long biết ơn đến tập thể anh em trong sự án SDK, FGA.DPA, tập đoàn FPT. Trong thời gian tôi làm việc tại đây, cũng chính là thời gian tôi làm đồ án tốt nghiệp, các anh chị Project manager, Technical Lead đã cung cấp cho tôi các điều kiện về cơ sở vật chất cũng linh kiện, thiết bị. Các anh em developer đã luôn tận tình giúp đỡ, chỉ bảo cho tôi không chỉ những kiến thức về lập trình, kiến trúc vi điều khiển mà còn cả những kỹ năng làm việc nhóm, kỹ năng tiếng anh, ký năng giao tiếp với khách hang hay hiểu biết về quy trình phát triển phần mềm. Đó thực sự là những kiến thức và trải nghiệm rất quý giá mà nhà trường không bao giờ dạy. Những điều đó giúp tôi tự tin hơn rất nhiều và vững bước trên con đường tôi đã chọn, con đường trở thành một kỹ sư lập trình nhúng.

Sau cùng, Con xin bày tỏ lòng biết ơn sâu sắc đến bố, mẹ. Những người luôn đứng sau, ủng hộ con vô điều kiện và là động lực giúp con vượt qua những khó khăn. Không có gia đình phía sau, con không thể có được những thành quả như ngày hôm nay.

Ha Noi, ngày 10 tháng 06 năm 2019

Ha Tran Van

**Lời Cam Đoan**

Tôi là Trần Văn Hà, mã số sinh viên 20141325, sinh viên lớp DT 04, khóa 59. Người hướng dẫn là TS. Lê Dũng. Tôi xin cam đoan toàn bộ nội dung được trình bày trong đồ án *lập trình thiết bị thu âm và phát nhạc với STM32F4* là kết quả quá trình tìm hiểu và nghiên cứu của tôi. Các dữ liệu được nêu trong đồ án là hoàn toàn trung thực, phản ánh đúng kết quả đo đạc thực tế. Mọi thông tin trích dẫn đều tuân thủ các quy định về sở hữu trí tuệ; các tài liệu tham khảo được liệt kê rõ ràng. Tôi xin chịu hoàn toàn trách nhiệm với những nội dung được viết trong đồ án này.

|  |  |
| --- | --- |
|  | Hà nội, ngày 10 tháng 06 năm 2019  **Người cam đoan**  **Trần Văn Ha** |

**Mục Lục**

[ACKNOWLEDGEMENTS 7](#_bookmark0)

[Commitment 8](#_bookmark1)

[Contents 9](#_bookmark2)

[Acronyms 11](#_bookmark3)

[List of Figure 12](#_bookmark4)

[List of Table 14](#_bookmark5)

[ABSTRACT 15](#_bookmark6)

[CHAPTER I: PROJECT SUMMARY 16](#_bookmark7)

* 1. [Project summary 16](#_bookmark8)
  2. [Aims of research 16](#_bookmark9)
  3. [Research methods 17](#_bookmark10)
  4. [Conclusion 17](#_bookmark11)

[CHAPTER II: UNDERSTANDING THE THEORY 18](#_bookmark12)

* 1. [FAT File system 19](#_bookmark13)
     1. [History of FAT File system 19](#_bookmark14)
     2. [Technical overview 19](#_bookmark15)
     3. [FAT File system layout 20](#_bookmark16)
  2. [Digital to Analog Converter 22](#_bookmark18)
     1. [DAC peripheral on STM32F407VG MCU 23](#_bookmark19)
     2. [DAC functional description 24](#_bookmark21)
     3. [DAC operation summary 26](#_bookmark26)
  3. [Direct Memory Access (DMA) 26](#_bookmark27)
     1. [DMA principles 27](#_bookmark28)
     2. [Bus mastering 27](#_bookmark29)
     3. [Transfer types 27](#_bookmark30)
     4. [DMA operation modes 28](#_bookmark31)
     5. [DMA module intergraded in STM32F407VG 29](#_bookmark32)
     6. [DMA functional description 31](#_bookmark33)
  4. [Serial Peripheral Interface/ Inter-IC Sound (SPI/I2S) 44](#_bookmark43)
     1. [SPI/I2S Introduction 44](#_bookmark44)
     2. [SPI and I2S main features 45](#_bookmark45)
     3. [SPI functional description 47](#_bookmark46)
     4. [I2S functional description 51](#_bookmark50)
  5. [I2C protocol 55](#_bookmark53)
     1. [Introduction 55](#_bookmark54)
     2. [I2C bus features 55](#_bookmark55)
     3. [I2C mode 56](#_bookmark57)
     4. [The I2C-bus protocol 57](#_bookmark58)
  6. [USB protocol 62](#_bookmark65)
     1. [USB Communication 63](#_bookmark67)
     2. [USB Advantages and Disadvantages 63](#_bookmark69)
  7. [WAV File Format 64](#_bookmark71)
     1. [Wave Audio File Format 64](#_bookmark72)
     2. [Wave file header 65](#_bookmark74)
     3. [Format Chunk 66](#_bookmark76)
     4. [Data chunk 66](#_bookmark77)

[CHAPTER III: HARDWARE AND DEVELOPMENT TOOLS 68](#_bookmark79)

* 1. [STM32F4 Discovery kit 69](#_bookmark80)
     1. [Features 69](#_bookmark81)
     2. [Hardware structure 69](#_bookmark82)
     3. [Embedded Debug Interface 70](#_bookmark85)
     4. [On-board audio capability 71](#_bookmark88)
  2. [Keil C uVison 5 IDE 72](#_bookmark89)
  3. [STM32 CubeMx 73](#_bookmark93)

[CHAPTER IV: SYSTEM DESIGN 75](#_bookmark97)

* 1. [Application overview 76](#_bookmark98)
  2. [Audio Playback Application 77](#_bookmark101)
  3. [Audio Recorder Application 77](#_bookmark103)
  4. [Volume control 78](#_bookmark105)

[CHAPTER V: CONCLUSION AND REFERENCE 80](#_bookmark108)

[Conclusion 80](#_bookmark109)

[Reference 81](#_bookmark110)

**Danh Mục Từ Viết Tắt**

|  |  |
| --- | --- |
| **Acronyms** | **Full word** |
| DAC | Digital to analog converter |
| I2C | Inter Intergraded Circuit |
| I2S | Inter-IC sound |
| SPI | Serial Peripherals Interface |
| USB | Universal Serial Bus |
| FAT | File Allocation Table |
| DMA | Direct Memory Access |
| IDE | Intergraded Development Environments |
| ACK | Acknowledgement |
| NACK | Non-Acknowledgement |
| SDA | Serial Data |
| SCL | Serial Clock |
| MOSI | Master Out Slave In |
| MISO | Master In Slave Out |
| CS | Chip Select |
| SD | Serial Data |
| WS | Word Select |

# Danh mục hình ảnh

## [Figure 1 DAC channel block diagram 24](#_bookmark20)

## [Figure 2 DAC control register bit map 25](#_bookmark22)

## [Figure 3 DAC channel 1 12-bit right-aligned data holding register 25](#_bookmark23)

## [Figure 4 DAC channel 1 data output register 25](#_bookmark24)

## [Figure 5 DAC status register 26](#_bookmark25)

## [Figure 6 DMA block diagram 31](#_bookmark34)

## [Figure 7 System implementation of the two DMA controllers on STM32F407 32](#_bookmark35)

## [Figure 8 DMA channel selection 33](#_bookmark36)

## [Figure 9 DMA1 request mapping 33](#_bookmark37)

## [Figure 10 DMA2 request mapping 34](#_bookmark38)

## [Figure 11 DMA direction bitmap 35](#_bookmark39)

## [Figure 12 Peripheral-to-memory mode 36](#_bookmark40)

## [Figure 13 Source and destination address registers in Double buffer mode 38](#_bookmark41)

## [Figure 14 DMA FIFO structure 40](#_bookmark42)

## [Figure 15 The block diagram of the SPI 47](#_bookmark47)

## [Figure 16 Single master/ single slave application 48](#_bookmark48)

## [Figure 17 Data clock timing diagram 50](#_bookmark49)

## [Figure 18 I2S block diagram 51](#_bookmark51)

## [Figure 19 I2S Philips protocol waveforms 53](#_bookmark52)

## [Figure 20 Example of I2C-bus applications 56](#_bookmark56)

## [Figure 21 Devices with various supply voltages sharing the same bus 57](#_bookmark59)

## [Figure 22 Bit transfer on the I2C-bus 58](#_bookmark60)

## [Figure 23 START and STOP conditions 58](#_bookmark61)

## [Figure 24 Data transfer on the I2C-bus 59](#_bookmark62)

## [Figure 25 A complete data transmission 60](#_bookmark63)

## [Figure 26 START byte procedure 61](#_bookmark64)

## [Figure 27 USB flash drive 62](#_bookmark66)

## [Figure 28 Logical Connections between USB Host Clients and USB Device](#_bookmark68)

## [Endpoints 63](#_bookmark68)

## [Figure 29 WAV File Format Layout 65](#_bookmark73)

## [Figure 30 Hardware block diagram 70](#_bookmark83)

## [Figure 31 STM32F4 DISCOVERY board 70](#_bookmark84)

## [Figure 32 STM32 ST Link interface 71](#_bookmark86)

## [Figure 33 Driver update for STM board 71](#_bookmark87)

## [Figure 34 Keil C pack installer 72](#_bookmark90)

## [Figure 35 Keil C IDE interface 73](#_bookmark91)

## [Figure 36 Keil C Debug Interface 73](#_bookmark92)

## [Figure 37 ST Cube Mx GUI 74](#_bookmark94)

## [Figure 38 Clock configuration with STM32 Cube Mx 74](#_bookmark95)

## [Figure 39 Project management with ST Cube Mx 74](#_bookmark96)

## [Figure 40 Schematic of Audio peripherals connection on STM32F4 discovery 76](#_bookmark99)

## [Figure 41 Audio playback and record architecture 76](#_bookmark100)

## [Figure 42 Audio playback application flow chart 77](#_bookmark102)

## [Figure 43 Audio recording flow chart 78](#_bookmark104)

## [Figure 44 Setup MCU pin for external trigger 78](#_bookmark106)

## [Figure 45 Volume control diagram 79](#_bookmark107)

**Danh Mục Bảng Biểu**

## [Table 1 FAT file system structure 20](#_bookmark17)

[**Table 2 Advantages and Disadvantages of USB 64**](#_bookmark70)

[**Table 3 Wave File Header 65**](#_bookmark75)

[**Table 4 Wav Data chunk 66**](#_bookmark78)

Embedded system is now can be found in every equipment in our life from simple device as refrigerator, microwave oven … to high technology entertainment devices as smart TV, smart phone or automotive system as engine control system, in-car entertainment system…The heart of these systems is microcontroller. In the role of an Embedded system developer, knowledges about microcontroller architecture and C coding skills are very necessary and important. In order to prepare the necessary knowledge for working as an embedded software developer, I decided to work on the project “Audio playback and recorder using STM32F4 Discovery”.

In this project, I will use development kit STM32F4 Discovery to record sound signal, store the data to an USB stick in the WAV file format and playback recorded signal via a speaker of a headphone. In order to finish this project, I will have to study about many peripherals such as USB to store data then transmit its to sound process chipset to output the sound, I2S to transmit the data to DAC module, DAC in charge of converting digital signal to analog signal, I2C to control the sound process chipset, DMA to allow us to transmit data directly from memory to I2S peripheral in order to reduce the precious clock cycles of CPU. I also have to learn how to associate all above peripherals, ensure every thing can work correctly to create one application.

# Chương I: Tóm Tắt Đồ Án

## Tóm tắt đồ án

Đồ án “Lập trình thiết bị thu âm và phát nhạc với STM32F4” là đề tài ứng dụng kiến thức lập trình và kiến trúc vi điều khiển để tạo ra một đề tài cụ thể.

Trong thời đại cách mạng khoa học và công nghiệp 4.0, cuộc sống con người càng ngày càng yêu cầu nhiều ứng dung như nhà thông minh( cửa tự động, hệ thống tưới nước, tự dộng điều chỉnh, điều khiển giọng nói, camera an ninh…), điện thoại thông minh, hệ thống tự động hóa công nghệ cao trong ô tô như cảnh báo vật cản, tự dộng đỗ xe… Tất cả các ứng dụng trên đều có chung một đặc điểm. Thành phần quan trọng nhất của các hệ thống này chính là các chip vi điều khiển.

Vi điều khiển là một hệ thống rất phức tạp nhưng cũng rất thú vị để tìm hiểu. Với chi phí thấp, năng lượng tiêu thụ thấp cùng những tính năng đa dạng, mạnh mẽ, vi điều khiển xuất hiện trong mọi thiết bị điện tử. Nó cho phép chúng ta thực thi rất nhiều tính năng như thu thập dữ liệu tương tự từ cảm biến, chuyển đổi ra dạng số để xử lý hoặc lưu trữ, điều khiển tốc độ, chiều quay động cơ với bộ điều chế xung, truyền nhận dữ liệu với các giao thức truyền thông như I2C, SPI, UART, tạo trễ và trigger với timer, các chức năng mạng với CAN hoặc Ethenet…

Nhận thức được tầm quan trọng của vi điều khiển trong mỗi hệ thống nhúng, em thực hiện đồ án này nhằm mục tiêu nâng cao kiến thức về kiến trúc vi điều khiển và kỹ năng lập trình C.

## Mục đích nghiên cứu

Như phân tích bên trên, vi điều khiển đóng vai trò rất quan trọng trong hệ thống nhúng nên kiến thức về vi điều khiển là bắt buộc với bất kỳ nhà phát triển phần mềm nhúng nào. Trong những ứng dụng sử dụng rất nhiều ngoại vi, lập trình viên có thể có được những kiến thức rất quý báu về cả vi điều khiển và kỹ năng lập trình.

Có rất nhiều loại vi điều khiển , nhiều kiến trúc. Dựa trên độ dài thanh ghi, ta có vi điều khiển 8 bit, 16 bit, 32 bit. Dựa trên kiến trúc core, chúng ta có vi điều khiển ARM, MIPS, PowerPC.

Trong số các kiên trúc trên , kiến trúc ARM 32 bit là phổ biến nhất hiện nay vì tiết kiệm năng lượng( thích hợp ứng dụng IoT hay thiết bị cầm tay), giá rẻ và khả năng đáp ứng hầu hết các yêu cầu của hệ thống IoT, tự dộng hóa, công nghiệp…

Trong đồ án này, tôi thực thi một số chức năng được liệt kê dưới đây:

* Đọc và ghi dữ liệu định dạng WAV vào thẻ nhớ SD với định dạng FAT32 thông qua USB OTG
* Thu thập dữ liệu tương tự (tín hiệu âm thanh) thông qua mic thu âm.
* Chuyển đổi tín hiệu từ tương tự sang số để chuyển đổi.
* Chuyển đổi tín hiệu từ số ra tương tự để phát nhạc qua loa.
* Sử dụng DMA để nâng cao tốc độ truyền dữ liệu.

## Phương pháp nghiên cứu

Để thực thi các tính năng trên, tôi xác định các yêu cầu của hệ thống như sau:

* + - Đọc tài liệu thiết kế vi điều khiển để hiểu cấu tạo phần cứng của các ngoại vi cần sử dụng.
    - Đọc các tài liệu trên mạng để hiểu các giao thức truyền thông như I2C, I2S…
    - Đọc datasheet của các cảm biến để biết cấu tạo, hoạt động và cách chúng được kết nối đến vi điều khiển.
    - Lập trình, nạp code xuống kit phát triển, debug và kiểm thử.

## Conclusion

Trong chương này, tôi đã trình bày khái quá đề tài đồ án, lý do tại sao tôi thực hiện đồ án. Trong chương tiếp theo, tôi sẽ trình bày về lý thuyết, cấu túc phần cứng và cách hoạt động của các ngoại vi trên vi điều khiển.

# Chương II: Lý Thuyết Tổng Hợp

Trong chương này, tôi sẽ trình bày các nội dung sau:

* FAT32 File system
* Bộ chuyển đổi số tương tự (DAC)
* Phương pháp truy xuất bộ nhớ trực tiếp (DMA)\
* Giao thức truyền thông I2C, SPI/I2S
* Giao thức USB
* Định dạng file WAV

## FAT File system

## Lịch sử FAT File system

FAT (File Allocation Table) là kiến trúc hệ thống file và thư mục và họ các quy chuẩn về file để sử dụng nó.

Hệ thống tệp FAT là một hệ thống tệp kế thừa đơn giản và mạnh mẽ. Nó cung cấp hiệu suất tốt với các hệ thống cần lưu trữ tệp nhẹ, đơn giản, nhưng không thể cung cấp hiệu suất, độ tin cậy và khả năng mở rộng tương tự như một số hệ thống tệp hiện đại. Tuy nhiên, nó được hỗ trợ vì lý do tương thích bởi gần như tất cả hệ điều hành hiện đang phát triển cho máy tính cá nhân và nhiều máy tính gia đình, thiết bị di động và hệ thống nhúng, và do đó là định dạng tương thích để trao đổi dữ liệu giữa các máy tính và thiết bị ở hầu hết mọi loại thiết bị phát triển từ năm 1981 đến hiện tại.

Được thiết kế ban đầu vào năm 1977 để sử dụng trên các đĩa mềm, FAT đã sớm được điều chỉnh và sử dụng gần như phổ biến trên các đĩa cứng trong suốt thời đại của hệ điều hành DOS và Windows 9x trong hai thập kỷ. Ngày nay, các hệ thống tệp FAT vẫn thường được tìm thấy trên các đĩa mềm, thẻ nhớ USB, flash và các thẻ nhớ và mô-đun trạng thái rắn khác, cùng nhiều thiết bị di động và nhúng.

## Tổng quan về hệ thống tệp FAT

Tên của hệ thống tệp bắt nguồn từ việc sử dụng nổi bật của hệ thống tệp của bảng chỉ mục, Bảng phân bổ tệp, được phân bổ tĩnh tại thời điểm định dạng. Bảng chứa các mục nhập cho mỗi cụm, một vùng lưu trữ đĩa liền kề. Mỗi mục chứa số cụm tiếp theo trong tệp hoặc một dấu hiệu khác chỉ ra phần cuối của tệp, không gian đĩa không sử dụng hoặc các khu vực dành riêng đặc biệt của đĩa. Thư mục gốc của đĩa chứa số cụm đầu tiên của mỗi tệp trong thư mục đó; hệ điều hành sau đó có thể duyệt qua bảng FAT, tìm kiếm số cụm của từng phần liên tiếp của tệp đĩa dưới dạng chuỗi cụm cho đến khi kết thúc tệp. Theo cách tương tự, các thư mục con được triển khai dưới dạng các tệp.

Được thiết kế ban đầu như một hệ thống tệp 8 bit, số lượng cụm tối đa đã được tăng lên đáng kể khi các ổ đĩa đã phát triển, và do đó số lượng bit được sử dụng để xác định từng cụm đã tăng lên. Các phiên bản chính liên tiếp của định dạng FAT được đặt tên theo số bit phần tử bảng: 12 (FAT12), 16 (FAT16) và 32 (FAT32). Ngoại trừ tiền thân FAT 8 bit ban đầu, mỗi biến thể này vẫn đang được sử dụng. Tiêu chuẩn FAT cũng đã được mở rộng theo những cách khác nhưng nói chung vẫn duy trì khả năng tương thích với phần mềm hiện có.

## Cấu trúc phân vùng hệ thống tệp FAT

Tổng quan về tổ chức phân vùng trong hệ thống tệp FAT

*Bảng 1 Cấu trúc hệ thống tệp FAT*

|  |  |  |
| --- | --- | --- |
| **Phân vùng** | **Kích thước(sector)** | **Nội dung** |
| Sectors dự phòng | number of reserved sectors | Boot Sector |
| Sector về thông tin hệ thống tệp (chỉ FAT32) |
| sectors dự phòng khác(tùy chọn) |
| Vùng  FAT | (Số bảng FATs) \* (số sectors trên một bảng FAT) | Bảng phân bố file #1 |
| Bảng phân bố file #2 ... (tùy chọn) |
| Phân vùng thư mục gốc | (Số thư mục gốc  \* 32) / (số bytes trên sector) | Thư mục gốc (chỉ FAT12 và FAT16) |
| Phân vùng dữ liệu | (số cụm)\* (số sectors trên cụm) | Phân vùng dữ liệu (cho file hay thư mục) ... (cho đến khi kết thúc đĩa cứng) |

## Sectors dự phòng

Phân vùng này chứa một số thông tin hệ thống tệp cơ bản, đặc biệt là loại và con trỏ tới vị trí của các phần khác và thường chứa mã bộ tải khởi động của hệ điều hành.

Riêng với FAT32, phân vùng này còn chưa thêm sector về thông tin hệ thống file ở vị trí sector 1 và một boot sector dự phòng ở vị trí sector thứ

## Phân vùng FAT

Đây là phân vùng chứa 2 bảng FAT, một bản chính, một bản dự phòng dù rất ít khi dùng, ngay cả với các tiện ích sửa chữa đĩa. Hai bảng FAT này là bản đồ của cả ổ cứng hay vùng nhớ. Nó chỉ ra cụm nào được sử dụng bởi file hay thư mục nào, từ đó hệ điều hành có thể tìm tiếm và thực hiện các tác vụ với file hay thư mục.

## Phân vùng thư mục gốc

Đây là Bảng thư mục lưu trữ thông tin về các tệp và thư mục nằm trong thư mục gốc. Nó chỉ được sử dụng với FAT12 và FAT16 và áp đặt cho thư mục gốc một kích thước tối đa cố định được phân bổ trước khi tạo tập này. FAT32 lưu trữ thư mục gốc trong Vùng dữ liệu, cùng với các tệp và các thư mục khác, cho phép nó phát triển mà không bị ràng buộc như vậy. Do đó, đối với FAT32, Vùng dữ liệu bắt đầu tại đây.

## Vùng dữ liệu

Đây là nơi lưu trữ dữ liệu tệp và thư mục thực tế và chiếm phần lớn phân vùng.

Kích thước của tệp và thư mục con có thể được tăng tùy ý (miễn là có các cụm miễn phí) bằng cách thêm nhiều liên kết vào chuỗi của tệp trong FAT. Tuy nhiên, lưu ý rằng các tệp được phân bổ theo đơn vị cụm, vì vậy nếu tệp 1 KB nằm trong cụm 32 KB, 31 KB bị lãng phí.

FAT32 thường có bảng thư mục gốc bắt đầu trong cụm thứ 2: cụm đầu tiên của phân vùng dữ liệu.

Fat sử dụng định dạng little edian cho các entry trong phần header và bảng FAT. Nó có thể cấp phát nhiều FAT sector hơn cho một số lượng cluster.

## Bộ chuyển đổi số tương tự DAC

Trong điện tử, bộ chuyển đổi tín hiệu số sang tương tự (DAC, D / A, D2A hoặc D-to-A) là một hệ thống chuyển đổi tín hiệu số thành tín hiệu tương tự. Bộ chuyển đổi tương tự sang số (ADC) thực hiện chức năng đảo ngược.

Có một số kiến ​​trúc DAC; sự phù hợp của một bộ xử lý tín hiệu cho một ứng dụng cụ thể được xác định bằng các tham số bao gồm: độ phân giải, tần suất lấy mẫu tối đa và các thứ khác. Chuyển đổi từ kỹ thuật số sang tương tự có thể làm giảm tín hiệu, do đó, một bộ định tuyến phải được chỉ định có lỗi không đáng kể về mặt ứng dụng.

Các bộ xử lý tín hiệu thường được sử dụng trong các máy nghe nhạc để chuyển đổi luồng dữ liệu số thành tín hiệu âm thanh analog. Chúng cũng được sử dụng trong TV và điện thoại di động để chuyển đổi dữ liệu video kỹ thuật số thành tín hiệu video tương tự kết nối với trình điều khiển màn hình để hiển thị màu đơn sắc hoặc ảnh màu.

Hai ứng dụng này sử dụng các bộ xử lý tín hiệu ở hai đầu đối diện của sự hy sinh tần số hoặc độ phân giải. Bộ xử lý âm thanh là loại tần số thấp, độ phân giải cao trong khi bộ xử lý video là loại có độ phân giải thấp đến trung bình.

Do sự phức tạp và nhu cầu về các thành phần được kết hợp chính xác, tất cả trừ các bộ vi xử lý chuyên dụng nhất được triển khai dưới dạng các mạch tích hợp (IC). Các bộ xử lý rời rạc thường có tốc độ cực cao, độ phân giải thấp, các loại ngốn điện, như được sử dụng trong các hệ thống radar quân sự. Thiết bị thử nghiệm tốc độ rất cao, đặc biệt là máy hiện sóng lấy mẫu, cũng có thể sử dụng các bộ vi xử lý rời rạc.

Một bộ chuyển đổi chuyển đổi một số chính xác thành một đại lượng vật lý (ví dụ: điện áp hoặc áp suất). Cụ thể, các bộ xử lý tín hiệu thường được sử dụng để chuyển đổi dữ liệu chuỗi thời gian có độ chính xác hữu hạn thành tín hiệu vật lý liên tục thay đổi.

Một bộ xử lý lý tưởng chuyển đổi các số trừu tượng thành một chuỗi các xung sau đó được xử lý bởi bộ lọc và được tái tạo bằng cách sử dụng một số dạng nội suy để điền dữ liệu giữa các xung. Một bộ xử lý thực tế thông thường chuyển đổi các số thành một hàm hằng số được tạo thành từ một chuỗi các hàm hình chữ nhật được mô hình hóa với việc giữ không thứ tự. Các phương pháp DAC khác (chẳng hạn như các phương pháp dựa trên điều chế delta-sigma) tạo ra đầu ra điều chế mật độ xung có thể được lọc tương tự để tạo ra tín hiệu thay đổi trơn tru.

Theo lý thuyết lấy mẫu Nyquist-Shannon, một bộ DAC có thể tái tạo tín hiệu gốc từ dữ liệu được lấy mẫu với điều kiện băng thông của nó đáp ứng một số yêu cầu nhất định (ví dụ: tín hiệu băng cơ sở có băng thông nhỏ hơn một nửa tần số Nyquist). Lấy mẫu kỹ thuật số đưa ra lỗi lượng tử hóa biểu hiện là nhiễu mức thấp trong tín hiệu được tái tạo.

## Bộ chuyển đổi số tương tự trên vi điều khiển STM32F407VG

Mô-đun DAC là bộ chuyển đổi kỹ thuật số sang tương tự 12 bit, điện áp đầu ra. Bộ xử lý có thể được cấu hình ở chế độ 8- hoặc 12 bit và có thể được sử dụng cùng với bộ điều khiển DMA. Ở chế độ 12 bit, dữ liệu có thể được căn trái hoặc phải. Bộ xử lý tín hiệu có hai kênh đầu ra, mỗi kênh có bộ chuyển đổi riêng. Trong chế độ kênh DAC kép, chuyển đổi có thể được thực hiện độc lập hoặc đồng thời khi cả hai kênh được nhóm lại với nhau cho các hoạt động cập nhật đồng bộ. Một pin tham chiếu đầu vào, VREF + (được chia sẻ với ADC) có sẵn để làm cơ sở chuyển đổi mức tín hiệu.

Các tính năng chính của DAC:

• Hai bộ chuyển đổi DAC: mỗi kênh một đầu ra

• Căn chỉnh dữ liệu trái hoặc phải trong chế độ 12 bit

• Kh năng cập nhật được đồng bộ hóa.

• Tạo sóng nhiễu

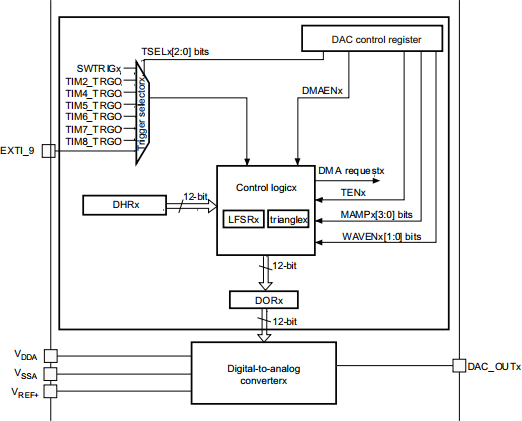
• Tạo sóng tam giác

• Kênh DAC kép để chuyển đổi độc lập hoặc đồng thời

• Khả năng truy cập bộ nhớ trực tiếp cho từng kênh

• Kích hoạt bên ngoài để chuyển đổi

• Tham chiếu điện áp đầu vào, VREF +



*Ảnh 1 Sơ đồ khối*

## Các bước khởi tạo và sử dụng DAC

Để chuyển đổi dữ liệu số sang dữ liệu tương tự với mô-đun DAC:

• Khởi tạo đầu vào, đầu ra, đất, điện áp tham chiếu cho DAC

• Chọn nguồn kích hoạt cho mô-đun: timer nội, pin bên ngoài hoặc trigger phần mềm

• Định cấu hình các tham số (yêu cầu DMA, tạo xung tam giác, bật ngắt DMA, biên độ dao động) để xác định chế độ hoạt động của mô-đun DAC.

• Kích hoạt ISR kênh DMA để quản lý hoạt động của mô-đun DAC qua ngắt

## Cơ chế truy cập bộ nhớ trực tiếp (DMA)

Truy cập bộ nhớ trực tiếp (DMA) là một tính năng của các hệ thống máy tính cho phép một số hệ thống con phần cứng nhất định truy cập bộ nhớ hệ thống chính (bộ nhớ truy cập ngẫu nhiên), độc lập với bộ xử lý trung tâm (CPU).

Không có DMA, khi CPU đang sử dụng đầu vào / đầu ra được lập trình, nó thường bị chiếm dụng hoàn toàn trong toàn bộ thời gian của hoạt động đọc hoặc ghi và do đó không thẻ để thực hiện công việc khác. Với DMA, CPU trước tiên bắt đầu truyền, sau đó nó thực hiện các hoạt động khác trong khi quá trình truyền đang diễn ra và cuối cùng nó nhận được một ngắt từ bộ điều khiển DMA (DMAC) khi hoạt động được thực hiện. Tính năng này hữu ích bất cứ lúc nào CPU không theo kịp tốc độ truyền dữ liệu hoặc khi CPU cần thực hiện công việc trong khi chờ truyền dữ liệu I / O tương đối chậm. Nhiều hệ thống phần cứng sử dụng DMA, bao gồm bộ điều khiển ổ đĩa, card đồ họa, card mạng và card âm thanh. DMA cũng được sử dụng để truyền dữ liệu trong chip xử lý đa lõi. Máy tính có kênh DMA có thể truyền dữ liệu đến và từ các thiết bị mà ít sử dụng CPU hơn so với máy tính không có kênh DMA. Tương tự, một phần tử xử lý bên trong bộ xử lý đa lõi có thể truyền dữ liệu đến và từ bộ nhớ cục bộ của nó mà không cần dùng tới những xung đồng hồ quý giá của CPU. Điều này cho phép việc tính toán và đọc ghi dữ liệu được thức hiện song song.

DMA cũng có thể được sử dụng để sao chép "bộ nhớ vào bộ nhớ" hoặc di chuyển dữ liệu trong bộ nhớ.

## DMA principles

DMA tiêu chuẩn, còn được gọi là DMA của bên thứ ba, sử dụng bộ điều khiển DMA. Bộ điều khiển DMA có thể tạo địa chỉ bộ nhớ và khởi tạo chu kỳ đọc hoặc ghi bộ nhớ. Nó chứa một số thanh ghi phần cứng có thể được ghi và đọc bởi CPU. Chúng bao gồm một thanh ghi địa chỉ bộ nhớ, một thanh ghi đếm byte và một hoặc nhiều thanh ghi điều khiển. Tùy thuộc vào tính năng nào mà bộ điều khiển DMA cung cấp, các thanh ghi điều khiển này có thể chỉ định một số kết hợp nguồn, đích, hướng chuyển (đọc từ thiết bị I / O hoặc ghi vào thiết bị I / O), kích thước của đơn vị truyền và số byte cần truyền trong một cụm.

Để thực hiện thao tác nhập, xuất hoặc bộ nhớ vào bộ nhớ, bộ xử lý CPU khởi tạo bộ điều khiển DMA với số lượng từ cần truyền và địa chỉ bộ nhớ sẽ sử dụng. CPU sau đó ra lệnh cho thiết bị ngoại vi để bắt đầu truyền dữ liệu. Bộ điều khiển DMA sau đó cung cấp địa chỉ và các dòng điều khiển đọc / ghi vào bộ nhớ hệ thống. Mỗi khi một byte dữ liệu sẵn sàng được chuyển giữa thiết bị ngoại vi và bộ nhớ, bộ điều khiển DMA sẽ tăng thanh ghi địa chỉ bên trong cho đến khi toàn bộ khối dữ liệu được truyền hết.

## Vấn đề quản lý đường Bus

Trong hệ thống DMA của bên thứ nhất, mỗi CPU và các thiết bị ngoại vi có thể được cấp quyền điều khiển bus bộ nhớ. Khi một thiết bị ngoại vi có thể trở thành bus chính, nó có thể ghi trực tiếp vào bộ nhớ hệ thống mà không cần sự tham gia của CPU, cung cấp địa chỉ bộ nhớ và tín hiệu điều khiển theo yêu cầu. Một số biện pháp phải được cung cấp để đưa bộ xử lý tranh chấp quyền sử dụng bus với ngoại vi.

## Các kiểu chuyền dữ liệu

Việc truyền dữ liệu qua DMA có thể chuyển từng byte một lần hoặc tất cả cùng một lúc trong chế độ truyền liên tục. Nếu truyền một byte tại một thời điểm, CPU có thể truy cập bộ nhớ vào các chu kỳ bus xen kẽ.

Trong chế độ truyền liên tục, DMA giữ quyền kiểm soát đường bus và điều khiển bộ nhớ để truyền đi hang tram hay hang triệu byte một cách liên tCPU sẽ không thể sử dụng các tài nguyên này trong thời gian đó.

2.3.4 Các chế độ hoạt động của DMA

Chế đọ truyền liên tục:

Trong chế độ chụp liên tục, toàn bộ khối dữ liệu được truyền theo một chuỗi liền kề. Khi bộ điều khiển DMA được CPU cấp quyền truy cập vào bus hệ thống, nó sẽ chuyển tất cả byte dữ liệu trong khối dữ liệu trước khi giải phóng điều khiển của bus hệ thống trở lại CPU, nhưng khiến CPU không hoạt động trong thời gian tương đối dài. Chế độ này còn được gọi là "Chế độ chuyển khối".

Chế độ chuyền từng byte:

Chế độ truyền từng byte được sử dụng trong các hệ thống không nên tắt CPU trong khoảng thời gian cần thiết cho các chế độ truyền phát. Trong chế độ truyền từng byte, bộ điều khiển DMA có được quyền truy cập vào bus hệ thống giống như ở chế độ truyền liên tục, sử dụng tín hiệu BR (Bus Request) và BG (Bus Grant), là hai tín hiệu điều khiển giao diện giữa CPU và Bộ điều khiển DMA. Tuy nhiên, trong chế độ truyền từng byte, sau một byte truyền dữ liệu, việc điều khiển bus hệ thống được xác nhận lại cho CPU thông qua BG. Sau đó, nó liên tục được yêu cầu lại thông qua BR, truyền một byte dữ liệu cho mỗi yêu cầu, cho đến khi toàn bộ khối dữ liệu được truyền. Bằng cách liên tục có được và giải phóng điều khiển bus hệ thống, bộ điều khiển DMA về cơ bản xen kẽ các lệnh truyền và truyền dữ liệu. CPU xử lý một lệnh, sau đó bộ điều khiển DMA truyền một giá trị dữ liệu, v.v. Một mặt, khối dữ liệu không được truyền nhanh trong chế độ truyền từng byte theo chu kỳ như ở chế độ truyền liên tục, nhưng mặt khác, CPU không được nhàn rỗi lâu như ở chế độ truyền liên tục. Chế độ truyền từng byte rất hữu ích cho các bộ điều khiển theo dõi dữ liệu trong thời gian thực.

## Chế độ hỗn hợp:

Chế độ hỗn hợp mất nhiều thời gian nhất để truyền một khối dữ liệu, tuy nhiên nó cũng là chế độ hiệu quả nhất về mặt hiệu suất hệ thống tổng thể. Ở chế độ hỗn hợp, bộ điều khiển DMA chỉ truyền dữ liệu khi CPU đang thực hiện các hoạt động không sử dụng các bus hệ thống. Ưu điểm chính của chế độ hỗn hợp là CPU không bao giờ ngừng thực thi các chương trình của nó và việc chuyển DMA là miễn phí về thời gian, trong khi nhược điểm là phần cứng cần xác định khi CPU không sử dụng các bus hệ thốn. Điều này đôi khi rất phức tạp.

**DMA mo-dun tích hợp trên STM32F407VG**

Các tính năng chính :

• Kiến trúc bus chính AHB kép, một dành riêng cho truy cập bộ nhớ và một dành riêng cho truy cập ngoại vi

• Giao diện lập trình AHB chỉ hỗ trợ truy cập 32 bit

• 8 luồng cho mỗi bộ điều khiển DMA, tối đa 8 kênh (yêu cầu) trên mỗi luồng

* + Mỗi luồn có thể được cài đặt để:

- một kênh thông thường hỗ trợ chuyển từ bộ nhớ ngoại vi sang bộ nhớ, bộ nhớ đến ngoại vi và bộ nhớ sang bộ nhớ

- kênh song song cũng hỗ trợ bộ đệm song song ở phía bộ nhớ

• Mỗi trong số 8 luồng được kết nối với các kênh (yêu cầu) phần cứng chuyên dụng

• Ưu tiên giữa các yêu cầu luồng DMA là lập trình phần mềm (4 cấp độ bao gồm rất cao, cao, trung bình, thấp) hoặc phần cứng trong trường hợp bình đẳng (yêu cầu 0 được ưu tiên hơn yêu cầu 1, v.v.),

• Mỗi luồng cũng hỗ trợ kích hoạt phần mềm để chuyển từ bộ nhớ sang bộ nhớ (chỉ khả dụng cho bộ điều khiển DMA2)

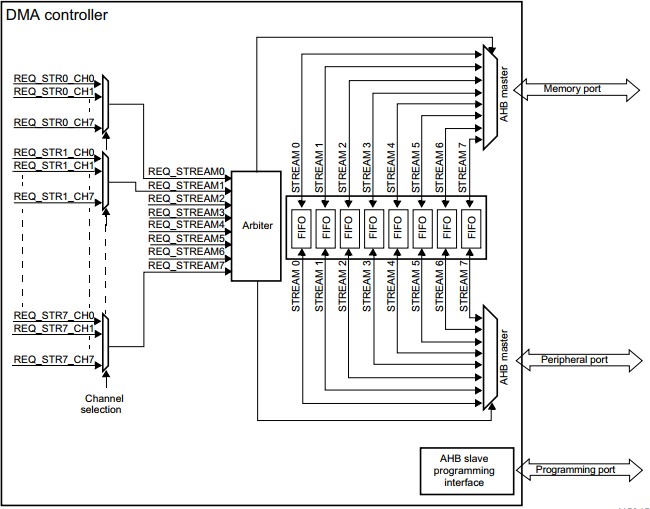
• Mỗi yêu cầu luồng có thể được chọn trong số tối đa 8 yêu cầu kênh có thể. Lựa chọn này có thể cấu hình bằng phần mềm và cho phép một số thiết bị ngoại vi khởi tạo DMA yêu cầu

• Độ rộng truyền nguồn và đích độc lập (byte, nửa từ, từ): khi độ rộng dữ liệu của nguồn và đích không bằng nhau, DMA sẽ tự động đóng gói / giải nén các dữ liệu cần thiết để tối ưu hóa băng thông. Tính năng này chỉ khả dụng trong chế độ FIFO

• Tăng địa chỉ hoặc không tăng địa chỉ cho nguồn và đích

• 5 cờ sự kiện (Chuyển một nửa DMA, Hoàn thành chuyển DMA, Lỗi chuyển DMA, Lỗi DMA FIFO, Lỗi chế độ trực tiếp) hợp lý với nhau trong một yêu cầu ngắt cho mỗi luồng

## Chi tiết chức năng của DMA



*Ảnh 6 Sơ đồ khối DMA*

Bộ điều khiển DMA thực hiện chuyển bộ nhớ trực tiếp: khi có quyền khiểm soát bus AHB, nó có thể điều khiển ma trận bus AHB để bắt đầu các hoạt động truyền dữ liệu sử dụng bus AHB.

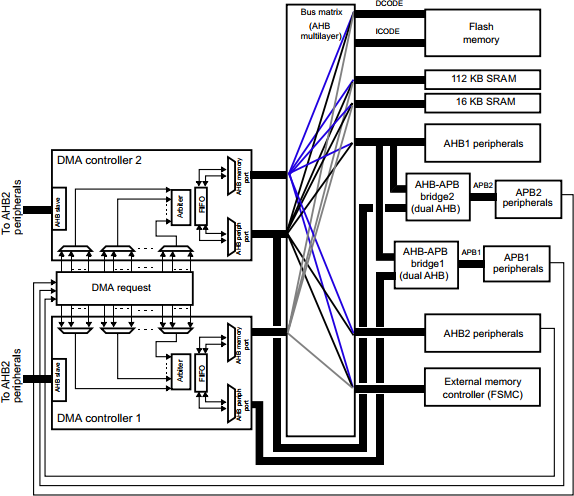
Nó có thể thực hiện các hướng chuyển dữ liệu sau:

• bộ nhớ ngoại vi

• bộ nhớ ngoại vi

• bộ nhớ đến bộ nhớ

Bộ điều khiển DMA cung cấp hai cổng chính AHB: cổng bộ nhớ AHB, dự định được kết nối với bộ nhớ và cổng ngoại vi AHB, dự định được kết nối với các thiết bị ngoại vi. Tuy nhiên, để cho phép chuyển bộ nhớ sang bộ nhớ, cổng ngoại vi AHB cũng phải có quyền truy cập vào bộ nhớ.



*Ảnh 7 Cấu tạo phần cứng trên vi điều khiển STM32F407*

## Quá trình truyền dữ liệu với DMA

Việc truyền dữ liệu qua DMA bao gồm một chuỗi các lần chuyển dữ liệu nhất định. Số lượng các mục dữ liệu được truyền và chiều rộng của chúng (8 bit, 16 bit hoặc 32 bit) được lập trình phần mềm. Mỗi lần chuyển DMA bao gồm ba thao tác:

• Tải từ thanh ghi dữ liệu ngoại vi hoặc vị trí trong bộ nhớ

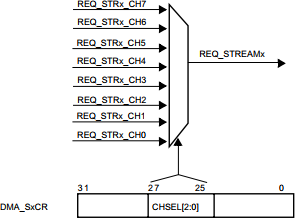
• Lưu trữ dữ liệu được tải vào thanh ghi dữ liệu ngoại vi.

• Giảm số lượng phần tử cần truyền.

Sau một sự kiện, thiết bị ngoại vi sẽ gửi tín hiệu yêu cầu đến bộ điều khiển DMA. Bộ điều khiển DMA phục vụ yêu cầu tùy thuộc vào các ưu tiên của kênh. Ngay khi bộ điều khiển DMA truy cập vào thiết bị ngoại vi, tín hiệu ACK được gửi đến thiết bị ngoại vi bởi bộ điều khiển DMA. Thiết bị ngoại vi giải phóng yêu cầu của nó ngay khi nhận được tín hiệu ACK từ bộ điều khiển DMA. Khi yêu cầu đã được xác nhận lại bởi thiết bị ngoại vi, bộ điều khiển DMA sẽ phát ra tín hiệu ACK.

Lựa chọn kênh truyền:

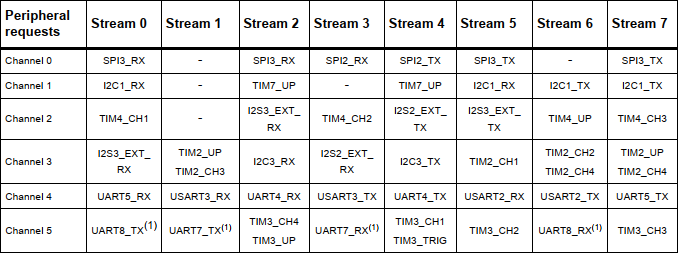
Mỗi luồng được liên kết với một yêu cầu DMA có thể được chọn trong số 8 yêu cầu kênh có thể.



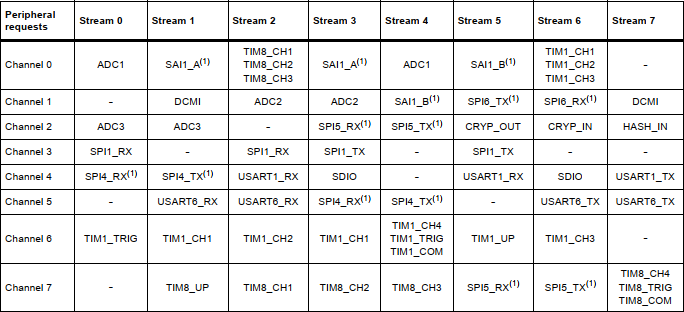
*Ảnh 8 Cơ chế lựa chọn kênh truyền DMA*

8 yêu cầu từ các thiết bị ngoại vi (TIM, ADC, SPI, I2C, v.v.) được kết nối độc lập với từng kênh.

Hai hình dưới đây cho thấy kết nối giữa các kênh DMA với các thiết bị ngoại vi



*Ảnh 9 Kết nối kênh và ngoại vi trên DMA1*



*Ảnh 10 Kết nối kênh và ngoại vi trên DMA2*

## Arbiter

An arbiter manages the 8 DMA stream requests based on their priority for each of the two AHB master ports (memory and peripheral ports) and launches the peripheral/memory access sequences.

Priorities are managed in two stages:

* + Software: each stream priority can be configured in the DMA control register. There are four levels:
* Very high priority
* High priority
* Medium priority
* Low priority
  + Hardware: If two requests have the same software priority level, the stream with the lower number takes priority over the stream with the higher number. For example, Stream 2 takes priority over Stream 4.

## DMA streams

Each of the 8 DMA controller streams provides a unidirectional transfer link between a source and a destination.

Each stream can be configured to perform:

* + Regular type transactions: memory-to-peripherals, peripherals-to-memory or memory-to- memory transfers
  + Double-buffer type transactions: double buffer transfers using two memory pointers for the memory (while the DMA is reading/writing from/to a buffer, the application can write/read to/from the other buffer).

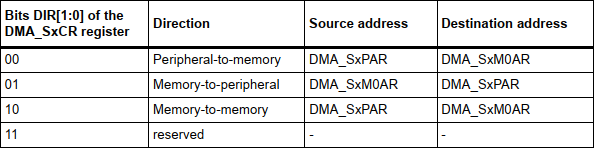
The amount of data to be transferred (up to 65535) is programmable and related to the source width of the peripheral that requests the DMA transfer connected to the peripheral AHB port. The register that contains the amount of data items to be transferred is decremented after each transaction.

## Source, destination and transfer modes

Both source and destination transfers can address peripherals and memories in the entire 4 GB area, at addresses comprised between 0x0000 0000 and 0xFFFF FFFF.

The direction is configured using specific register and offers three possibilities: memory-to- peripheral, peripheral-to-memory or memory-to-memory transfers.

This figure below describes the corresponding source and destination addresses



*Figure 11 DMA direction bitmap*

## Source and destination address

When the data width is a half-word or a word, respectively, the peripheral or memory address written into registers has to be aligned on a word or half-word address boundary, respectively.

In this application, I only use peripheral-to-memory mode so I will explain this DMA mode in details.

## Peripheral-to-memory mode

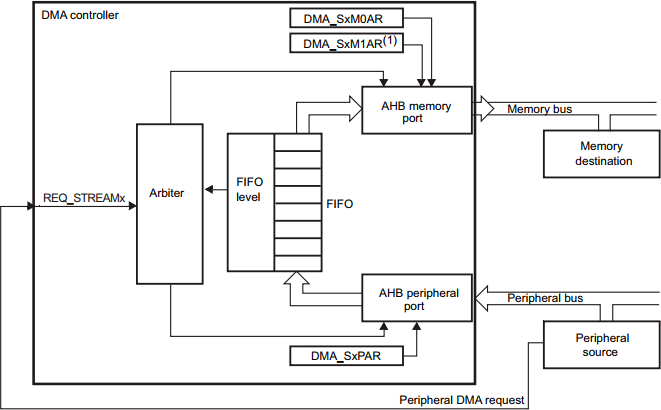
When this mode is enabled, each time a peripheral request occurs, the stream initiates a transfer from the source to fill the FIFO.

When the threshold level of the FIFO is reached, the contents of the FIFO are drained and stored into the destination.

The transfer stops once the DMA counter register reaches zero, when the peripheral requests the end of transfers (in case of a peripheral flow controller) or when the EN bit in the DMA control register is cleared by software.

In direct mode, the threshold level of the FIFO is not used: after each single data transfer from the peripheral to the FIFO, the corresponding data are immediately drained and stored into the destination.

The stream has access to the AHB source or destination port only if the arbitration of the corresponding stream is won. This arbitration is performed using the priority defined for each stream using the DMA control register



*Figure 12 Peripheral-to-memory mode*

## Pointer incrementation

Peripheral and memory pointers can optionally be automatically post-incremented or kept constant after each transfer depending on the PINC and MINC bits in the DMA control register.

Disabling the Increment mode is useful when the peripheral source or destination data are accessed through a single register.

If the Increment mode is enabled, the address of the next transfer will be the address of the previous one incremented by 1 (for bytes), 2 (for half-words) or 4 (for words) depending on the data width programmed in the PSIZE or MSIZE bits in the DMA control register.

In order to optimize the packing operation, it is possible to fix the increment offset size for the peripheral address whatever the size of the data transferred on the AHB peripheral port.

The PINCOS bit in the DMA control register is used to align the increment offset size with the data size on the peripheral AHB port, or on a 32-bit address (the address is then incremented by 4). The PINCOS bit has an impact on the AHB peripheral port only.

If PINCOS bit is set, the address of the next transfer is the address of the previous one incremented by 4 (automatically aligned on a 32-bit address) whatever the PSIZE value.

The AHB memory port, however, is not impacted by this operation.

## Circular mode

The Circular mode is available to handle circular buffers and continuous data flows (e.g. ADC scan mode). This feature can be enabled using the CIRC bit in the DMA control register.

When the circular mode is activated, the number of data items to be transferred is automatically reloaded with the initial value programmed during the stream configuration phase, and the DMA requests continue to be served.

## Double buffer mode

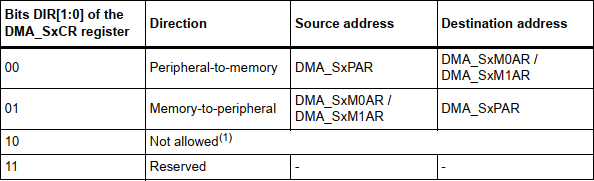
This mode is available for all the DMA1 and DMA2 streams.

The Double buffer mode is enabled by setting the DBM bit in the DMA control register.

A double-buffer stream works as a regular (single buffer) stream with the difference that it has two memory pointers. When the Double buffer mode is enabled, the Circular mode is automatically enabled and at each end of transaction, the memory pointers are swapped.

In this mode, the DMA controller swaps from one memory target to another at each end of transaction. This allows the software to process one memory area while the second memory area is being filled/used by the DMA transfer. The double-buffer stream can work in both

directions (the memory can be either the source or the destination) as described in the figure below



*Figure 13 Source and destination address registers in Double buffer mode*

## Single and burst transfers

The DMA controller can generate single transfers or incremental burst transfers of 4, 8 or 16 beats.

The size of the burst is configured by software independently for the two AHB ports by using DMA control register.

The burst size indicates the number of beats in the burst, not the number of bytes transferred.

To ensure data coherence, each group of transfers that form a burst are indivisible: AHB transfers are locked and the arbiter of the AHB bus matrix does not degrant the DMA master during the sequence of the burst transfer.

Depending on the single or burst configuration, each DMA request initiates a different number of transfers on the AHB peripheral port:

* + When the AHB peripheral port is configured for single transfers, each DMA request generates a data transfer of a byte, half-word or word depending on the peripheral size bits in the DMA control register
  + When the AHB peripheral port is configured for burst transfers, each DMA request generates 4,8 or 16 beats of byte, half word or word transfers depending on the configuration about the peripheral burst and peripheral size in the DMA control register.

The same as above has to be considered for the AHB memory port considering the MBURST and MSIZE bits.

In direct mode, the stream can only generate single transfers and the memory burst and peripherals burst are forced by hardware.

The address pointers must be chosen so as to ensure that all transfers within a burst block are aligned on the address boundary equal to the size of the transfer.

The burst configuration has to be selected in order to respect the AHB protocol, where bursts must not cross the 1 KB address boundary because the minimum address space that can be allocated to a single slave is 1 KB. This means that the 1 KB address boundary should not be crossed by a burst block transfer, otherwise an AHB error would be generated, that is not reported by the DMA registers.

## FIFO

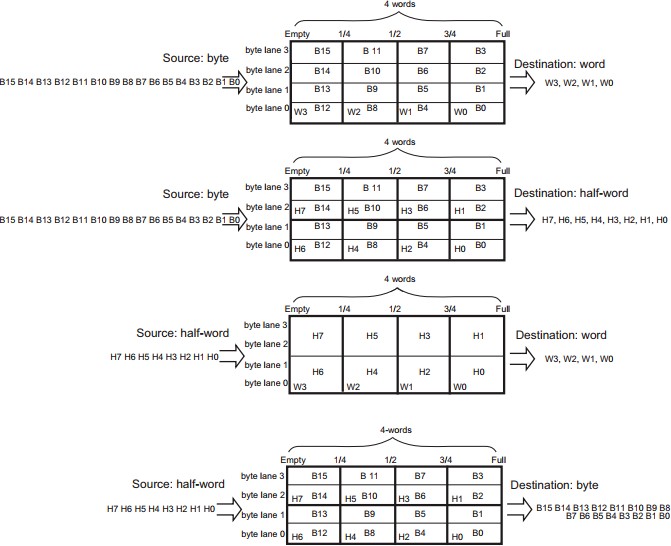
FIFO structure

The FIFO is used to temporarily store data coming from the source before transmitting them to the destination.

Each stream has an independent 4-word FIFO and the threshold level is software configurable between 1/4, 1/2, 3/4 or full.

To enable the use of the FIFO threshold level, the direct mode must be disabled.

The structure of the FIFO differs depending on the source and destination data widths, and is described in the figure below:



*Figure 14 DMA FIFO structure*

## FIFO structure

The FIFO can be flushed when the stream is disabled by resetting the EN bit in the DMA control register and when the stream is configured to manage peripheral-to-memory or memory-to- memory transfers: If some data are still present in the FIFO when the stream is disabled, the DMA controller continues transferring the remaining data to the destination (even though stream is effectively disabled). When this flush is completed, the transfer complete status bit in the DMA\_LISR or DMA\_HISR register is set.

The remaining data counter keeps the value in this case to indicate how many data items are currently available in the destination memory.

If the number of remaining data items in the FIFO is lower than a burst size, single transactions will be generated to complete the FIFO flush.

## Direct mode

By default, the FIFO operates in direct mode and the FIFO threshold level is not used. This mode is useful when the system requires an immediate and single transfer to or from the memory after each DMA request.

When the DMA is configured in direct mode (FIFO disabled), to transfer data in memory-to peripheral mode, the DMA preloads one data from the memory to the internal FIFO to ensure an immediate data transfer as soon as a DMA request is triggered by a peripheral.

To avoid saturating the FIFO, it is recommended to configure the corresponding stream with a high priority.

This mode is restricted to transfers where:

* + The source and destination transfer widths are equal and both defined by the peripherals size bits in DMA control register
  + Burst transfers are not possible

Direct mode must not be used when implementing memory-to-memory transfers.

## DMA transfer completion

Different events can generate an end of transfer by setting the bit in the status registers:

* + In DMA flow controller mode:
* The counter has reached zero in the memory-to-peripheral mode
* The stream is disabled before the end of transfer (by clearing the module enable bit in the DMA control register) and (when transfers are peripheral-to-memory or memory-to-memory) all the remaining data have been flushed from the FIFO into the memory
  + In Peripheral flow controller mode:
* The last external burst or single request has been generated from the peripheral and (when the DMA is operating in peripheral-to-memory mode) the remaining data have been transferred from the FIFO into the memory
* The stream is disabled by software, and (when the DMA is operating in peripheral-to-memory mode) the remaining data have been transferred from the FIFO into the memory

If the stream is configured in noncircular mode, after the end of the transfer (that is when the number of data to be transferred reaches zero), the DMA is stopped (module enable bit in DMA control register is cleared by Hardware) and no DMA request is served unless the software reprograms the stream and re-enables it (by setting the enable bit in the DMA control register).

## DMA transfer suspension

At any time, a DMA transfer can be suspended to be restarted later on or to be definitively disabled before the end of the DMA transfer.

There are two cases:

* + The stream disables the transfer with no later-on restart from the point where it was stopped. There is no particular action to do, except to clear the enable bit in the DMA control register to disable the stream. The stream may take time to be disabled (ongoing transfer is completed first). The transfer complete interrupt flag is set in order to indicate the end of transfer. The value of the enable bit in DMA control register is now ‘0’ to confirm the stream interruption. The counter register contains the number of remaining data items at the moment when the stream was stopped so that the software can determine how many data items have been transferred before the stream was interrupted.
  + The stream suspends the transfer before the number of remaining data items to be transferred in the DMA counter register reaches 0. The aim is to restart the transfer later by re-enabling the stream. In order to restart from the point where the transfer was stopped, the software has to read the DMA counter register after disabling the stream by writing the enable bit in DMA control register (and then checking that it is at ‘0’) to know the number of data items already collected. Then:
* The peripheral and/or memory addresses have to be updated in order to adjust the address pointers
* The counter register has to be updated with the remaining number of data items to be transferred (the value read when the stream was disabled)
* The stream may then be re-enabled to restart the transfer from the point it was stopped

## Flow controller

The entity that controls the number of data to be transferred is known as the flow controller.

This flow controller is configured independently for each stream using a specific bit in the DMA control register.

The flow controller can be:

* + The DMA controller: in this case, the number of data items to be transferred is programmed by software into the register before the DMA stream is enabled.
  + The peripheral source or destination: this is the case when the number of data items to be transferred is unknown. The peripheral indicates by hardware to the DMA controller when the last data are being transferred. This feature is only supported for peripherals which are able to signal the end of the transfer, that is:

## DMA interrupts

For each DMA stream, an interrupt can be produced on the following events:

* + Half-transfer reached
  + Transfer complete
  + Transfer error
  + FIFO error (overrun, underrun or FIFO level error)
  + Direct mode error

## Stream configuration procedure

The following sequence should be followed to configure a DMA stream x (where x is the stream number):

1. If the stream is enabled, disable it, then read this bit in order to confirm that there is no ongoing stream operation. Writing this bit to 0 is not immediately effective since it is actually written to 0 once all the current transfers have finished. When the EN bit is read as 0, this means that the stream is ready to be configured. It is therefore necessary to wait for the EN bit to be cleared before starting any stream configuration. All the stream dedicated bits set in the status register from the previous data block DMA transfer should be cleared before the stream can be re-enabled.
2. Set the peripheral port register address. The data will be moved from/ to this address to/ from the peripheral port after the peripheral event.
3. Set the memory address. The data will be written to or read from this memory after the peripheral event.
4. Configure the total number of data items to be transferred.

After each peripheral event or each beat of the burst, this value is decremented.

1. Select the DMA channel (request).
2. If the peripheral is intended to be the flow controller and if it supports this feature.
3. Configure the stream priority.
4. Configure the FIFO usage (enable or disable, threshold in transmission and reception)
5. Configure the data transfer direction, peripheral and memory incremented/fixed mode, single or burst transactions, peripheral and memory data widths, Circular mode, Double buffer mode and interrupts after half and/or full transfer, and/or errors.
6. Activate the stream.

As soon as the stream is enabled, it can serve any DMA request from the peripheral connected to the stream. Once half the data have been transferred on the AHB destination port, the half- transfer flag is set and an interrupt is generated if the half-transfer interrupt enable bit is set. At the end of the transfer, the transfer complete flag is set and an interrupt is generated if the transfer complete interrupt enable bit is set.

## Serial Peripheral Interface/ Inter-IC Sound (SPI/I2S)

## SPI/I2S Introduction

The SPI interface provides two main functions, supporting either the SPI protocol or the I2S audio protocol. By default, it is the SPI function that is selected. It is possible to switch the interface from SPI to I2S by software.

The serial peripheral interface (SPI) allows half/ full-duplex, synchronous, serial communication with external devices. The interface can be configured as the master and in this case, it provides the communication clock (SCK) to the external slave device. The interface is also capable of operating in multi-master configuration.

It may be used for a variety of purposes, including simplex synchronous transfers on two lines with a possible bidirectional data line or reliable communication using CRC checking.

The I2S is also a synchronous serial communication interface. It can address four different audio standards including the I2S Philips standard, the MSB- and LSB-justified standards, and the PCM standard. It can operate as a slave or a master device in full-duplex mode (using 4 pins) or in half-duplex mode (using 3 pins). Master clock can be provided by the interface to an external slave component when the I2S is configured as the communication master.

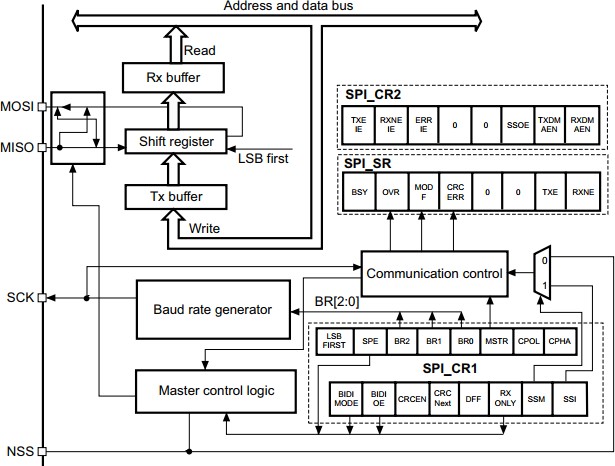
## SPI and I2S main features SPI features

* + Full-duplex synchronous transfers on three lines
  + Simplex synchronous transfers on two lines with or without a bidirectional data line
  + 8- or 16-bit transfer frame format selection
  + Master or slave operation
  + Multi-master mode capability
  + 8 master mode baud rate pre-scalers (fPCLK/2 max.)
  + Slave mode frequency (fPCLK/2 max)
  + Faster communication for both master and slave
  + NSS management by hardware or software for both master and slave: dynamic change of master/slave operations
  + Programmable clock polarity and phase
  + Programmable data order with MSB-first or LSB-first shifting
  + Dedicated transmission and reception flags with interrupt capability
  + SPI bus busy status flag
  + SPI TI mode
  + Hardware CRC feature for reliable communication:
* CRC value can be transmitted as last byte in Tx mode
* Automatic CRC error checking for last received byte
  + Master mode fault, overrun and CRC error flags with interrupt capability
  + 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests

## I2S features

* + Full duplex communication
  + Half-duplex communication (only transmitter or receiver)
  + Master or slave operations
  + 8-bit programmable linear pre-scaler to reach accurate audio sample frequencies (from 8 kHz to 192 kHz)
  + Data format may be 16-bit, 24-bit or 32-bit
  + Packet frame is fixed to 16-bit (16-bit data frame) or 32-bit (16-bit, 24-bit, 32-bit data frame) by audio channel
  + Programmable clock polarity (steady state)
  + Underrun flag in slave transmission mode, overrun flag in reception mode (master and slave), and Frame Error flag in reception and transmission mode (slave only)
  + 16-bit register for transmission and reception with one data register for both channel sides
  + Supported I2S protocols:
* I2S Phillips standard
* MSB-justified standard (left-justified)
* LSB-justified standard (right-justified)
* PCM standard (with short and long frame synchronization on 16-bit channel frame or 16-bit data frame extended to 32-bit channel frame)
  + Data direction is always MSB first
  + DMA capability for transmission and reception (16-bit wide)
  + Master clock may be output to drive an external audio component. Ratio is fixed at 256 × FS (where FS is the audio sampling frequency)
  + Both I2S (I2S2 and I2S3) have a dedicated PLL (PLLI2S) to generate an even more accurate clock.
  + I2S (I2S2 and I2S3) clock can be derived from an external clock mapped on the I2S\_CKIN pin.

## SPI functional description General description



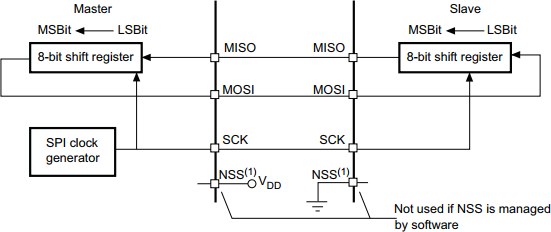
*Figure 15 The block diagram of the SPI*

Usually, the SPI is connected to external devices through four pins:

* + MISO: Master In / Slave Out data. This pin can be used to transmit data in slave mode and receive data in master mode.
  + MOSI: Master Out / Slave In data. This pin can be used to transmit data in master mode and receive data in slave mode.
  + SCK: Serial Clock output for SPI masters and input for SPI slaves.
  + NSS: Slave select. This is an optional pin to select a slave device. This pin acts as a ‘chip select’ to let the SPI master communicate with slaves individually and to avoid contention on the data lines. Slave NSS inputs can be driven by standard IO ports on the master device. The NSS pin may also be used as an output if enabled and driven low if the SPI is in master

configuration. In this manner, all NSS pins from devices connected to the Master NSS pin see a low level and become slaves when they are configured in NSS hardware mode. When configured in master mode with NSS configured as an input and if NSS is pulled low, the SPI enters the master mode fault state: the MSTR bit is automatically cleared and the device is configured in slave mode.

A basic example of interconnections between a single master and a single slave is illustrated in below figure



*Figure 16 Single master/ single slave application*

The MOSI pins are connected together and the MISO pins are connected together. In this way data is transferred serially between master and slave (most significant bit first).

The communication is always initiated by the master. When the master device transmits data to a slave device via the MOSI pin, the slave device responds via the MISO pin. This implies full- duplex communication with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

## Slave select (NSS) pin management

Hardware or software slave select management can be set using the SPI\_CR1 register.

* + Software NSS management

The slave select information is driven internally by the value of the SSI bit in the SPI\_CR1 register. The external NSS pin remains free for other application uses.

* + Hardware NSS management

Two configurations are possible depending on the NSS output configuration

* NSS output enabled

This configuration is used only when the device operates in master mode. The NSS signal is driven low when the master starts the communication and is kept low until the SPI is disabled.

* NSS output disabled

This configuration allows multi-master capability for devices operating in master mode. For devices set as slave, the NSS pin acts as a classical NSS input: the slave is selected when NSS is low and deselected when NSS high.

## Clock phase and clock polarity

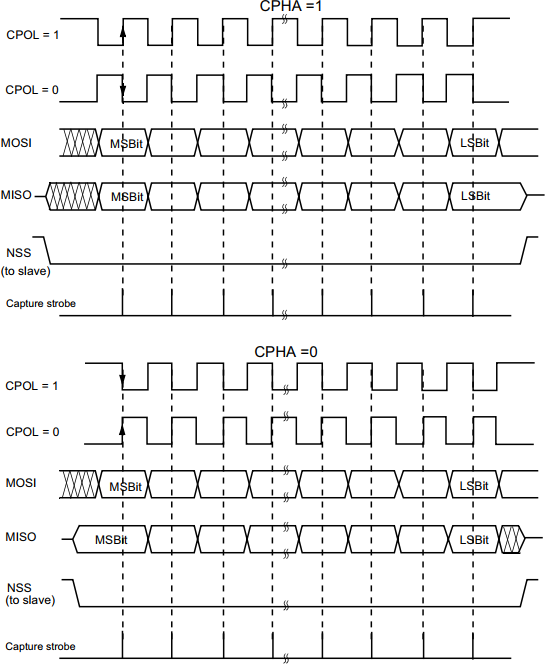
Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits in the SPI\_CR1 register. The CPOL (clock polarity) bit controls the steady state value of the clock when no data is being transferred. This bit affects both master and slave modes. If CPOL is reset, the SCK pin has a low-level idle state. If CPOL is set, the SCK pin has a high-level idle state.

If the CPHA (clock phase) bit is set, the second edge on the SCK pin (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set) is the MSB bit capture strobe. Data are latched on the occurrence of the second clock transition. If the CPHA bit is reset, the first edge on the SCK pin (falling edge if CPOL bit is set, rising edge if CPOL bit is reset) is the MSB bit capture strobe. Data are latched on the occurrence of the first clock transition.

The combination of the CPOL (clock polarity) and CPHA (clock phase) bits selects the data capture clock edge.

Figure below shows an SPI transfer with the four combinations of the CPHA and CPOL bits.

The diagram may be interpreted as a master or slave timing diagram where the SCK pin, the MISO pin, the MOSI pin are directly connected between the master and the slave device.



*Figure 17 Data clock timing diagram*

## Data frame format

Data can be shifted out either MSB-first or LSB-first depending on the value of the LSBFIRST bit in the SPI\_CR1 Register.

Each data frame is 8 or 16 bits long depending on the size of the data programmed using the DFF bit in the SPI\_CR1 register. The selected data frame format is applicable for transmission and/or reception.

## Status flags

With STM32F4xx, there are some flags to manage the operations of SPI protocol. Tx buffer empty flag (TXE)

When it is set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can be loaded into the buffer. The TXE flag is cleared when writing to the SPI\_DR register.

Rx buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the Rx buffer. It is cleared when SPI\_DR is read.

BUSY flag

This BSY flag is set and cleared by hardware (writing to this flag has no effect). The BSY flag indicates the state of the communication layer of the SPI.

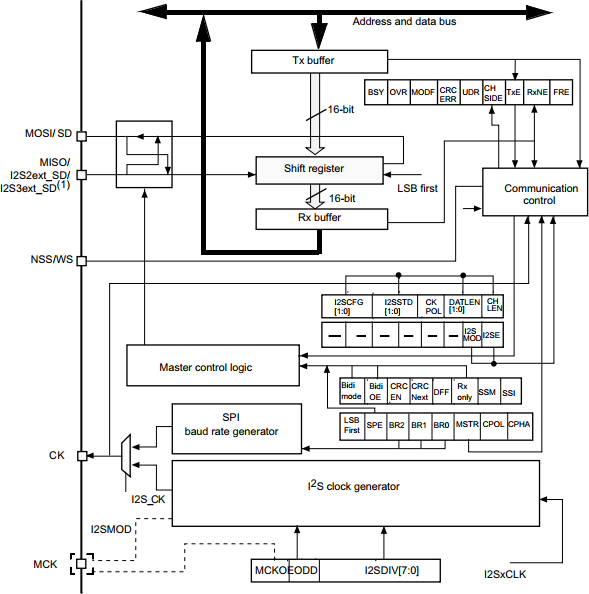
When BSY is set, it indicates that the SPI is busy communicating. There is one exception in master mode / bidirectional receive mode where the BSY flag is kept low during reception.

The BSY flag is useful to detect the end of a transfer if the software wants to disable the SPI and enter Halt mode (or disable the peripheral clock). This avoids corrupting the last transfer.

## Data transmission and reception procedures

In reception, data are received and then stored into an internal Rx buffer while in transmission, data are first stored into an internal Tx buffer before being transmitted. A read access of the SPI\_DR register returns the Rx buffered value whereas a write access to the SPI\_DR stores the written data into the Tx buffer.

## I2S functional description



*Figure 18 I2S block diagram*

The I2S shares three common pins with the SPI:

* + SD: Serial Data (mapped on the MOSI pin) to transmit or receive the two multiplexed data channels (in half-duplex mode only).
  + WS: Word Select (mapped on the NSS pin) is the data control signal output in master mode and input in slave mode.
  + CK: Serial Clock (mapped on the SCK pin) is the serial clock output in master mode and serial clock input in slave mode.
  + I2S2ext\_SD and I2S3ext\_SD: additional pins (mapped on the MISO pin) to control the I2S full duplex mode.

An additional pin could be used when a master clock output is needed for some external audio devices:

* + MCK: Master Clock (mapped separately) is used, when the I2S is configured in master mode to output this additional clock generated at a preconfigured frequency rate equal to 256 × FS, where FS is the audio sampling frequency.

The I2S uses its own clock generator to produce the communication clock when it is set in master mode. This clock generator is also the source of the master clock output. Two additional registers are available in I2S mode. One is linked to the clock generator configuration and the other one is a generic I2S configuration register (audio standard, slave/master mode, data format, packet frame, clock polarity, etc.).

The I2S uses the same SPI register for data transfer in 16-bit wide mode

## Supported audio protocols

The four-line bus has to handle only audio data generally time-multiplexed on two channels: the right channel and the left channel. However, there is only one 16-bit register for the transmission and the reception. So, it is up to the software to write into the data register the adequate value corresponding to the considered channel side, or to read the data from the data register and to identify the corresponding channel by checking the CHSIDE bit in the SPI\_SR register. Channel Left is always sent first followed by the channel right (CHSIDE has no meaning for the PCM protocol).

Four data and packet frames are available. Data may be sent with a format of:

* + 16-bit data packed in 16-bit frame
  + 16-bit data packed in 32-bit frame
  + 24-bit data packed in 32-bit frame
  + 32-bit data packed in 32-bit frame

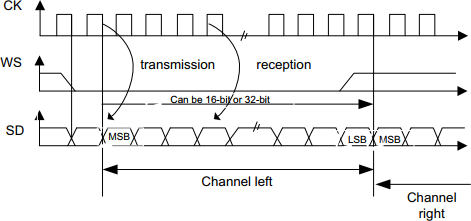
When using 16-bit data extended on 32-bit packet, the first 16 bits (MSB) are the significant bits, the 16-bit LSB is forced to 0 without any need for software action or DMA request (only one read/write operation).

The 24-bit and 32-bit data frames need two CPU read or write operations to/from the SPI\_DR or two DMA operations if the DMA is preferred for the application. For 24-bit data frame specifically, the 8 non -significant bits are extended to 32 bits with 0-bits (by hardware).

For all data formats and communication standards, the most significant bit is always sent first (MSB first).

The I2S interface supports four audio standards. I will describe in detail which protocol I use in this application, Phillip audio protocol.

For this standard, the WS signal is used to indicate which channel is being transmitted. It is activated one CK clock cycle before the first bit (MSB) is available.



*Figure 19 I2S Philips protocol waveforms*

Data are latched on the falling edge of CK (for the transmitter) and are read on the rising edge (for the receiver). The WS signal is also latched on the falling edge of CK.

For transmission, each time an MSB is written to SPI\_DR, the TXE flag is set and its interrupt, if allowed, is generated to load SPI\_DR with the new value to send.

For reception, the RXNE flag is set and its interrupt, if allowed, is generated when the first 16 MSB half-word is received.

In this way, more time is provided between two write or read operations, which prevents underrun or overrun conditions (depending on the direction of the data transfer).

## I2S flags and interrupts

To manage the transmitting and receiving operations, I2S protocol supports below interrupt flags:

## Busy flag (BSY)

The BSY flag is set and cleared by hardware (writing to this flag has no effect). It indicates the state of the communication layer of the I2S.

When BSY is set, it indicates that the I2S is busy communicating. There is one exception in master receive mode where the BSY flag is kept low during reception.

The BSY flag is useful to detect the end of a transfer if the software needs to disable the I2S.

This avoids corrupting the last transfer. For this, the procedure described below must be strictly respected.

The BSY flag is set when a transfer starts, except when the I2S is in master receiver mode. The BSY flag is cleared:

* + When a transfer completes (except in master transmit mode, in which the communication is supposed to be continuous)
  + When the I2S is disabled

When communication is continuous:

* + In master transmit mode, the BSY flag is kept high during all the transfers
  + In slave mode, the BSY flag goes low for one I2S clock cycle between each transfer

## TX buffer empty flag (TXE)

When set, this flag indicates that the Tx buffer is empty and the next data to be transmitted can then be loaded into it. The TXE flag is reset when the Tx buffer already contains data to be transmitted. It is also reset when the I2S is disabled (I2SE bit is reset).

## RX buffer not empty (RXNE)

When set, this flag indicates that there are valid received data in the RX Buffer. It is reset when SPI\_DR register is read.

## Channel Side flag (CHSIDE)

In transmission mode, this flag is refreshed when TXE goes high. It indicates the channel side to which the data to transfer on SD has to belong. In case of an underrun error event in slave transmission mode, this flag is not reliable and I2S needs to be switched off and switched on before resuming the communication.

In reception mode, this flag is refreshed when data are received into SPI\_DR. It indicates from which channel side data have been received. Note that in case of error (like OVR) this flag becomes meaningless and the I2S should be reset by disabling and then enabling it (with configuration if it needs changing).

This flag has no meaning in the PCM standard (for both Short and Long frame modes).

## I2C protocol

## Introduction

I²C (Inter-Integrated Circuit), is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented in 1982 by Philips Semiconductor (now NXP Semiconductors). It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

## I2C bus features

In consumer electronics, telecommunications and industrial electronics, there are often many similarities between seemingly unrelated designs. For example, nearly every system includes:

* + Some intelligent control, usually a single-chip microcontroller
  + General-purpose circuits like LCD and LED drivers, remote I/O ports, RAM, EEPROM, real- time clocks or A/D and D/A converters
  + Application-oriented circuits such as digital tuning and signal processing circuits for radio and video systems, temperature sensors, and smart cards

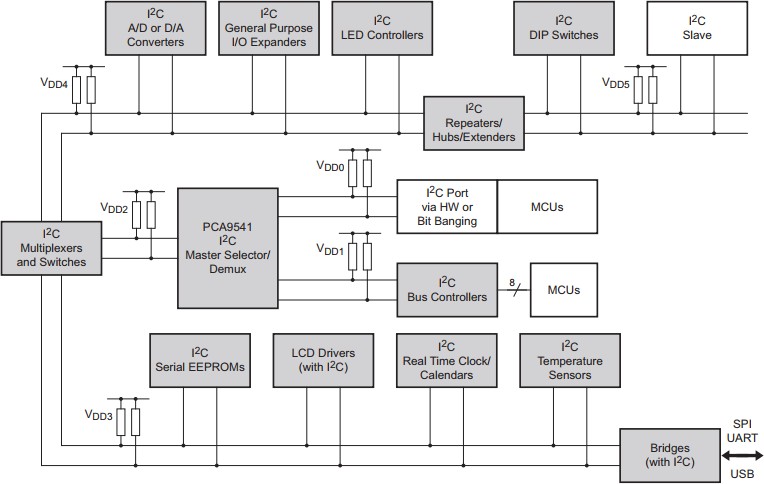
This bus is called the Inter IC or I2C-bus. All I2C-bus compatible devices incorporate an on- chip interface which allows them to communicate directly with each other via the I2C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits.

Here are some of the features of the I2C-bus:

* + Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL).
  + Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers.
  + It is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.
  + Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in Fast-mode Plus, or up to

3.4 Mbit/s in the High-speed mode.

* + Serial, 8-bit oriented, unidirectional data transfers up to 5 Mbit/s in Ultra Fast-mode
  + On-chip filtering rejects spikes on the bus data line to preserve data integrity.
  + The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance. More capacitance may be allowed under some conditions.



*Figure 20 Example of I2C-bus applications*

## I2C mode

Originally, the I2C-bus was limited to 100 kbit/s operation. Over time there have been several additions to the specification so that there are now five operating speed categories. Standard- mode, Fast-mode (Fm), Fast-mode Plus (Fm+), and High-speed mode (Hs-mode) devices are downward-compatible — any device may be operated at a lower bus speed. Ultra Fast-mode devices are not compatible with previous versions since the bus is unidirectional.

* + Bidirectional bus:
* Standard-mode (Sm), with a bit rate up to 100 kbit/s
* Fast-mode (Fm), with a bit rate up to 400 kbit/s
* Fast-mode Plus (Fm+), with a bit rate up to 1 Mbit/s
* High-speed mode (Hs-mode), with a bit rate up to 3.4 Mbit/s.
  + Unidirectional bus:
* Ultra Fast-mode (UFm), with a bit rate up to 5 Mbit/s

## The I2C-bus protocol

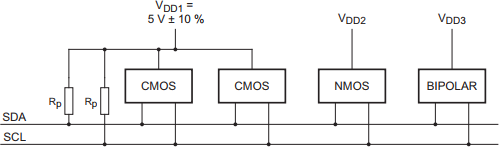
In this below part of project, I only describe the I2C protocol operation for I2C bidirectional bus modes.

## SDA and SCL signals

I2C bus protocol uses two line to transmit and receive: Serial Data (SDA) and Serial Clock (SCL). Both SDA and SCL are bidirectional lines, connected to a positive supply voltage via a current-source or pull-up resistor. When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired- AND function. Data on the I2C-bus can be transferred at rates of up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in Fast-mode Plus, or up to

3.4 Mbit/s in the High-speed mode. The bus capacitance limits the number of interfaces connected to the bus.

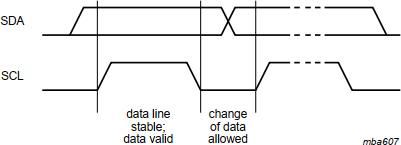
For a single master application, the master’s SCL output can be a push-pull driver design if there are no devices on the bus which would stretch the clock.



## Data validity

*Figure 21 Devices with various supply voltages sharing the same bus*

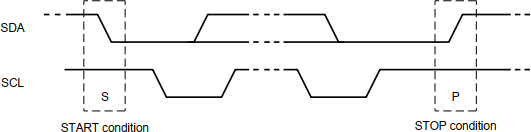
The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW. One clock pulse is generated for each data bit transferred.



*Figure 22 Bit transfer on the I2C-bus*

## START and STOP conditions

All transactions begin with a START (S) and are terminated by a STOP (P). A HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.



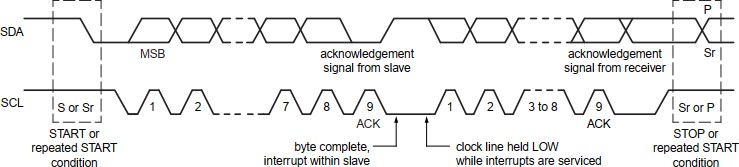
*Figure 23 START and STOP conditions*

START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free again a certain time after the STOP condition.

The bus stays busy if a repeated START (Sr) is generated instead of a STOP condition. In this respect, the START (S) and repeated START (Sr) conditions are functionally identical.

## Byte format

Every byte put on the SDA line must be eight bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte must be followed by an Acknowledge bit. Data is transferred with the Most Significant Bit (MSB) first. If a slave cannot receive or transmit another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL LOW to force the master into a wait state. Data transfer then continues when the slave is ready for another byte of data and releases clock line SCL.



*Figure 24 Data transfer on the I2C-bus*

## Acknowledge (ACK) and Not Acknowledge (NACK)

The acknowledge takes place after every byte. The acknowledge bit allows the receiver to signal the transmitter that the byte was successfully received and another byte may be sent. The master generates all clock pulses, including the acknowledge ninth clock pulse.

The Acknowledge signal is defined as follows: the transmitter releases the SDA line during the acknowledge clock pulse so the receiver can pull the SDA line LOW and it remains stable LOW during the HIGH period of this clock pulse.

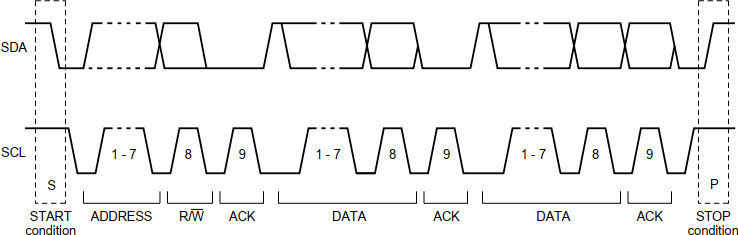
When SDA remains HIGH during this ninth clock pulse, this is defined as the Not Acknowledge signal. The master can then generate either a STOP condition to abort the transfer, or a repeated START condition to start a new transfer. There are five conditions that lead to the generation of a NACK:

1. No receiver is present on the bus with the transmitted address so there is no device to respond with an acknowledge.
2. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start communication with the master.
3. During the transfer, the receiver gets data or commands that it does not understand.
4. During the transfer, the receiver cannot receive any more data bytes.
5. A master-receiver must signal the end of the transfer to the slave transmitter.

## The slave address and R/W bit

Data transfers follow the format shown in below figure. After the START condition (S), a slave address is sent. This address is seven bits long followed by an eighth bit which is a data direction bit (R/W) — a ‘zero’ indicates a transmission (WRITE), a ‘one’ indicates a request for data (READ). A data transfer is always terminated by a STOP condition (P) generated by the master.

However, if a master still wishes to communicate on the bus, it can generate a repeated START condition (Sr) and address another slave without first generating a STOP condition. Various combinations of read/write formats are then possible within such a transfer.



*Figure 25 A complete data transmission*

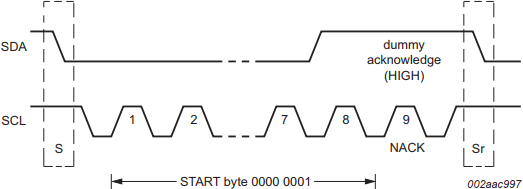
## START byte

Microcontrollers can be connected to the I2C-bus in two ways. A microcontroller with an on- chip hardware I2C-bus interface can be programmed to be only interrupted by requests from the bus. When the device does not have such an interface, it must constantly monitor the bus via software. Obviously, the more times the microcontroller monitors, or polls the bus, the less time it can spend carrying out its intended function.

There is therefore a speed difference between fast hardware devices and a relatively slow microcontroller which relies on software polling.

In this case, data transfer can be preceded by a start procedure which is much longer than normal. The start procedure consists of:

* + A START condition (S)
  + A START byte (0000 0001)
  + An acknowledge clock pulse (ACK)
  + A repeated START condition (Sr).



*Figure 26 START byte procedure*

After the START condition S has been transmitted by a master which requires bus access, the START byte (0000 0001) is transmitted. Another microcontroller can therefore sample the SDA line at a low sampling rate until one of the seven zeros in the START byte is detected. After detection of this LOW level on the SDA line, the microcontroller can switch to a higher sampling rate to find the repeated START condition Sr which is then used for synchronization.

A hardware receiver resets upon receipt of the repeated START condition Sr and therefore ignores the START byte.

An acknowledge-related clock pulse is generated after the START byte. This is present only to conform with the byte handling format used on the bus. No device is allowed to acknowledge the START byte

## USB protocol

USB, or the Universal Serial Bus Interface is now well established as an interface for computer communications. In many areas it has completely overtaken RS232 and the parallel or Centronics interface for printers, and it is also widely used for memory sticks, computer mice, keyboards and for many other functions. One of the advantages of USB is its flexibility: another is the speed that USB provides.

USB gives quite fast serial data transfer method for data communication, however power can also be obtained through connectors and this has further enhanced the popularity of USB as many low power computer accessories. USB find a wide utility from memory to disk drives. The development USB interface was as a result of the demand for a data transfer interface that was easy to use and one that supports higher data rates which is a key requirement for the computer and peripherals industries.



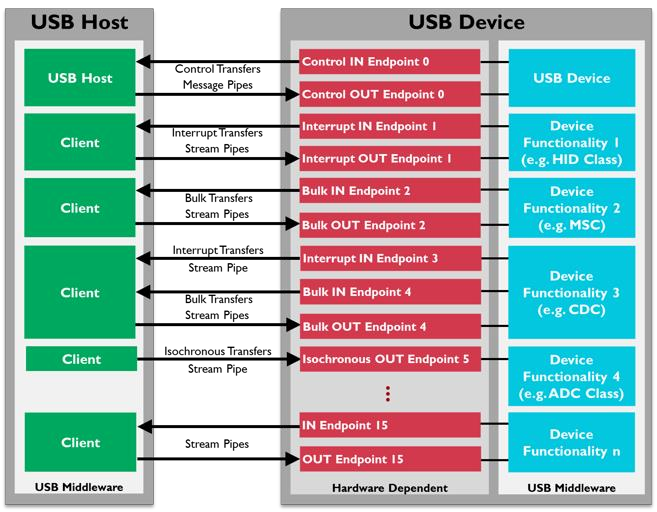
*Figure 27 USB flash drive*

With USB 1.0 well established, faster data transfer rates were required, and accordingly a new specification, USB 2 was released. With the importance of USB already established it did not take long for the new standard to be adopted.

With USB shaping its place in the computer market, other improvements of the standard had to be researched. With the need for portability in many areas of the electronics industry taking off, the next predictable move for USB could have been to employ a wireless interface. In making it possible USB will need to employ a flexible methodology that has proved to a success for a wired interface. Moreover, the wireless USB interface has to be able to send data at rates higher than the wired USB 2 connections.

## USB Communication

USB is a serial bus, in which all the data transfer and receiving is initiated by the USB Host. The data is transmitted to or from endpoints in an USB Device. The client in the USB Host stores data in buffers, but does not have endpoints. As shown below different layers of data transmission can be seen. The interaction across different layers are logical Host-Device connection between each horizontal layer. Between the logical connections data is transferred using pipes.



*Figure 28 Logical Connections between USB Host Clients and USB Device Endpoints*

## USB Advantages and Disadvantages

USB has many advantages when compared to other technologies, but it also has a number of disadvantages which need to be considered when deciding on a technology to be used.

*Table 2 Advantages and Disadvantages of USB*

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| User friendly | Data transfer speed is not as fast as other Communication Protocols |
| Many applications can find its utility with data transfer speed in range. | Capabilities and speed are limited. |
| Connector system is robust |  |
| Variety of connector types available |  |
| Cost is low |  |

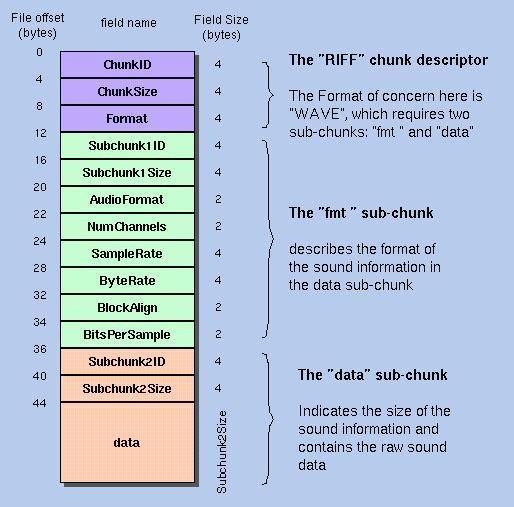
USB has many advantages and this is why it is so widely used. However, its simplicity and ease of use, mean that it is not always applicable in applications where more sophisticated interfaces are required for very high-speed data transfer.

## WAV File Format

## Wave Audio File Format

The Wave file format is Windows' local file format for storing computerized digital audio information. It has turned into a standout amongst the most broadly upheld computerized audio formats on the PC because of the notoriety of Windows and the immense number of projects composed for the stage. In this file format the arrangement of data is little-endian that is the least significant byte has to appear first.

Wave files make use of a general Resource Interchange File Format (RIFF)] structure which stores the contents of the file in to different “chunks”. The chunk header tells the application about the type and the length of data bytes stored in the chunk. This method of organization makes a program powerful by allowing it to skip over unrecognizable types of chunk and continue the processing of known data chunks. Following figure shows the basic wave file layout.



*Figure 29 WAV File Format Layout*

## Wave file header

It follows standard RIFF structure. The very first 8 bytes of the file is a RIFF chunk header which has an ID of “RIFF” and chunk size which is same as file size minus 8 bytes for header. WAV files are stored uncompressed therefore they can get quite large, but they cannot exceed 4 gigabytes due to the fact that the file size header field is a 32-bit unsigned integer.

A WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub- chunks -- a "fmt" chunk specifying the data format and a "data" chunk containing the actual sample data.

The following table gives the details of the wave file header.

*Table 3 Wave File Header*

|  |  |  |  |
| --- | --- | --- | --- |
| BYTE NUMBER | SIZE | DESCRIPTION | VALUES |
| 0-3 | 4 | Chunk ID | “RIFF” (0x52494646) |
| 4-7 | 4 | Chunk Data Size | 8 (file size) |
| 8-11 | 4 | RIFF type | “WAVE”  (0x57415645) |

## Format Chunk

Format chunk has detailed information about storing the data waveform and playing it including the mode of compression, channel number, rate of sampling, number of bits in each sample and other properties. Following table provides good insight into format chunk.

## Channel

* An independent waveform in the audio data. The number of channels is important: one channel is “Mono,” two channels is “Stereo” – there are different waves for the left and right speakers. 5.1 surround sound has 5 channels, one of which is for the lowest sounds and is usually sent to a subwoofer. Again, each channel holds audio data that is independent of all the other channels, although all channels will be the same overall length.

## Frame

* A frame is like a sample, but in multichannel format – it is a snapshot of all the channels at a specific data point.

## Sampling Rate/Sample Rate

* The number of samples (or frames) that exist for each second of data. This field is represented in Hz, or “per second.” For example, CD-quality audio has 44,100 samples per second. A higher sampling rate means higher fidelity audio.

## Bit Depth/Bits per Sample

* The number of bits available for one sample. Common bit depths are 8-bit, 16-bit and 32-bit. A sample is almost always represented by a native data type, such as byte, short, or int. A higher bit depth means each sample can be more precise, resulting in higher fidelity audio.

## Data chunk

*Table 4 Wav Data chunk*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BYTE NUMBER | SIZE (Bytes) | DESCRIPTION | VALUES | |
| 0-3 | 4 | Chunk ID | "fmt " (0x666D7420) |  |
|  | |
| 4-7 | 4 | Chunk Data Size | Length Of format | |
| Chunk (always 0x10) | |
| 8-9 | 2 | Compression Code | Always 0x01 | |
| 10-11 | 2 | Channel Number | 0x01=mono 0x02=stereo | |

|  |  |  |  |
| --- | --- | --- | --- |
| 12-15 | 4 | Sample Rate | Binary, in Hertz |
| 16-19 | 4 | Bytes Per Second |  |
| 20-21 | 2 | Bytes per Sample | 1=8 bit mono, 2=8 bit  stereo, 3=16 bit  mono, 4= 16 bit stereo |
| 22-23 | 2 | Bits Per Sample |  |

It contains the main digital audio data which has to be decoded by utilizing the format and compression scheme described by the wave format chunk. Following is the detailed structure of data chunk.

# CHAPTER III: HARDWARE AND DEVELOPMENT TOOLS

In this chapter, I will introduce hardware and software development tools I used in my project include:

* STM32F4 Discovery kit
* KEIL C uVision 5
* STM 32 Cube Mx

## STM32F4 Discovery kit

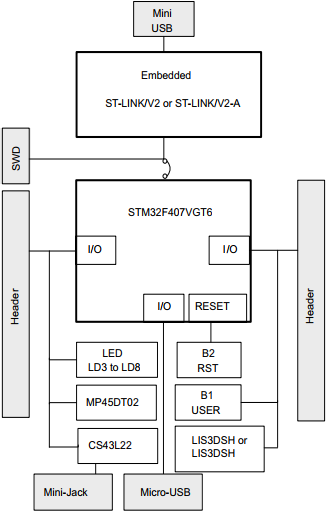
The STM32F4DISCOVERY is a low-cost and easy-to-use development kit to quickly Evaluate and start a development with an STM32F407VG high-performance microcontroller

## Features

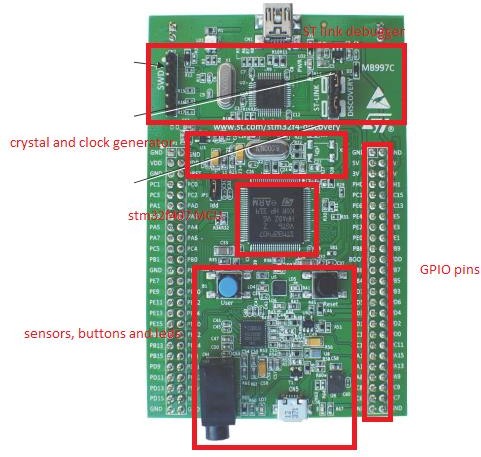
The STM32F4DISCOVERY offers the following features:

* + STM32F407VGT6 microcontroller featuring 32-bit ARM Cortex® -M4 with FPU core, 1-Mbyte Flash memory, 192-Kbyte RAM in an LQFP100 package
  + On-board ST-LINK/V2 on STM32F4DISCOVERY or ST-LINK/V2-A on STM32F407G-DISC1
  + USB ST-LINK with re-enumeration capability and three different interfaces:
* Virtual COM port (with ST-LINK/V2-A only)
* Mass storage (with ST-LINK/V2-A only)
* Debug port
  + Board power supply:
* Through USB bus
* External power sources: 3 V and 5 V
  + LIS302DL or LIS3DSH ST MEMS 3-axis accelerometer
  + MP45DT02 ST MEMS audio sensor omni-directional digital microphone
  + CS43L22 audio DAC with integrated class D speaker driver
  + Eight LEDs:
* LD1 (red/green) for USB communication
* LD2 (red) for 3.3 V power on
* Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)
* 2 USB OTG LEDs LD7 (green) VBUS and LD8 (red) over-current
  + Two push buttons (user and reset)
  + USB OTG FS with micro-AB connector

## Hardware structure



*Figure 30 Hardware block diagram*



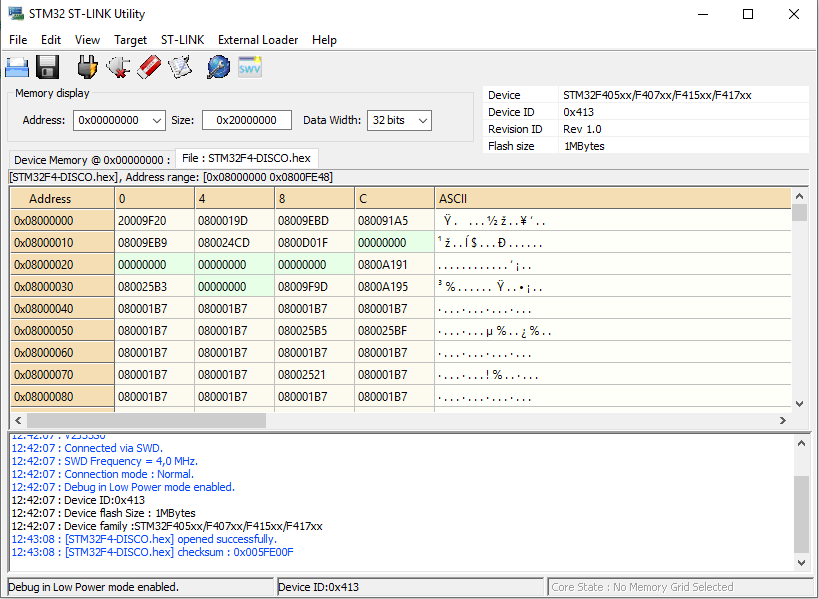
*Figure 31 STM32F4 DISCOVERY board*

## Embedded Debug Interface

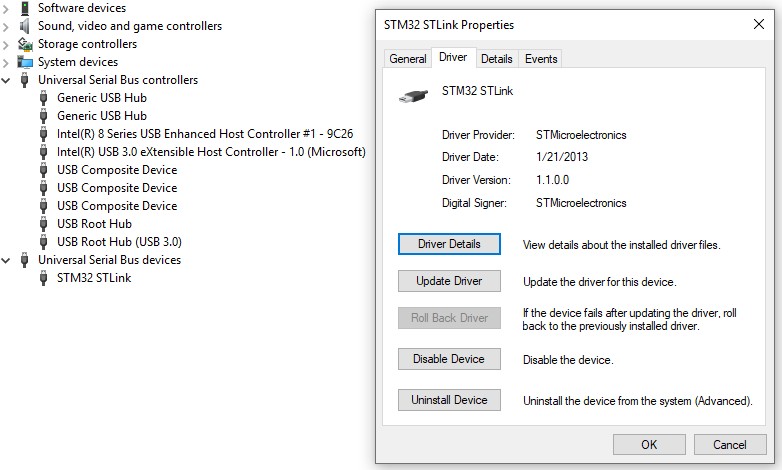
For debugging, STM32F4 DISCOVERY kit supports an on-board ST-LINK/V2 (or V2-A) interface. To debug, users only need to connect the development kit to computer via an USB connector. This interface is compatible with almost popular IDE like KeilC uVison, IAR embedded system workbench, Eclipse…

For the first use, users have to update driver for connecting the device. This is very easy due to a tool supported by ST, called STM32 ST Link Utility. The tool can be easily to downloaded in this link:

Otherwise, ST Link is also a powerful GUI tool for connect board, observe memory, erase flash memory of chip…



*Figure 32 STM32 ST Link interface*



*Figure 33 Driver update for STM board*

## On-board audio capability

The STM32F407VG microcontroller uses an audio DAC (CS43L22) to output sounds through the audio mini-jack connector.

The STM32F407VG microcontroller controls the audio DAC through the I2C interface and processes digital signals through an I2S connection or an analog input signal.

* + The sound can come independently from different inputs:
* ST-MEMS microphone (MP45DT02): digital using PDM protocol or analog when using the low pass filter
* USB connector: from external mass storage such as a USB key, USB HDD, and

so on

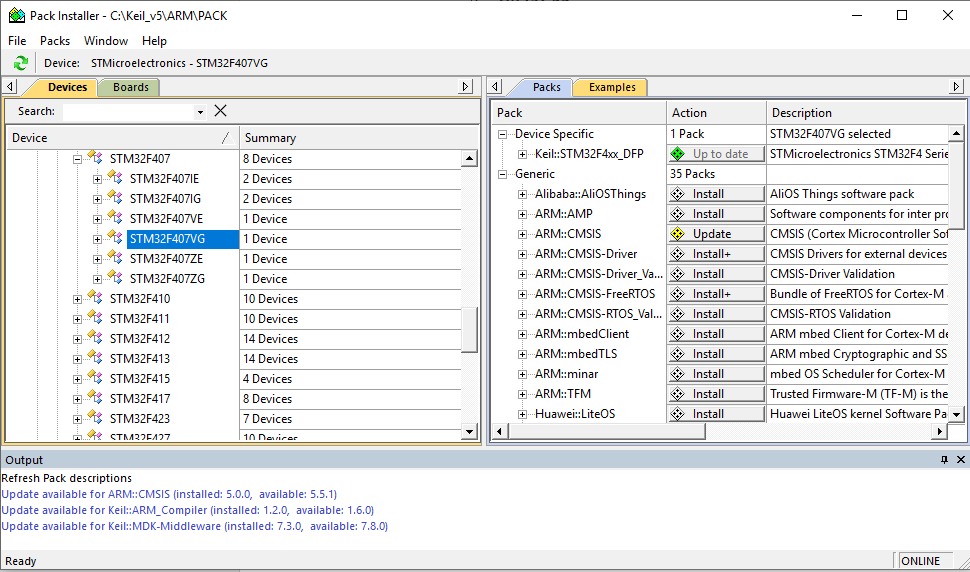
* Internal memory of the STM32F407VG microcontroller
  + The sound can be output in different ways through the audio DAC:
* Using I2S protocol
* Using DAC to analog input AIN1x of the CS43L22
* Using the microphone output directly via a low-pass filter to analog input AIN4x of the CS43L22

## Keil C uVison 5 IDE

Keil C uVison 5 is a powerful IDE for embedded programming and debugging. It supports almost MCU families of almost MCU manufacturers such as ST, NXP, TI, Microchip, Renesas… The tool can be easy to downloaded from the website of the developer according the below link:

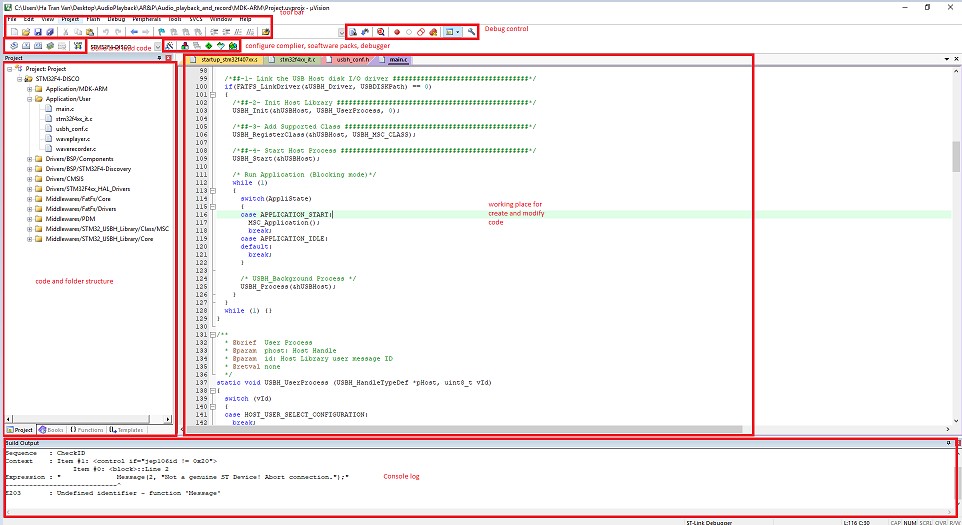
<https://www.keil.com/download/>

After downloading and installing, we now have to download the software packs for the microcontroller we want to use. These packs contain basic software that we need to initialize an MCU such a startup code, to compile and build a program such a linker file or peripherals library that help us easy to configure and use these peripherals. This work can be done with the pack installer that installed along with the IDE. All we need is to choose the pack we want and click install.



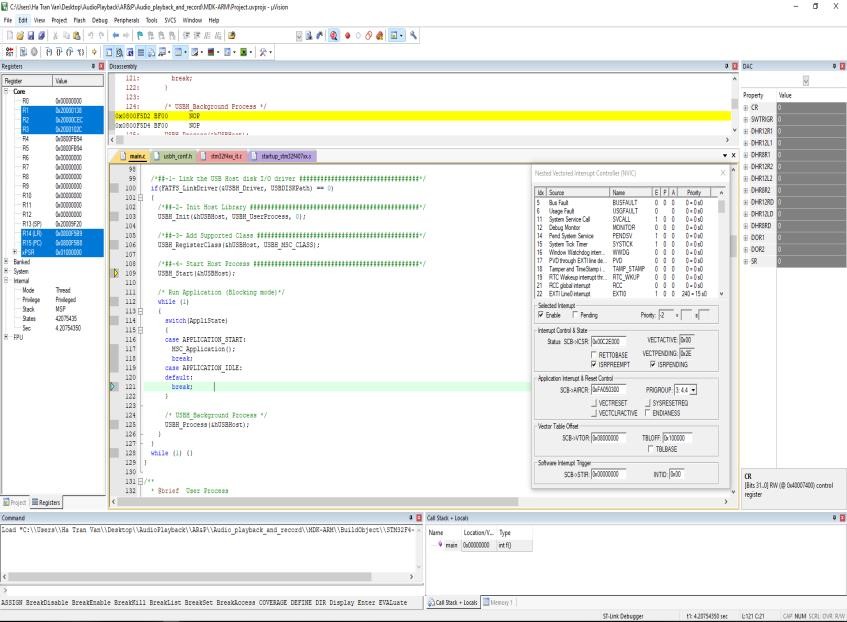
*Figure 34 Keil C pack installer*

After installing required software, we now can create a project for our own purpose. Below is the main working place of Keil C uVison 5.



*Figure 35 Keil C IDE interface*

Keil C also supports a nice and powerful debug interface. It allows developer to observe the behavior of every peripherals, assembly code, status of core register. With these functionalities, developer can easily debug and make their code works.

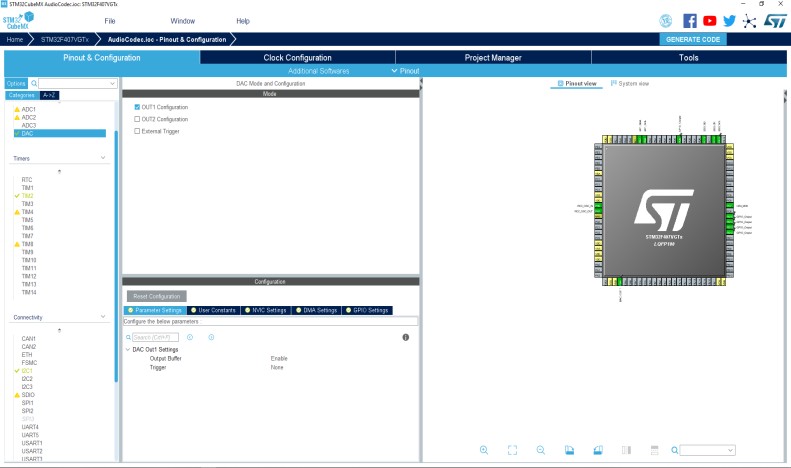


*Figure 36 Keil C Debug Interface*

## STM32 CubeMx

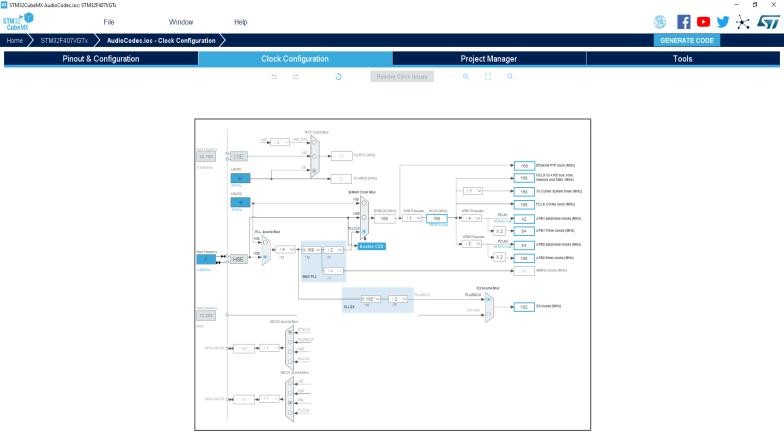
STM32 Cube Mx is a tool developer by ST to support users to update software pack for their product or configure peripherals easier. This useful software can be downloaded from ST’s website at below link:

<https://www.st.com/en/development-tools/stm32cubemx.html>

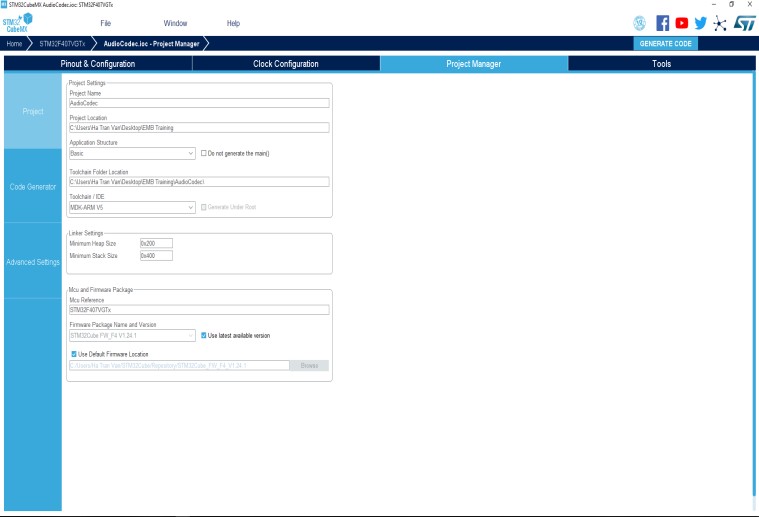


*Figure 37 ST Cube Mx GUI*

After choosing the MCU, developer can configure all the characteristics he wants like GPIO pins, clock, linker, code generator…



*Figure 38 Clock configuration with STM32 Cube Mx*



*Figure 39 Project management with ST Cube Mx*

# CHAPTER IV: SYSTEM DESIGN

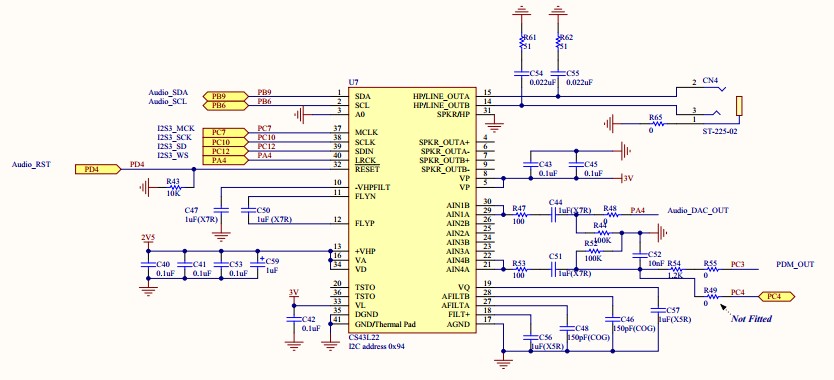
In this chapter, I will analyze the functionalities of the system, duties of every components, peripherals in the system and how to associate all of these to make a complete application.

There are three main parts in this chapter

* + Application Overview
  + Audio Playback Application
  + Audio Record Application

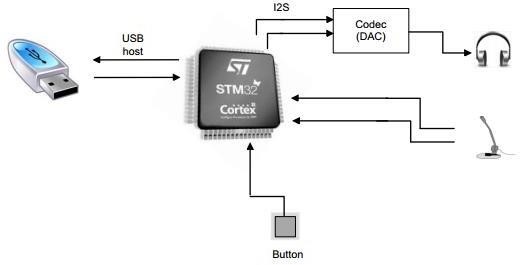
## Application overview

Wave audio files can be played using STM32F4 with USB storage device as the file destination. The application makes use of MEMS microphone, audio DAC, headphone and USB key.



*Figure 40 Schematic of Audio peripherals connection on STM32F4 discovery*

USB peripheral has to be configured in host mode. Mass Storage Class (MSC) is used to send/receive audio data to/from USB. I2S peripheral is configured in master transmitter mode and used to transmit audio data to the external audio codec (DAC). DMA is used to send from buffers to I2S peripherals which efficiently reduces the load on CPU. I2C peripheral is used to control external devices like audio codec and obtain data from that device. User button is for monitoring the operation (playback or recording).



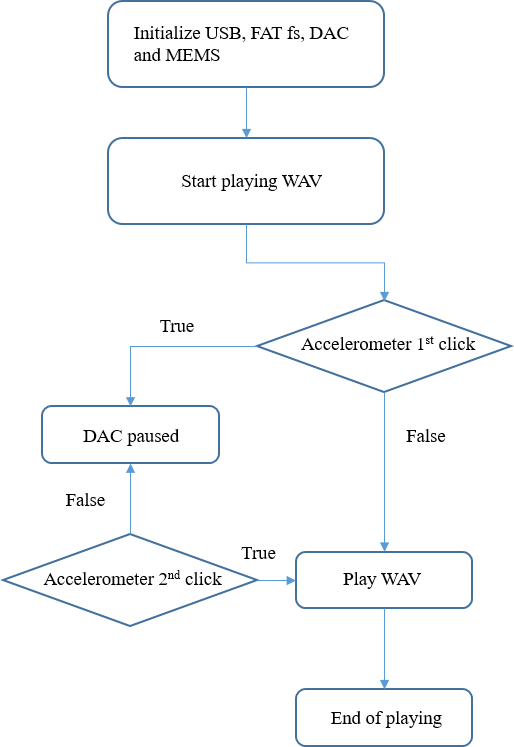
*Figure 41 Audio playback and record architecture*

## Audio Playback Application

Procedure for the audio play application on the board 1.Initialize the USB, Fat Fs File Systems, Audio DAC, MEMS

1. Transfer the WAV file from USB storage to internal SRAM of MCU, block by block (1024 bytes) using DMA in its first buffer.
2. DMA sends its data to I2S Peripheral which transfers it to external audio codec (DAC). 4.In the mean time data is stored from USB to DMA in its second buffer.

5.These two buffers are swapped indefinitely till the end of audio file.

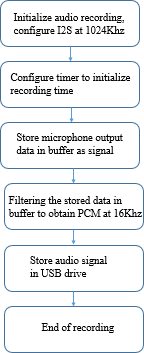


*Figure 42 Audio playback application flow chart*

## Audio Recorder Application

Procedure for the audio recording:

* + 1. Audio Record initialization: Configure I2S as 1024 Khz as an input clock for MEMS microphone
    2. Timer configuration to initialize recording time
    3. Store the microphone in buffer as signal
    4. Filter the stored data to obtain a PCM signal at 16 KHz sampling frequency
    5. Store the filtered signal in the USB mass media.

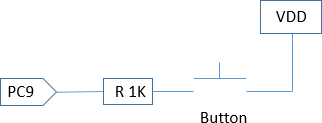


*Figure 43 Audio recording flow chart*

I2S peripheral has been configured in master mode in order to generate the correct clock cycle (1.024 MHz). The 1.024 MHz clock can be calculated from the output audio streaming (16 Khz) and the decimation factor (64) chosen for this application (16000 Hz x 64 = 1.024 MHz).

## Volume control

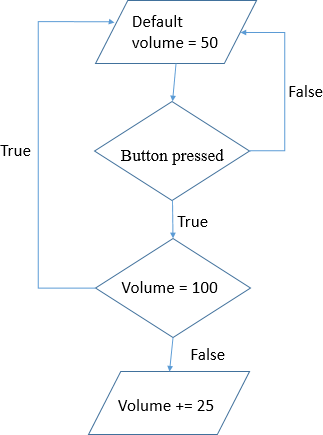
For controlling the volume, I use a button as the below schematic.



*Figure 44 Setup MCU pin for external trigger*

For detecting the button status, I set up MCU pin (PC9) for detect the external trigger (rising edge). By default, MCU pin is in internal pulldown mode so whenever the button is pressed, VDD is connected to PC9. This makes it pulled to high and also create a rising edge on this pin. This event causes an interrupt and the interrupt handler will be called to serve this event.

The changes of volume due to the button statuses is described in the below diagram.



*Figure 45 Volume control diagram*

# CHAPTER V: CONCLUSION AND REFERENCE

**Conclusion**

Through working on this project, I had chance to practice with microcontroller (how the interrupt works, how the clock is switched for each peripheral module, how the core executes an instruction and so on), learn about popular peripherals and associate its in a real application. Not only the knowledge about microcontroller structure, I had chance to improve my C coding skill when debugging. All brought me many precious experiences in creating an embedded product. In the future, I am going to keep working on this project to develop more functionalities such as, for instance, connect to a Bluetooth chipset to transmit the data to a speaker or laptop for live playing.

Although I tried my best but due to the limitations in both microcontroller knowledge and coding skills, I could not avoid the mistakes. That would be my fortune if I could receive feedbacks from readers or the examiners because these would allow me to realize my mistakes, correct its and through that improve my knowledges.

Once again, I want to express my gratefulness to my mentor, Dr. Dung Le. He helped me to find the topic of my thesis and create the methods for me to resolve the problems and go to the end. Without his helps, I wouldn’t finish this project.

Ha Noi, summer 2019 Tran Van Ha

# Reference

STM32F4 DISCOVERY schematic at the link: https://[www.st.com/content/ccc/resource/technical/document/user\_manual/70/fe/4a/3f/e7/e1/4f/7d/DM000](http://www.st.com/content/ccc/resource/technical/document/user_manual/70/fe/4a/3f/e7/e1/4f/7d/DM000) 39084.pdf/files/DM00039084.pdf/jcr:content/translations/en.DM00039084.pdf

last access in May/20/2019.

STM32F4 DISCOVERY reference manual at the link:

https://[www.st.com/content/ccc/resource/technical/document/reference\_manual/3d/6d/5a/66/b4/99/40/d4/](http://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/) DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

last access in May/27/2019 I2S bus protocol at the link

[https://www.nxp.com/docs/en/user-guide/UM10204.pdf,](https://www.nxp.com/docs/en/user-guide/UM10204.pdf) last access in May/25/2019.

DMA basic at the link: <https://en.wikipedia.org/wiki/Direct_memory_access>, last access in May/15/2019

SPI/I2S protocol at the link: <https://en.wikipedia.org/wiki/Serial_Peripheral_Interface>, last access in May/18/2019.

Design of FAT file system at the link: <https://en.wikipedia.org/wiki/Design_of_the_FAT_file_system>, last access in May/15/2019.

WAV file format at the link: <https://en.wikipedia.org/wiki/WAV>, last access in May/10/2019.

Instruction about development kit, sensor datasheet and sample project from ST microcontroller at the link: [https://www.st.com/en/evaluation-tools/stm32f4discovery.html#resource,](https://www.st.com/en/evaluation-tools/stm32f4discovery.html#resource) last access in May/29/2019.

Component library source code at <https://github.com/fboris>.