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, Summer 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/thực 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 / /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/thực 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 …/…/2019

Người nhận xét

(Ký, ghi rõ họ tên)

ACKNOWLEDGEMENTS

I wish to express my heartfelt gratitude to my supervisor Dr. Dung Le, lecturer, School of Electronics & Telecommunication, Ha Noi university of Science and Technology, for his valuable support, guidance, and time throughout my project. I also appreciate the freedom provided to me by provided by Dr. Dung Le to explore new ideas in the field of my project.

I am also grateful to Mrs. Lan Bui, project manager and Mr. Dat Bui, technical lead of SDK project, FGA.DAP, FPT corporation for providing me with outstanding facilities in the corporation for my research. The time I worker at FPT, I had been taught many things, not only technical skills but also soft skills like English, communication with foreign partners, working process and embedded software architecture… All make me have a clear vision about the career path. It helps me to be confident about the path I chose, Embedded System Developer. Once more time, thank you a lot because of giving me chance to work in a good company, a fantastic team.

Finally, I want to thank my parents and the almighty god for their backing, without which this would not have been conceivable.

I commit that all result of this project is my own study under the instruction of lecturer. All data, result in project is definitely honest and hasn't been published by any research. All reference documents are listed detail as regulations.

Ha Tran Van

Ha Noi, Summer 2019

PROJECT ABSTRACT

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.

Contents

List of Figure

List of Table

Acronyms

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

CHAPTER I: PROJECT SUMMARY

1.1 Project summary

Project "Audio recorder and playback with STM32F4" is a project that apply the microcontroller knowledge to create a specific application.

In the age of 4.0 industry and technology revolution, the more human life is improved, the more applications are required such as smart home (automatic gate, watering system, auto-adjusted air conditioner with AI, voice control service, security camera…), smartphone, automotive system with high technology about safety, entertainment, control system. We can see in every technology application or equipment around us, all have a common thing. The most important element in charge of controlling all the rest parts. It's microcontroller.

Microcontroller is a very complicate system but also interesting to learn. With low cost, low energy consumption, powerful abilities, microcontroller appears in every equipment. It allows us to execute many functionalities such as collecting data under analog form from sensors, convert it to digital for specific aim, control the electric engine with Pulse Width Modulation, exchange data with communication modules as SPI, I2C, UART, delay and trigger with timer, network functionalities with CAN or Ethernet…

Understanding the important role of microcontroller in every embedded system, I worked on this project with the aim to enhance the knowledge about microcontroller architecture and embedded C programming.

1.2 Aims of research

As the above analysis, microcontroller plays a very important roles in any embedded system so understanding about microcontroller architecture is compulsory for any embedded system developer. In a real application project with many on-chip peripherals, developer can have precious experience about microcontroller and

There are many different architectures of microcontroller. Based on register length, we have 8 bits, 16 bits, 32 bits architecture. Based on core architecture, we have ARM, MIPS, PowerPC. Among these architectures, ARM 32 bit is the most popular due to the low power consumption (adapt with hand device), cheap price and ability to serve almost functionalities of embedded application in IoT, automotive, industry…

In this application, to execute all the functionalities, I determined some requirements of the system as below:

- Read and write data to SD card (FAT32 file system) via USB high speed in WAV format.
- Collect data in analog form (sound signal) via a MEMS microphone.
- Convert analog data to digital data to process.
- Convert digital data to analog to output the sound via speaker or headphone.
- Use DMA to enhance the data transfer speed.
- Control operation of MEMS microphone via I2C to transfer data to DAC in playback

1.3 Research methods

In order to program all these functionalities, I used some below methods:

- Read the reference manual of microcontroller to understand the hardware implement of required peripherals
- Read documents on internet to understand the communication protocols such as I2C, I2S.
- Read datasheet of used sensor to understand how it work and how to connect to microcontroller.
- Coding, loading code to development board and debugging.

1.4 Conclusion

In this chapter, I mentioned the reasons why I worked on this project and the overview of project, also the methods I use to finish this project. The detail theory of peripheral, development tools, I will describe in the next chapters.

CHAPTER II: UNDERSTANDING THE THEORY

In this chapter, I will present below theories

- FAT file system (FAT-32)
- Digital to Analog Converter (DAC)
- Direct Memory Access (DMA)
- Serial Peripheral Interface/ Inter-IC Sound (SPI/I2S)
- I2C protocol
- USB protocol
- WAV file format

2.1 FAT File system

2.1.1 History of FAT File system

A FAT file system is a specific type of computer file system architecture and a family of industry-standard file systems utilizing it.

The FAT file system is a legacy file system which is simple and robust. It offers good performance even in very light-weight implementations, but cannot deliver the same performance, reliability and scalability as some modern file systems. It is, however, supported for compatibility reasons by nearly all currently developed operating system for personal computers and many home computers, mobile devices and embedded systems, and thus is a compatible format for data exchange between computers and devices of almost any type and age from 1981 through the present.

Originally designed in 1977 for use on floppy disks, FAT was soon adapted and used almost universally on hard disks throughout the DOS and Windows 9x eras for two decades. Today, FAT file systems are still commonly found on floppy disks, USB sticks, flash and other solid-state memory cards and modules, and many portable and embedded devices. DCF implements FAT as the standard file system for digital cameras since 1998. FAT is also utilized for the EFI system partition (partition type 0xEF) in the boot stage of EFI-compliant computers.

2.1.2 Technical overview

The name of the file system originates from the file system's prominent usage of an index table, the File Allocation Table, statically allocated at the time of formatting. The table contains entries for each cluster, a contiguous area of disk storage. Each entry contains either the number of the next cluster in the file, or else a marker indicating end of file, unused disk space, or special reserved areas of the disk. The root directory of the disk contains the number of the first cluster of each file in that directory; the operating system can then traverse the FAT table, looking up the cluster number of each successive part of the disk file as a cluster chain until the end of the file is reached. In much the same way, sub-directories are implemented as special files containing the directory entries of their respective files.

Originally designed as an 8-bit file system, the maximum number of clusters has been significantly increased as disk drives have evolved, and so the number of bits used to identify each cluster has grown. The successive major versions of the FAT format are named after the

number of table element bits: 12 (FAT12), 16 (FAT16), and 32 (FAT32). Except for the original 8-bit FAT precursor, each of these variants is still in use. The FAT standard has also been expanded in other ways while generally preserving backward compatibility with existing software

2.1.3 FAT File system layout

Overview of the order of structures in a FAT partition or disk

Table 1 FAT file system structure

Region	Size in sectors	Contents
Reserved sectors	number of reserved sectors	Boot Sector
		FS Information Sector (FAT32 only)
		More reserved sectors (optional)
FAT Region	(number of FATs) * (sectors per FAT)	File Allocation Table #1
		File Allocation Table #2 ... (optional)
Root Directory Region	(number of root entries * 32) / (bytes per sector)	Root Directory (FAT12 and FAT16 only)
Data Region	(number of clusters) * (sectors per cluster)	Data Region (for files and directories) ... (to end of partition or disk)

Reserved sectors

The first reserved sector (logical sector 0) is the Boot Sector (also called Volume Boot Record or simply VBR). It includes an area called the BIOS Parameter Block (BPB) which contains some basic file system information, in particular its type and pointers to the location of the other sections, and usually contains the operating system's boot loader code.

Important information from the Boot Sector is accessible through an operating system structure called the Drive Parameter Block (DPB) in DOS and OS/2.

The total count of reserved sectors is indicated by a field inside the Boot Sector, and is usually 32 on FAT32 file systems.

For FAT32 file systems, the reserved sectors include a File System Information Sector at logical sector 1 and a Backup Boot Sector at logical sector 6.

While many other vendors have continued to utilize a single-sector setup (logical sector 0 only) for the bootstrap loader, Microsoft's boot sector code has grown to span over logical sectors 0 and 2 since the introduction of FAT32, with logical sector 0 depending on sub-routines in logical sector 2. The Backup Boot Sector area consists of three logical sectors 6, 7, and 8 as well. In some cases, Microsoft also uses sector 12 of the reserved sectors area for an extended boot loader.

FAT Region

This typically contains two copies of the File Allocation Table for the sake of redundancy checking, although rarely used, even by disk repair utilities.

These are maps of the Data Region, indicating which clusters are used by files and directories. In FAT12 and FAT16 they immediately follow the reserved sectors.

Typically, the extra copies are kept in tight synchronization on writes, and on reads they are only used when errors occur in the first FAT. In FAT32, it is possible to switch from the default behavior and select a single FAT out of the available ones to be used for diagnosis purposes.

The first two clusters (cluster 0 and 1) in the map contain special values.

Root Directory Region

This is a Directory Table that stores information about the files and directories located in the root directory. It is only used with FAT12 and FAT16, and imposes on the root directory a fixed maximum size which is pre-allocated at creation of this volume. FAT32 stores the root directory in the Data Region, along with files and other directories, allowing it to grow without such a constraint. Thus, for FAT32, the Data Region starts here.

Data Region

This is where the actual file and directory data is stored and takes up most of the partition. Traditionally, the unused parts of the data region are initialized with a filler value of 0xF6 as per the INT 1Eh's Disk Parameter Table (DPT) during format on IBM compatible machines, but also used on the Atari Portfolio. 8-inch CP/M floppies typically came pre-formatted with a value of 0xE5 by way of Digital Research this value was also used on Atari ST formatted

floppies. Amstrad used 0xF4 instead. Some modern formatters wipe hard disks with a value of 0x00, whereas a value of 0xFF, the default value of a non-programmed flash block, is used on flash disks to reduce wear. The latter value is typically also used on ROM disks. (Some advanced formatting tools allow to configure the format filler byte.)

The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the FAT. Note however, that files are allocated in units of clusters, so if a 1 KB file resides in a 32 KB cluster, 31 KB are wasted.

FAT32 typically commences the Root Directory Table in cluster number 2: the first cluster of the Data Region.

FAT uses little-endian format for all entries in the header (except for, where explicitly mentioned, for some entries on Atari ST boot sectors) and the FAT(s). It is possible to allocate more FAT sectors than necessary for the number of clusters. The end of the last sector of each FAT copy can be unused if there are no corresponding clusters. The total number of sectors (as noted in the boot record) can be larger than the number of sectors used by data (clusters × sectors per cluster), FATs (number of FATs × sectors per FAT), the root directory (n/a for FAT32), and hidden sectors including the boot sector: this would result in unused sectors at the end of the volume. If a partition contains more sectors than the total number of sectors occupied by the file system it would also result in unused sectors, at the end of the partition, after the volume.

2.2 Digital to Analog Converter

In electronics, a digital-to-analog converter (DAC, D/A, D2A, or D-to-A) is a system that converts a digital signal into an analog signal. An analog-to-digital converter (ADC) performs the reverse function.

There are several DAC architectures; the suitability of a DAC for a particular application is determined by figures of merit including: resolution, maximum sampling frequency and others. Digital-to-analog conversion can degrade a signal, so a DAC should be specified that has insignificant errors in terms of the application.

DACs are commonly used in music players to convert digital data streams into analog audio signals. They are also used in televisions and mobile phones to convert digital video data into analog video signals which connect to the screen drivers to display monochrome or color

images. These two applications use DACs at opposite ends of the frequency/resolution trade-off. The audio DAC is a low-frequency, high-resolution type while the video DAC is a high-frequency low- to medium-resolution type.

Due to the complexity and the need for precisely matched components, all but the most specialized DACs are implemented as integrated circuits (ICs). Discrete DACs would typically be extremely high speed, low resolution, power hungry types, as used in military radar systems. Very high-speed test equipment, especially sampling oscilloscopes, may also use discrete DACs.

A DAC converts an abstract finite-precision number (usually a fixed-point binary number) into a physical quantity (e.g., a voltage or a pressure). In particular, DACs are often used to convert finite-precision time series data to a continually varying physical signal.

An ideal DAC converts the abstract numbers into a conceptual sequence of impulses that are then processed by a reconstruction filter using some form of interpolation to fill in data between the impulses. A conventional practical DAC converts the numbers into a piecewise constant function made up of a sequence of rectangular functions that is modeled with the zero-order hold. Other DAC methods (such as those based on delta-sigma modulation) produce a pulse-density modulated output that can be similarly filtered to produce a smoothly varying signal.

As per the Nyquist–Shannon sampling theory, a DAC can reconstruct the original signal from the sampled data provided that its bandwidth meets certain requirements (e.g., a baseband signal with bandwidth less than the Nyquist frequency). Digital sampling introduces quantization error that manifests as low-level noise in the reconstructed signal.

2.2.1 DAC peripheral on STM32F407VG MCU

The DAC module is a 12-bit, voltage output digital-to-analog converter. The DAC can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. In 12-bit mode, the data could be left- or right-aligned. The DAC has two output channels, each with its own converter. In dual DAC channel mode, conversions could be done independently or simultaneously when both channels are grouped together for synchronous update operations. An input reference pin, VREF+ (shared with ADC) is available for better resolution.

DAC main features:

• Two DAC converters: one output channel each

• Left or right data alignment in 12-bit mode

• Synchronized update capability

• Noise-wave generation

• Triangular-wave generation

• Dual DAC channel for independent or simultaneous conversions

• DMA capability for each channel

• DMA underrun error detection

• External triggers for conversion

• Input voltage reference, VREF+

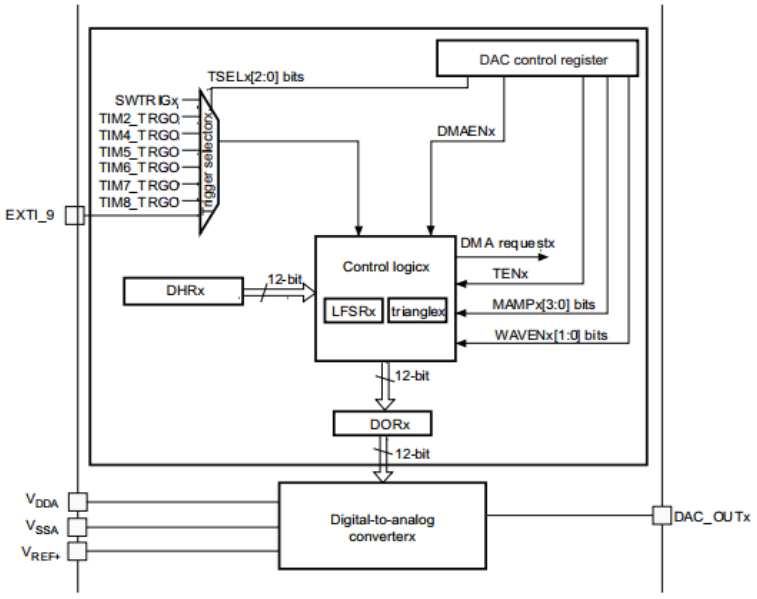

Figure 1 DAC channel block diagram

2.2.2 DAC functional description

DAC main register:

DAC control register:

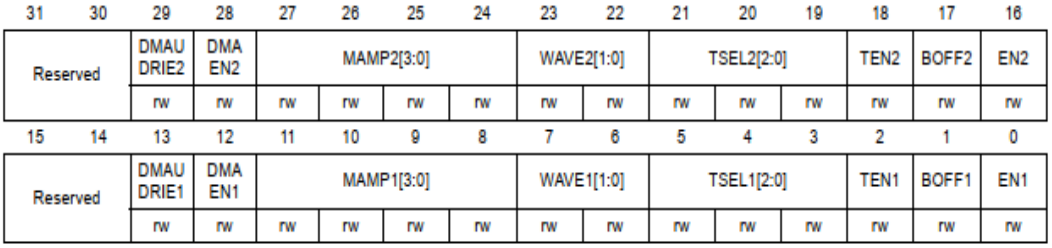

Figure 2 DAC control register bit map

This register contains bits to configure almost features of DAC such as module enable trigger sources, DMA enable, DMA underrun interrupt enable, mask/amplitude selector.

DAC data holding register:

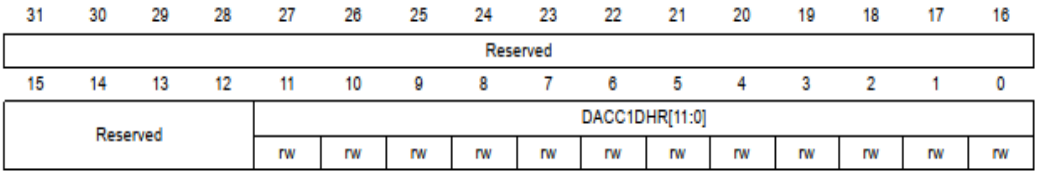

Figure 3 DAC channel 1 12-bit right-aligned data holding register

Based on resolution and alignment, DAC module supports some data holding register includes: DAC channel1 12-bit right-aligned data holding register, DAC channel1 12-bit left-aligned data holding register, DAC channel1 8-bit right-aligned data holding register … These registers contain digital data. After one AHB1 clock cycle, data stored in these registers is forwarded to data output registers before having processed with the digital to analog engine.

DAC data output register:

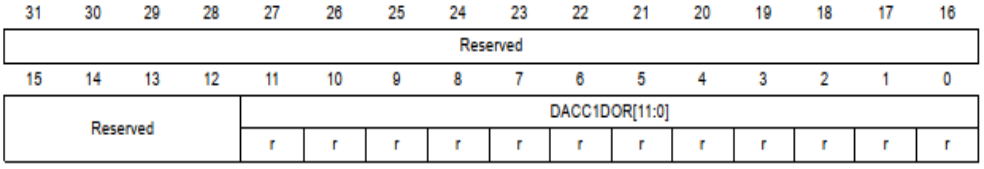

Figure 4 DAC channel 1 data output register

Each DAC channel contains a data output register to store digital data. These data will be processed with digital to analog engine to have analog signal transferred to DAC out put pin.

DAC status register (DAC_SR)

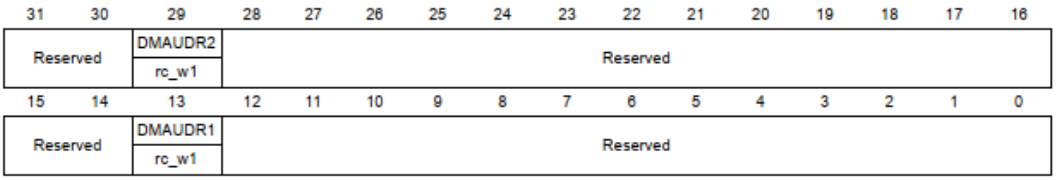

Figure 5 DAC status register

This register contains two DMA flags of two DAC channels to help developers to manage the operation of DAC module.

2.2.3 DAC operation summary

To convert digital data to analog data with DAC module:

- Initialize input, output, ground, reference voltage for DAC
- Select trigger source for module: internal timer, external pin or software trigger
- Configure parameters (DMA request, triangle pulse generation, DMA underrun interrupt enable, mask/amplitude selector …) to determine the operation mode of DAC module.
- Enable DMA channel ISR to manage the operation of DAC module

2.3 Direct Memory Access (DMA)

Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory (random-access memory), independent of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller (DMAC) when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory

without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O Acceleration Technology. DMA is of interest in network-on-chip and in-memory computing architectures.

2.3.1 DMA principles

Standard DMA, also called third-party DMA, uses a DMA controller. A DMA controller can generate memory addresses and initiate memory read or write cycles. It contains several hardware registers that can be written and read by the CPU. These include a memory address register, a byte count register, and one or more control registers. Depending on what features the DMA controller provides, these control registers might specify some combination of the source, the destination, the direction of the transfer (reading from the I/O device or writing to the I/O device), the size of the transfer unit, and/or the number of bytes to transfer in one burst.

To carry out an input, output or memory-to-memory operation, the host processor initializes the DMA controller with a count of the number of words to transfer, and the memory address to use. The CPU then commands peripheral device to initiate data transfer. The DMA controller then provides addresses and read/write control lines to the system memory. Each time a byte of data is ready to be transferred between the peripheral device and memory, the DMA controller increments its internal address register until the full block of data is transferred.

2.3.2 Bus mastering

In a bus mastering system, also known as a first-party DMA system, the CPU and peripherals can each be granted control of the memory bus. Where a peripheral can become bus master, it can directly write to system memory without involvement of the CPU, providing memory address and control signals as required. Some measure must be provided to put the processor into a hold condition so that bus contention does not occur.

2.3.3 Transfer types

DMA transfers can transfer either one byte at a time or all at once in burst mode. If they transfer a byte at a time, this can allow the CPU to access memory on alternating bus cycles – this is called cycle stealing since the CPU and either the DMA controller or the bus master contend

for memory access. In burst mode DMA, the CPU can be put on hold while the DMA transfer occurs and a full block of possibly hundreds or thousands of bytes can be moved. When memory cycles are much faster than processor cycles, an interleaved DMA cycle is possible, where the DMA controller uses memory while the CPU cannot.

2.3.4 DMA operation modes

Burst mode:

In burst mode, an entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode".

Cycle stealing mode:

The cycle stealing mode is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using BR (Bus Request) and BG (Bus Grant) signals, which are the two signals controlling the interface between the CPU and the DMA controller. However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is de-asserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers. The CPU processes an instruction, then the DMA controller transfers one data value, and so on. On the one hand, the data block is not transferred as quickly in cycle stealing mode as in burst mode, but on the other hand the CPU is not idled for as long as in burst mode. Cycle stealing mode is useful for controllers that monitor data in real time.

Transparent mode:

Transparent mode takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. In transparent mode, the DMA controller transfers data only when the CPU is performing operations that do not use the system buses. The primary advantage of transparent mode is that the CPU never stops executing its programs

and the DMA transfer is free in terms of time, while the disadvantage is that the hardware needs to determine when the CPU is not using the system buses, which can be complex.

2.3.5 DMA module intergraded in STM32F407VG

DMA main features:

• Dual AHB master bus architecture, one dedicated to memory accesses and one

dedicated to peripheral accesses

• AHB slave programming interface supporting only 32-bit accesses

• 8 streams for each DMA controller, up to 8 channels (requests) per stream

• Four-word depth 32 first-in, first-out memory buffers (FIFOs) per stream, that can be

used in FIFO mode or direct mode:

– FIFO mode: with threshold level software selectable between 1/4, 1/2 or 3/4 of the

FIFO size

– Direct mode

Each DMA request immediately initiates a transfer from/to the memory. When it is configured in direct mode (FIFO disabled), to transfer data in memory-to-peripheral mode, the DMA preloads only 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.

• Each stream can be configured by hardware to be:

– a regular channel that supports peripheral-to-memory, memory-to-peripheral and memory-to-memory transfers

– a double buffer channel that also supports double buffering on the memory side

• Each of the 8 streams are connected to dedicated hardware DMA channels (requests)

• Priorities between DMA stream requests are software-programmable (4 levels

consisting of very high, high, medium, low) or hardware in case of equality (request 0 has priority over request 1, etc.)

• Each stream also supports software trigger for memory-to-memory transfers (only available for the DMA2 controller)

• Each stream request can be selected among up to 8 possible channel requests. This selection is software-configurable and allows several peripherals to initiate DMA

requests

• The number of data items to be transferred can be managed either by the DMA controller or by the peripheral:

–DMA flow controller: the number of data items to be transferred is software programmable from 1 to 65535

– Peripheral flow controller: the number of data items to be transferred is unknown and controlled by the source or the destination peripheral that signals the end of the transfer by hardware

• Independent source and destination transfer width (byte, half-word, word): when the data widths of the source and destination are not equal, the DMA automatically packs/unpacks the necessary transfers to optimize the bandwidth. This feature is only available in FIFO mode

• Incrementing or non-incrementing addressing for source and destination

• Supports incremental burst transfers of 4, 8 or 16 beats. The size of the burst is software-configurable, usually equal to half the FIFO size of the peripheral

• Each stream supports circular buffer management

• 5 event flags (DMA Half Transfer, DMA Transfer complete, DMA Transfer Error, DMA FIFO Error, Direct Mode Error) logically together in a single interrupt request for each stream

2.3.6 DMA functional description

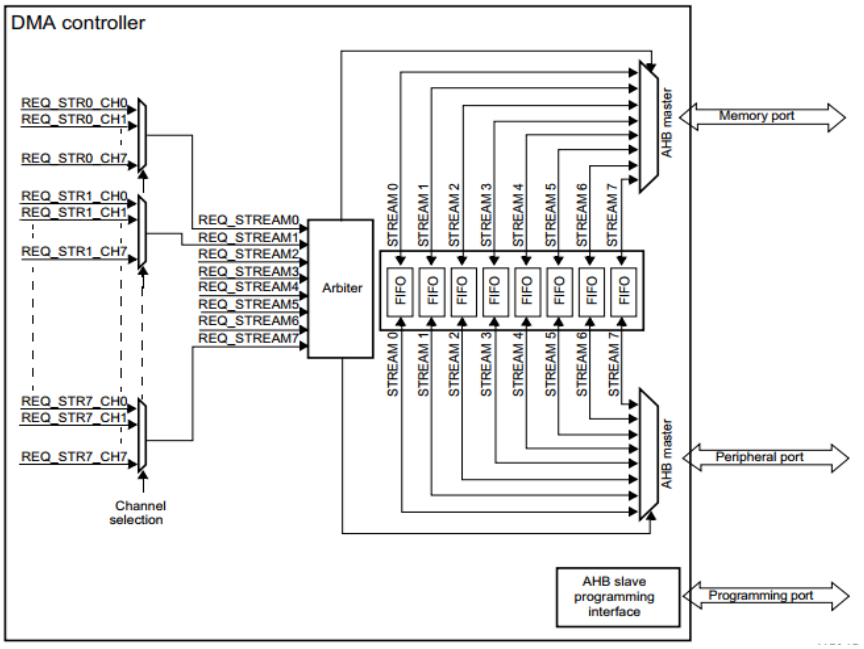

Figure 6 DMA block diagram

The DMA controller performs direct memory transfer: as an AHB master, it can take the control of the AHB bus matrix to initiate AHB transactions.

It can carry out the following transactions:

• peripheral-to-memory

• memory-to-peripheral

• memory-to-memory

The DMA controller provides two AHB master ports: the AHB memory port, intended to be connected to memories and the AHB peripheral port, intended to be connected to peripherals. However, to allow memory-to-memory transfers, the AHB peripheral port must also have access to the memories.

The AHB slave port is used to program the DMA controller (it supports only 32-bit accesses)

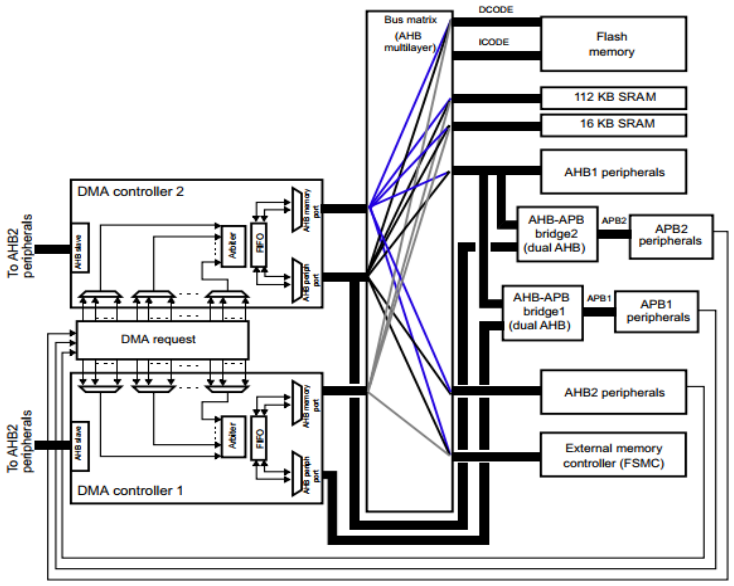

Figure 7 System implementation of the two DMA controllers on STM32F407

DMA transactions

A DMA transaction consists of a sequence of a given number of data transfers. The number of data items to be transferred and their width (8-bit, 16-bit or 32-bit) are software programmable.

Each DMA transfer consists of three operations:

• A loading from the peripheral data register or a location in memory

• A storage of the data loaded to the peripheral data register or a location in

• A post-decrement of the DMA counter register, which contains the number of transactions that still have to be performed

After an event, the peripheral sends a request signal to the DMA controller. The DMA controller serves the request depending on the channel priorities. As soon as the DMA controller accesses the peripheral, an Acknowledge signal is sent to the peripheral by the DMA controller. The peripheral releases its request as soon as it gets the Acknowledge signal from the DMA controller. Once the request has been de-asserted by the peripheral, the DMA controller releases the Acknowledge signal. If there are more requests, the peripheral can initiate the next transaction.

Channel selection

Each stream is associated with a DMA request that can be selected out of 8 possible channel requests.

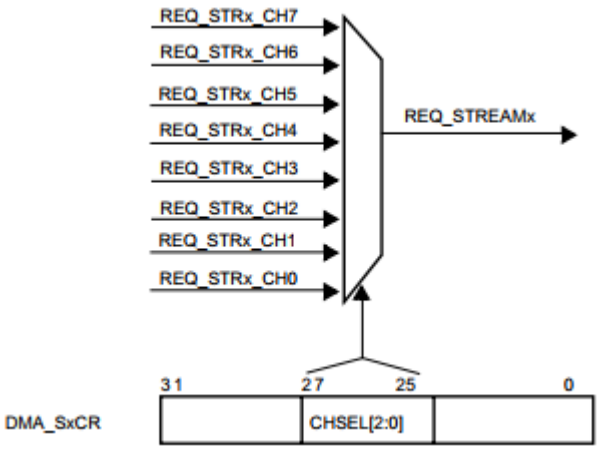

Figure 8 DMA channel selection

The 8 requests from the peripherals (TIM, ADC, SPI, I2C, etc.) are independently connected to each channel and their connection depends on the product implementation

Two figures below show the connection between DMA channels with peripherals

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	-	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	-	SPI3_TX
Channel 1	I2C1_RX	-	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX(1)	UART7_TX(1)	TIM3_CH4 TIM3_UP	UART7_RX(1)	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX(1)	TIM3_CH3

Figure 9 DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A(1)	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A(1)	ADC1	SAI1_B(1)	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B(1)	SPI6_TX(1)	SPI6_RX(1)	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX(1)	SPI5_TX(1)	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	-	SPI1_TX	-	-
Channel 4	SPI4_RX(1)	SPI4_TX(1)	USART1_RX	SDIO	-	USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX(1)	SPI4_TX(1)	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX(1)	SPI5_TX(1)	TIM8_CH4 TIM8_TRIG TIM8_COM

Figure 10 DMA2 request mapping

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

Bits DIR[1:0] of the DMA_SxCR register	Direction	Source address	Destination address
00	Peripheral-to-memory	DMA_SxPAR	DMA_SxM0AR
01	Memory-to-peripheral	DMA_SxM0AR	DMA_SxPAR
10	Memory-to-memory	DMA_SxPAR	DMA_SxM0AR
11	reserved	-	-

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

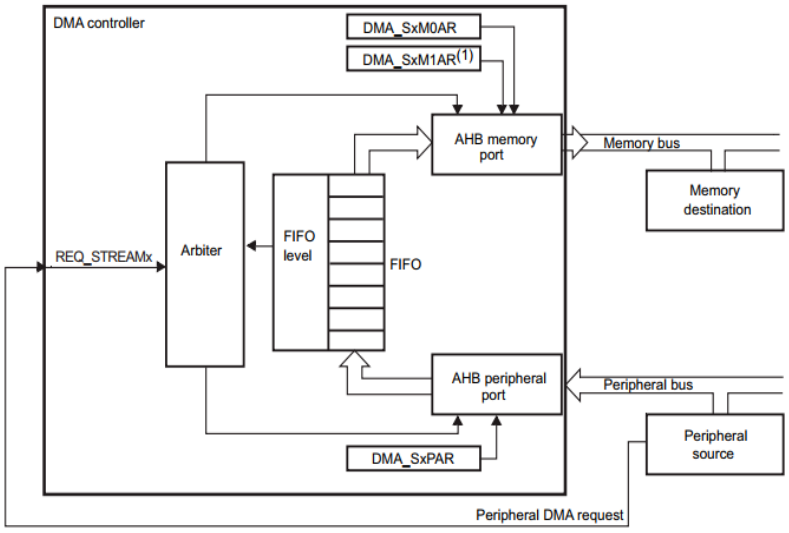

Figure 12 Peripheral-to-memory mode

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

Bits DIR[1:0] of the DMA_SxCR register	Direction	Source address	Destination address
00	Peripheral-to-memory	DMA_SxPAR	DMA_SxM0AR / DMA_SxM1AR
01	Memory-to-peripheral	DMA_SxM0AR / DMA_SxM1AR	DMA_SxPAR
10	Not allowed(1)		
11	Reserved	-	-

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:

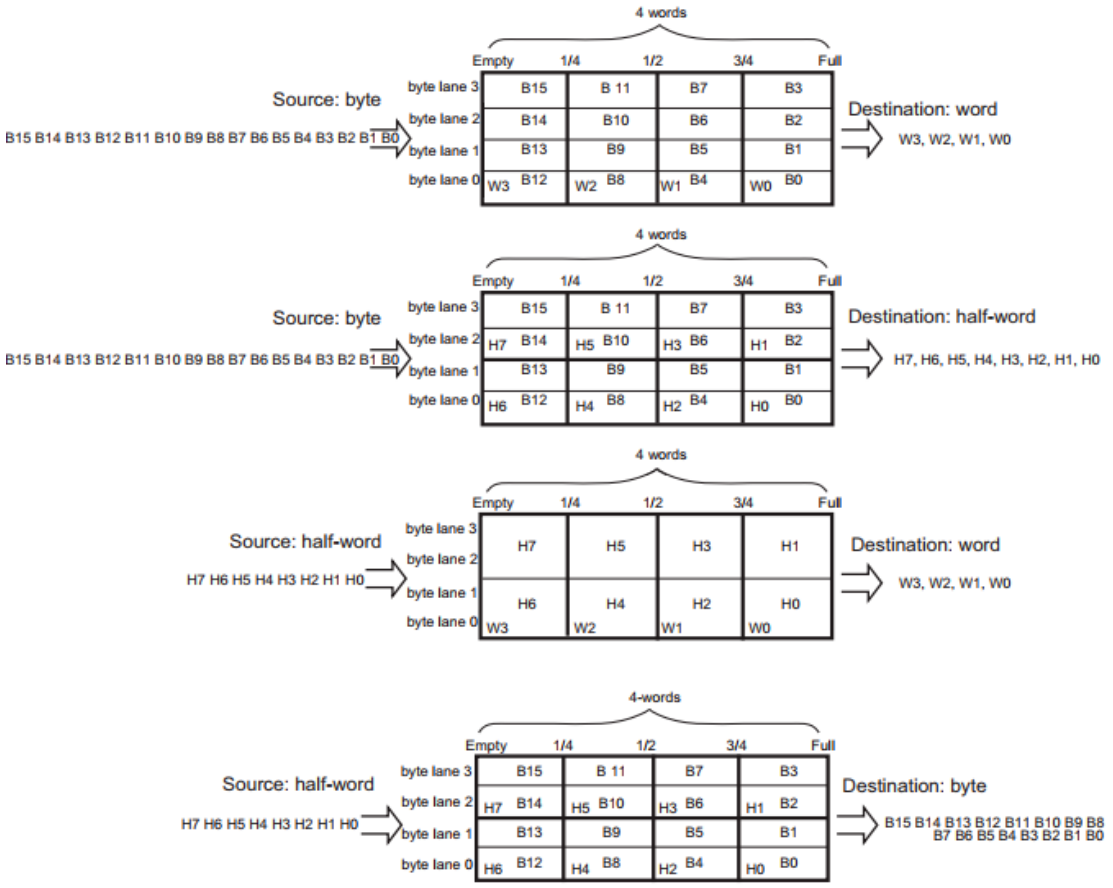

Figure 14 DMA FIFO structure

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.

5. Select the DMA channel (request).

6. If the peripheral is intended to be the flow controller and if it supports this feature.

7. Configure the stream priority.

8. Configure the FIFO usage (enable or disable, threshold in transmission and reception)

9. 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.

10. 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.

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

2.4.1 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.

2.4.2 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.

2.4.3 SPI functional description

General description

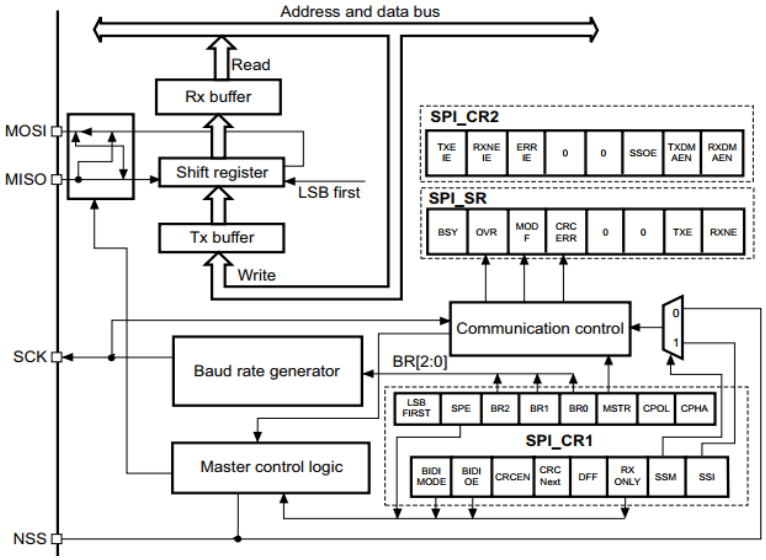

Figure 15 The block diagram of the SPI

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

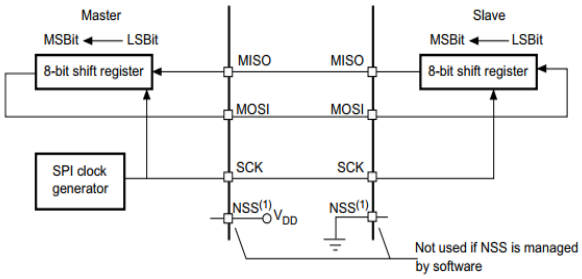

Figure 16 Single master/ single slave application

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.

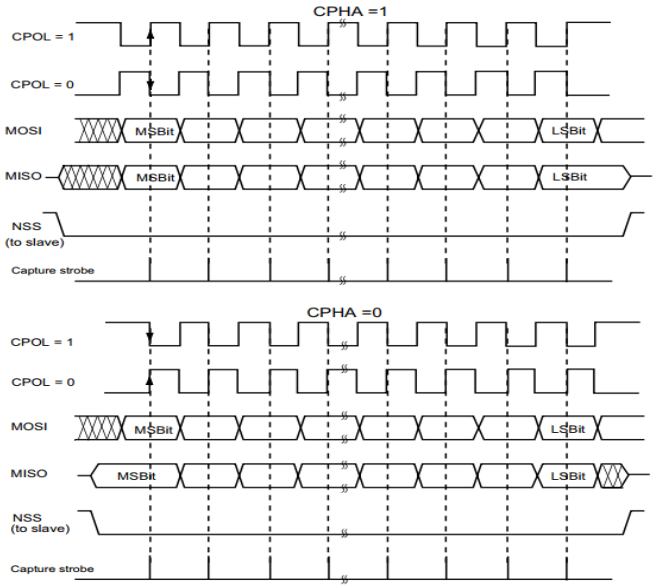

Figure 17 Data clock timing diagram

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.

2.4.4 I2S functional description

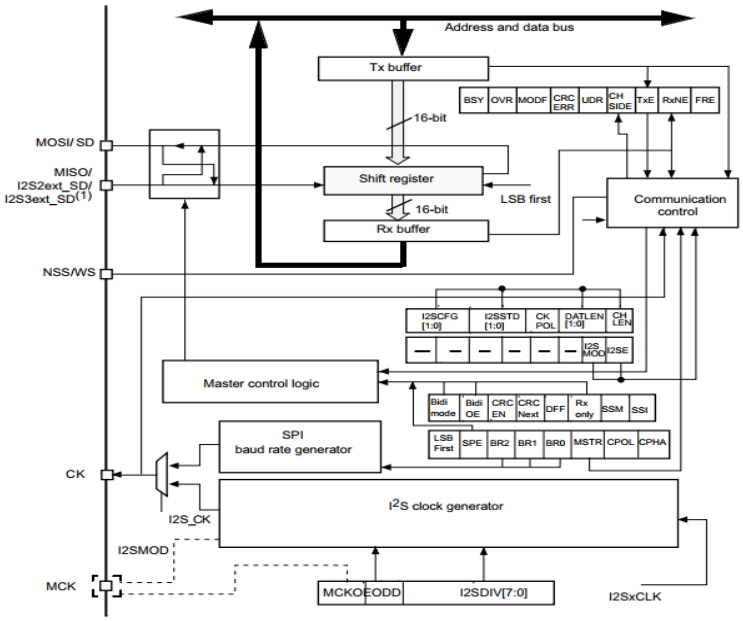

Figure 18 I2S block diagram

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.

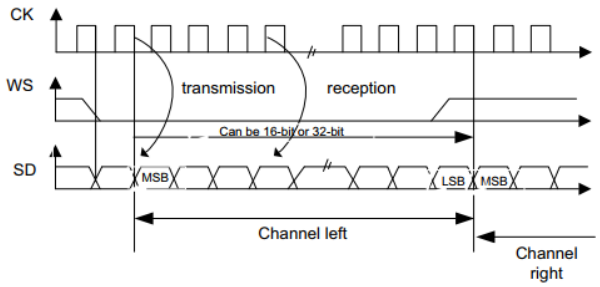

Figure 19 I2S Philips protocol waveforms

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).

2.5 I2C protocol

2.5.1 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.

2.5.2 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.

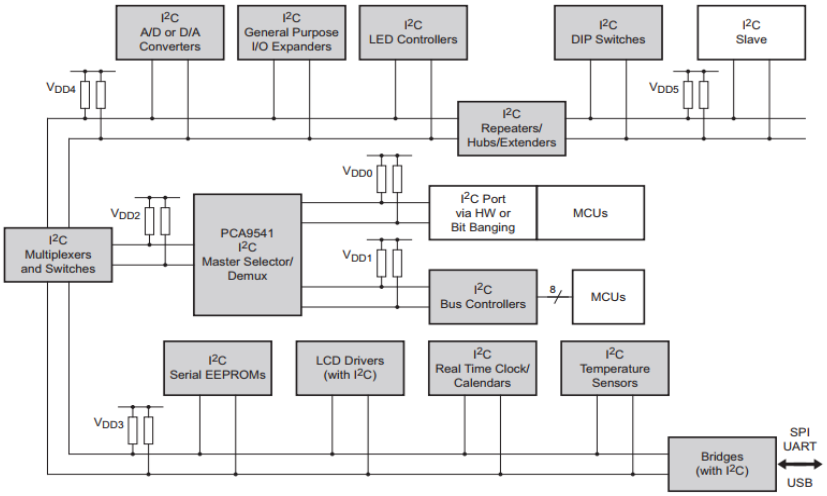

Figure 20 Example of I2C-bus applications

2.5.3 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

2.5.4 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.

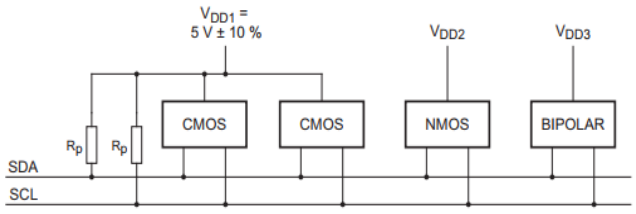

Figure 21 Devices with various supply voltages sharing the same bus

Data validity

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.

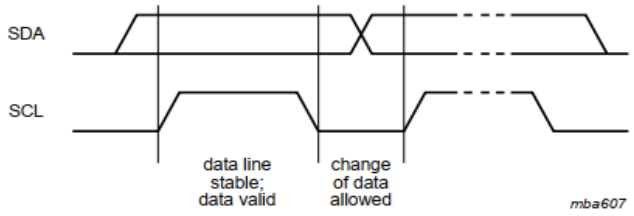

Figure 22 Bit transfer on the I2C-bus

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.

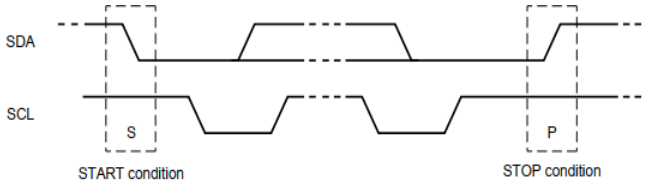

Figure 23 START and STOP conditions

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.

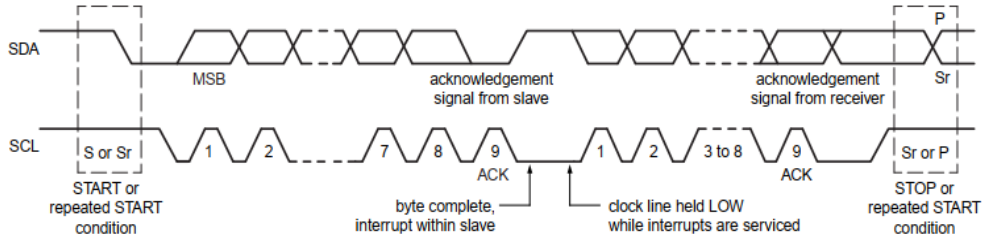

Figure 24 Data transfer on the I2C-bus

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.

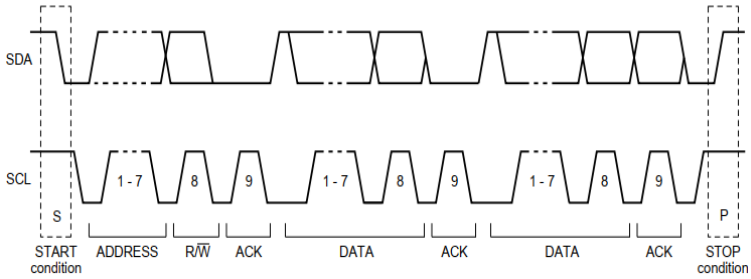

Figure 25 A complete data transmission

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).

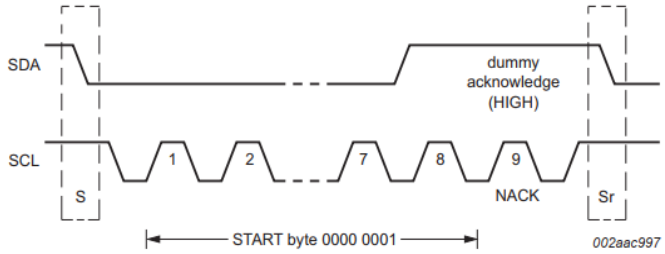

Figure 26 START byte procedure

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

2.6 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.

2.6.1 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.

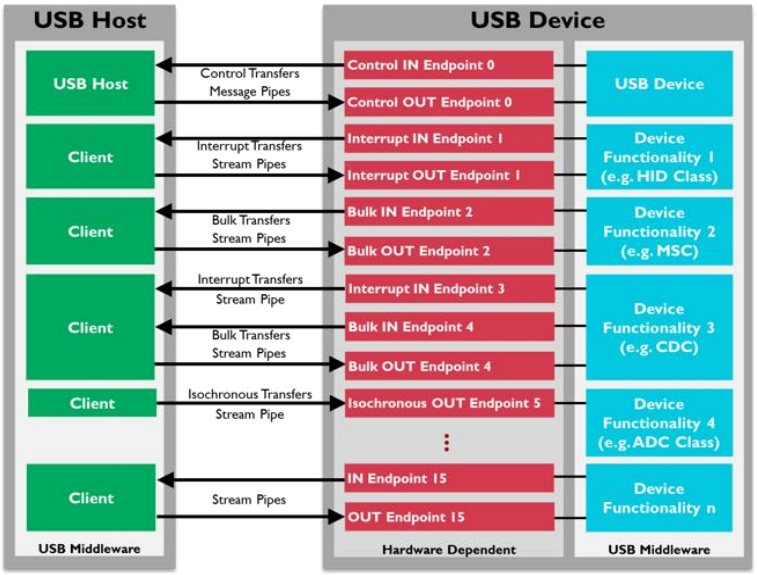

Figure 28 Logical Connections between USB Host Clients and USB Device Endpoints

2.6.2 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.

2.7 WAV File Format

2.7.1 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.

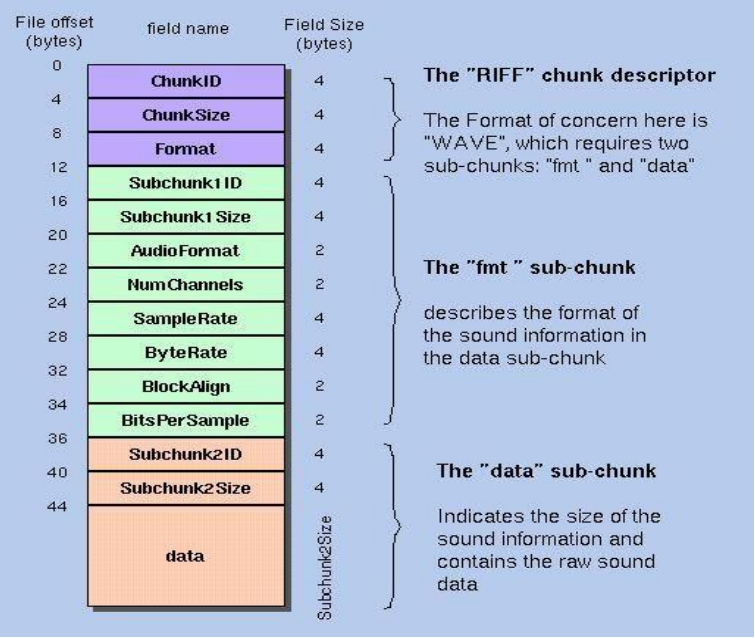

Figure 29 WAV File Format Layout

2.7.2 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)

2.7.3 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.

2.7.4 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

3.1 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

3.1.1 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

3.1.2 Hardware structure

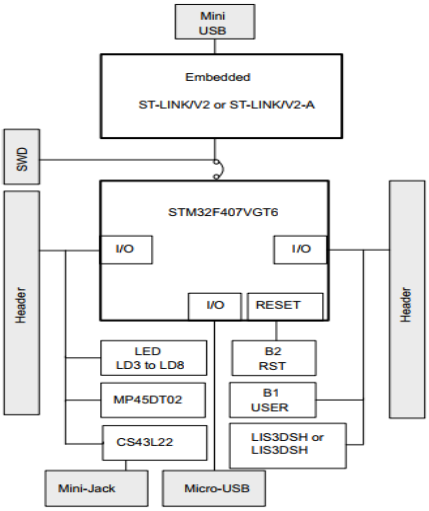

Figure 30 Hardware block diagram

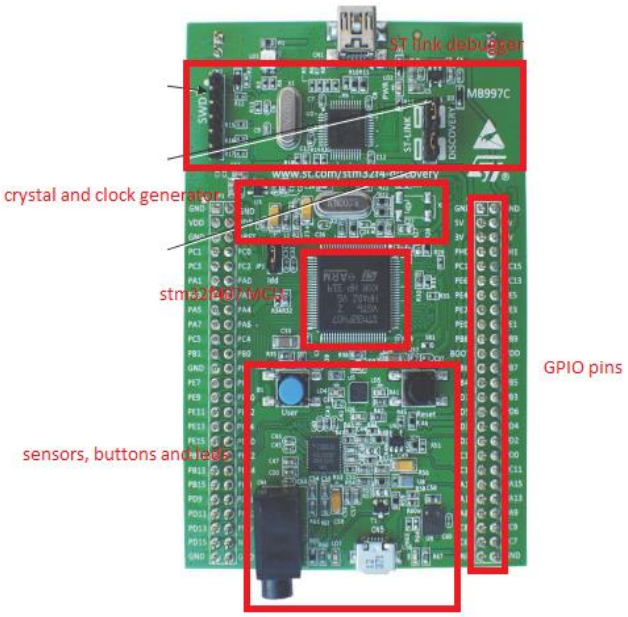

Figure 31 STM32F4 DISCOVERY board

3.1.3 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…

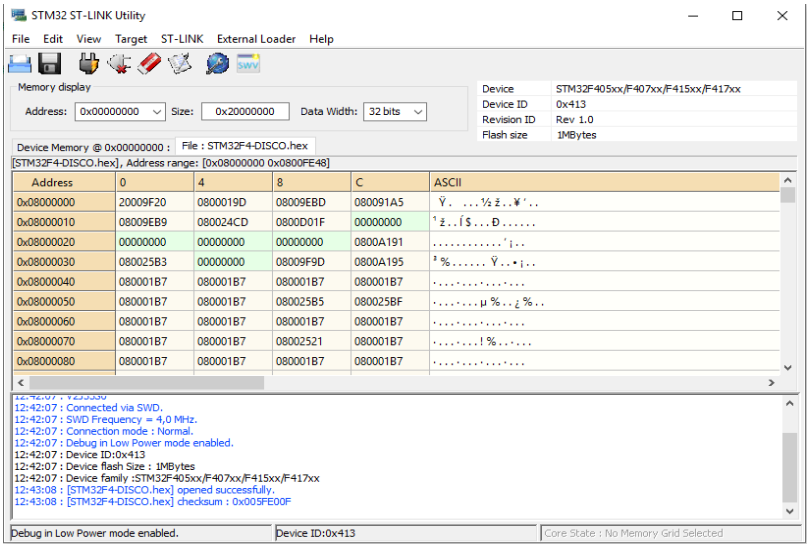

Figure 32 STM32 ST Link interface

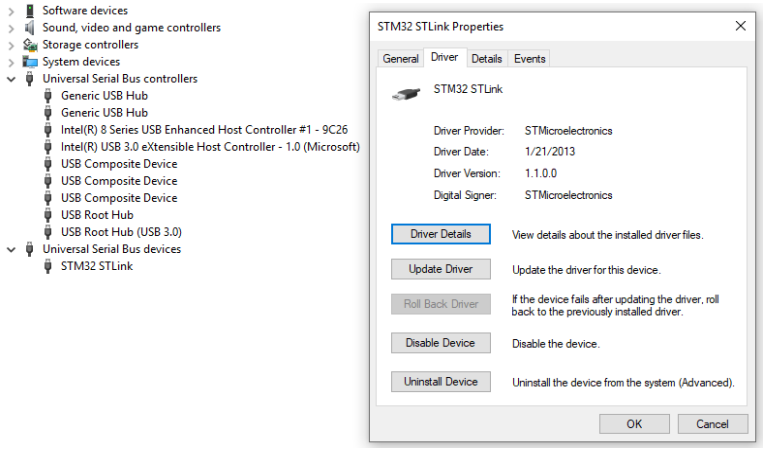

Figure 33 Driver update for STM board

3.1.3 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

3.2 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.

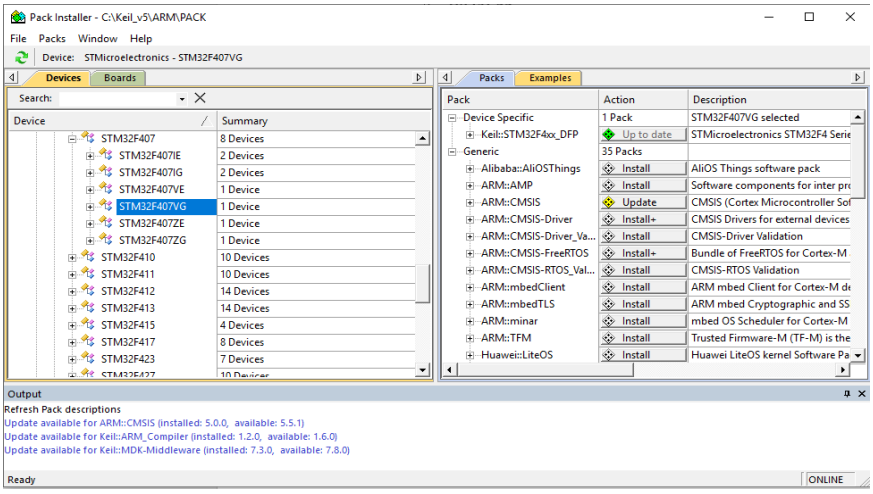

Figure 34 Keil C pack installer

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.

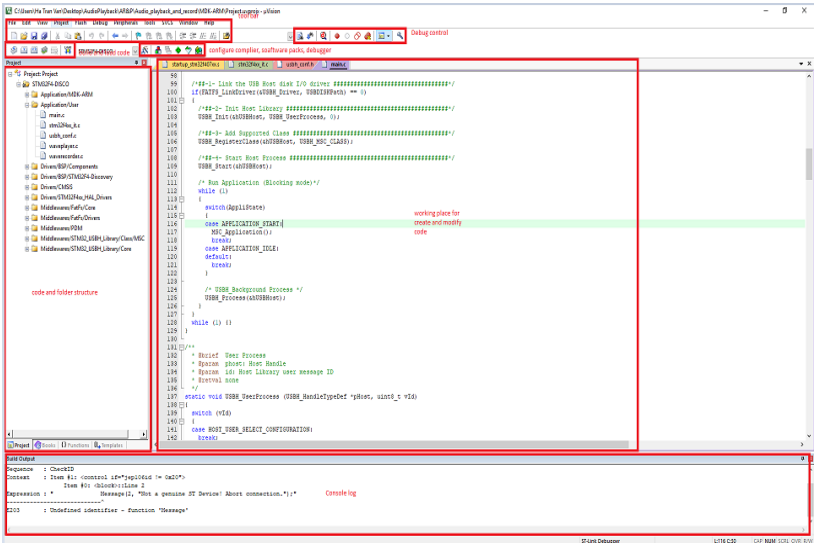

Figure 35 Keil C IDE interface

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.

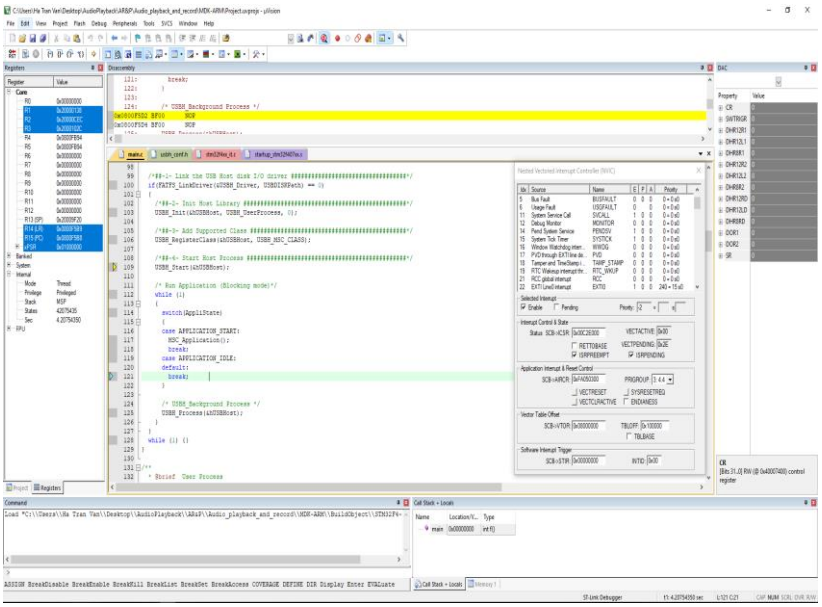

Figure 36 Keil C Debug Interface

3.3 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

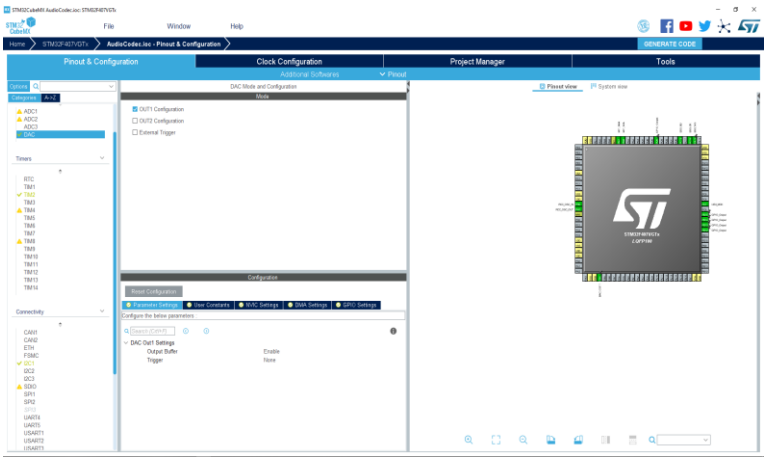

Figure 37 ST Cube Mx GUI

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

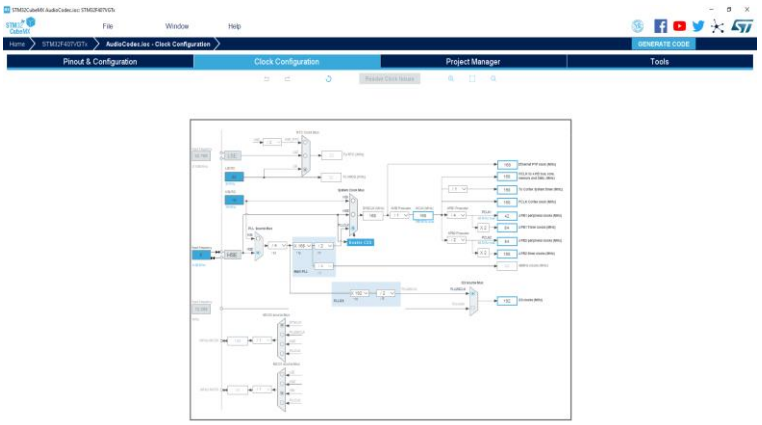

Figure 38 Clock configuration with STM32 Cube Mx

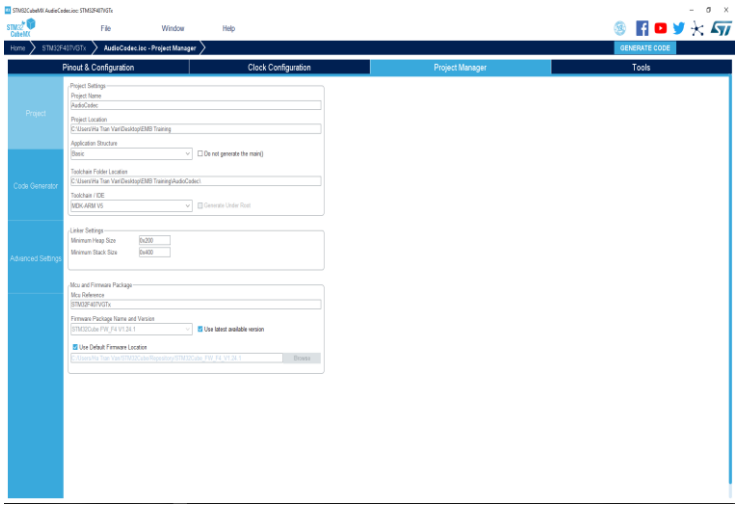

Figure 39 Project management with ST Cube Mx

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

4.1 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.

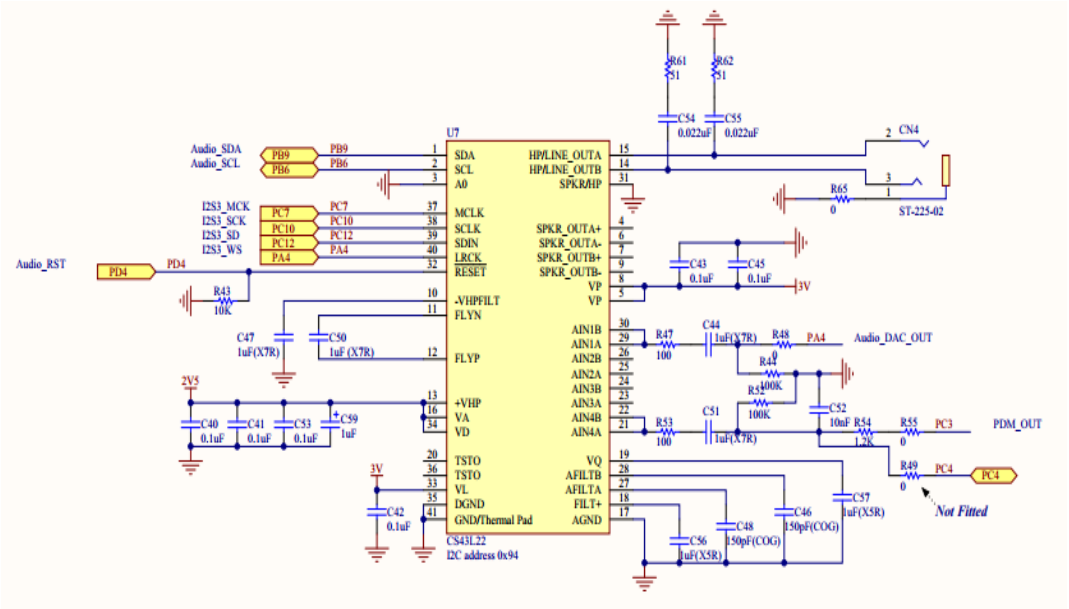

Figure 40 Schematic of Audio peripherals connection on STM32F4 discovery

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).

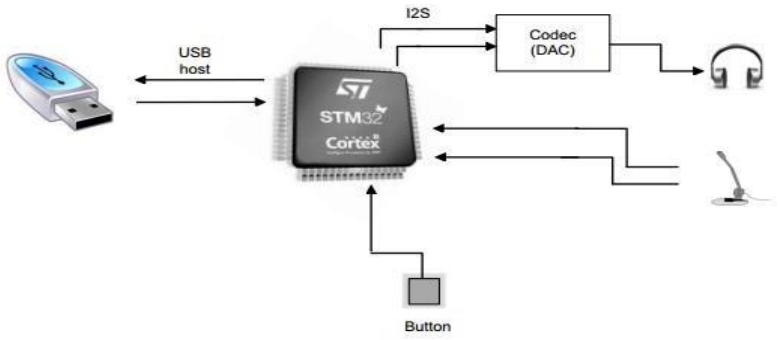

Figure 41 Audio playback and record architecture

4.2 Audio Playback Application

Procedure for the audio play application on the board

1.Initialize the USB, Fat Fs File Systems, Audio DAC, MEMS

2.Transfer the WAV file from USB storage to internal SRAM of MCU, block by block (1024 bytes) using DMA in its first buffer.

3.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.

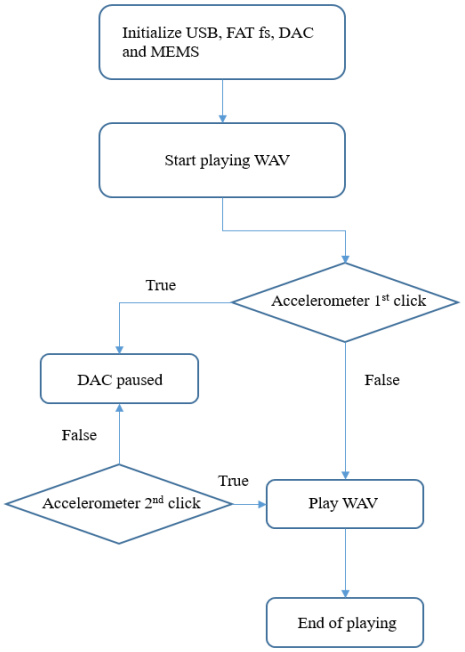

Figure 42 Audio playback application flow chart

4.3 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.

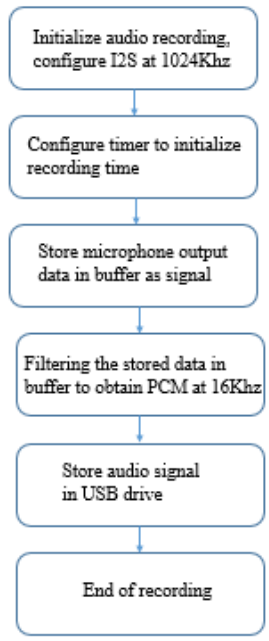

Figure 43 Audio recording flow chart

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).

4.4 Volume control

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

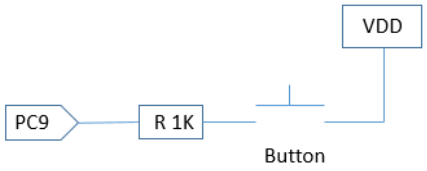

Figure 44 Setup MCU pin for external trigger

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.

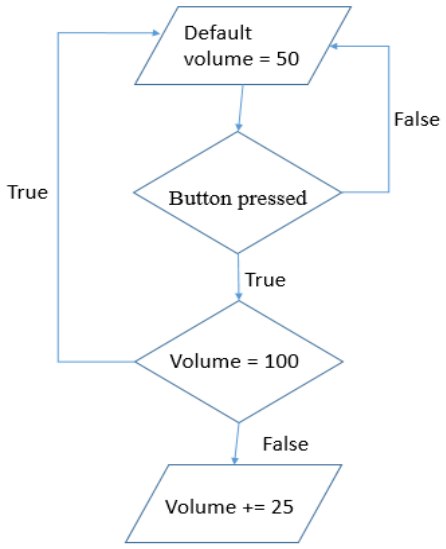

Figure 45 Volume control diagram

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/DM00039084.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/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, 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, last access in May/29/2019.

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