BTL

Môn Lập trình nhúng cơ bản

Đề tài : Lập trình đọc và phát nhạc MP3 trên vi điều khiển STM32F4

Nhóm 1: Bùi Thị Kim Anh
Phạm Văn Nhất
Nguyễn Tuấn Vinh
Lớp: KTPM CLC 2 – K6
GV hướng dẫn Ts Phạm Văn Hà

Lời mở đầu

Trên thế giới, lập trình nhúng đã hình thành và phát triển từ rất sớm nhưng đối với Việt Nam thì sự phát triển của hệ thống nhúng và lập trình nhúng mới được để ý tời gần đây. Hiện naycác quá trình sản xuất và quản lí như: các hệ thống đo lường điều khiển tự động trong sản xuất công nghiệp; các hệ thống di động và không dây tiên tiến, các hệ thống thông tin vệ tinh, các hệ thống thông tin dựa trên Web, chính phủ điện tử, thương mại điện tử, các cơ sở dữ liệu của nhiều ngành kinh tế và của Quốc gia, các hệ thống thiết bị Y tế hiện đại, các thiết bị điện tử dân dụng, ... đều là sản phẩm của sự kết hợp giữa các lĩnh vực khoa học trên. Và các lĩnh vực trên đều có một điểm chung đó là đều sử dụng hệ thống nhúng.Thật vậy, hệ thống nhúng đã và đang xuất hiện trên khắp các lĩnh vực của cuộc sống.

Do vậy, trong bài tập lớn này, nhóm em sẽ giới thiệu một cách tổng quan về hệ thống nhúng và một vài công nghệ được sử dụng trong chip STM32F407VG. Đồng thời giới thiệu về chương trình phát nhạc MP3.

Trong quá trình hoàn thành bài tập lớn, nhóm đã cố gắng hết sức có thể để hoàn thiện tốt nhất sản phẩm của mình. Hi vọng thầy có thể có những lời nhận xét để nhóm có thể hoàn thành tốt nhất bài tập lớn này. Chúng em xin chân thành cảm ơn thầy!

Mục lục

Chương 1. Tổng quan về lập trình nhúng

1.1.Định nghĩa hệ thống nhúng

Hệ thống nhúng (embedded system) được định nghĩa là một hệ thống chuyên dụng, thường có khả năng tự hành và được thiết kế tích hợp vào một hệ thống lớn hơn để thực hiện một chức năng chuyên biệt nào đó. Khác với các máy tính đa chức năng (multi-purposes computers), ví dụ như máy vi tính cá nhân (PC), một hệ thống nhúng thường chỉ thực hiện một hoặc một vài chức năng nhất định.

Hệ thống nhúng bao gồm cả thiết bị phần cứng và phần mềm, hầu hết đều phải thỏa mãn yêu cầu hoạt động theo thời gian thực (real-time). Tùy theo tính chất và yêu cầu, mức độ đáp ứng của hệ thống có thể phải là rất nhanh (ví dụ như hệ thống thắng trong xe hơi hoặc điều khiển thiết bị trong nhà máy), hoặc có thể chấp nhận một mức độ chậm trễ tương đối (ví dụ như điện thoại di động, máy lạnh, ti-vi). Để có thể dễ hình dung, ta xem ví dụ sau đây: một chiếc xe hơi trung bình có khoảng 70-80 chip vi xử lý (micro controller unit), mỗi bộ vi xử lý đảm nhiệm một nhiệm vụ, chẳng hạn như đóng mở cửa, điều khiển đèn tín hiệu, đo nhiệt độ trong/ngoài xe, hiển thị giao diện người dùng (dashboard), điều khiển thắng (nếu dùng hệ thống thắng điện)… Mỗi bộ phận như thế là một hệ thống nhúng, tất cả được thiết kế tích hợp vào một hệ thống chung lớn hơn, chính là chiếc xe hơi. Một ví dụ khác gần gũi hơn với cuộc sống hằng ngày, đó là những chiếc điện thoại di động. Các chức năng như điều khiển màn hình hiển thị, máy nghe nhạc và radio, bộ cảm ứng chụp hình, kết nối với máy tính và thiết bị ngoại vi, hoặc cao cấp hơn là kết nối với hệ thống định vị toàn cầu (GPS), tất cả đều là những hệ thống nhúng được tích hợp chung vào chiếc điện thoại.

1.2.Lịch sử phát triển

Hệ thống nhúng đầu tiên là Apollo Guidance Computer (Máy tính Dẫn đường Apollo) được phát triển bởi Charles Stark Draper tại phòng thí nghiệm

của trường đại học MIT. Hệ thống nhúng được sản xuất hàng loạt đầu tiên là máy hướng dẫn cho tên lửa quân sự vào năm 1961. Nó là máy hướng dẫn Autonetics D-17, được xây dựng sử dụng những bóng bán dẫn và một đĩa cứng để duy trì bộ nhớ. Khi Minuteman II được đưa vào sản xuất năm 1996, D-17 đã được thay thế với một máy tính mới sử dụng mạch tích hợp. Tính năng thiết kế chủ yếu của máy tính Minuteman là nó đưa ra thuật toán có thể lập trình lại sau đó để làm cho tên lửa chính xác hơn, và máy tính có thể kiểm tra tên lửa, giảm trọng lượng của cáp điện và đầu nối điện.

Từ những ứng dụng đầu tiên vào những năm 1960, các hệ thống nhúng đã giảm giá và phát triển mạnh mẽ về khả năng xử lý. Bộ vi xử lý đầu tiên hướng đến người tiêu dùng là Intel 4004, được phát minh phục vụ máy tính điện tử và những hệ thống nhỏ khác. Tuy nhiên nó vẫn cần các chip nhớ ngoài và những hỗ trợ khác. Vào những năm cuối 1970, những bộ xử lý 8 bit đã được sản xuất, nhưng nhìn chung chúng vẫn cần đến những chip nhớ bên ngoài.

Vào giữa thập niên 80, kỹ thuật mạch tích hợp đã đạt trình độ cao dẫn đến nhiều thành phần có thể đưa vào một chip xử lý. Các bộ vi xử lý được gọi là các vi điều khiển và được chấp nhận rộng rãi. Với giá cả thấp, các vi điều khiển đã trở nên rất hấp dẫn để xây dựng các hệ thống chuyên dụng. Đã có một sự bùng nổ về số lượng các hệ thống nhúng trong tất cả các lĩnh vực thị trường và số các nhà đầu tư sản xuất theo hướng này. Ví dụ, rất nhiều chip xử lý đặc biệt xuất hiện với nhiều giao diện lập trình hơn là kiểu song song truyền thống để kết nối các vi xử lý. Vào cuối những năm 80, các hệ thống nhúng đã trở nên phổ biến trong hầu hết các thiết bị điện tử và khuynh hướng này vẫn còn tiếp tục cho đến nay.

Cho đến nay, khái niệm hệ thống nhúng được nhiều người chấp nhận nhất là: hệ thống thực hiện một số chức năng đặc biệt. Không có hệ thống nhúng nào chỉ có phần mềm.

1.3.Đặc điểm của hệ thống nhúng

Hệ thống nhúng thường có một số đặc điểm chung như sau:

- Các hệ thống nhúng được thiết kế để thực hiện một số nhiệm vụ chuyên dụng chứ không phải đóng vai trò là các hệ thống máy tính đa chức năng. Một số hệ thống đòi hỏi ràng buộc về tính hoạt động thời gian thực để đảm bảo độ an toàn và tính ứng dụng; một số hệ thống không đòi hỏi hoặc ràng buộc chặt chẽ, cho phép đơn giản hóa hệ thống phần cứng để giảm thiểu chi phí sản xuất.
- Một hệ thống nhúng thường không phải là một khối riêng biệt mà là một hệ thống phức tạp nằm trong thiết bị mà nó điều khiển.
- Phần mềm được viết cho các hệ thống nhúng được gọi là firmware và được lưu trữ trong các chip bộ nhớ ROM hoặc bộ nhớ flash chứ không phải là trong một ổ đĩa. Phần mềm thường chạy với số tài nguyên phần cứng hạn chế: không có bàn phím, màn hình hoặc có nhưng với kích thước nhỏ, dung lượng bộ nhớ thấp Sau đây, ta sẽ đi sâu, xem xét cụ thể đặc điểm của các thành phần của hệ thống nhúng.

1.4.Kiến trúc của phần mềm hệ thống nhúng

Một số loại kiến trúc phần mềm thông dụng trong các hệ thống nhúng như sau:

1.4.1. Vòng lặp kiểm soát đơn giản

Theo thiết kế này, phần mềm được tổ chức thành một vòng lặp đơn giản. Vòng lặp gọi đến các chương trình con, mỗi chương trình con quản lý một phần của hệ thống phần cứng hoặc phần mềm.

1.4.2. Hệ thống ngắt điều khiển

Các hệ thống nhúng thường được điều khiển bằng các ngắt. Có nghĩa là các tác vụ của hệ thống nhúng được kích hoạt bởi các loại sự kiện khác nhau. Ví dụ, một ngắt có thể được sinh ra bởi một bộ định thời sau một chu kỳ được định nghĩa trước, hoặc bởi sự kiện khi cổng nối tiếp nhận được một byte nào đó.

Loại kiến trúc này thường được sử dụng trong các hệ thống có bộ quản lý sự kiện đơn giản, ngắn gọn và cần độ trễ thấp. Hệ thống này thường thực hiện một tác vụ đơn giản trong một vòng lặp chính. Đôi khi, các tác vụ phức tạp hơn sẽ được thêm vào một cấu trúc hàng đợi trong bộ quản lý ngắt để được vòng lặp xử lý sau đó. Lúc này, hệ thống gần giống với kiểu nhân đa nhiệm với các tiến trình rời rạc.

1.4.3. Đa nhiệm tương tác

Một hệ thống đa nhiệm không ưu tiên cũng gần giống với kỹ thuật vòng lặp kiểm soát đơn giản ngoại trừ việc vòng lặp này được ẩn giấu thông qua một giao diện lập trình API. Các nhà lập trình định nghĩa một loạt các nhiệm vụ, mỗi nhiệm vụ chạy trong một môi trường riêng của nó. Khi không cần thực hiện nhiệm vụ đó thì nó gọi đến các tiến trình con tạm nghỉ (bằng cách gọi "pause", "wait", "yield" …).

Ưu điểm và nhược điểm của loại kiến trúc này cũng giống với kiểm vòng lặp kiểm soát đơn giản. Tuy nhiên, việc thêm một phần mềm mới được thực hiện dễ dàng hơn bằng cách lập trình một tác vụ mới hoặc thêm vào hàng đợi thông dịch (queue-interpreter).

1.4.4. Đa nhiệm ưu tiên

Ở loại kiến trúc này, hệ thống thường có một đoạn mã ở mức thấp thực hiện việc chuyển đổi giữa các tác vụ khác nhau thông qua một bộ định thời. Đoạn mã này thường nằm ở mức mà hệ thống được coi là có một hệ điều hành và vì thế cũng gặp phải tất cả những phức tạp trong việc quản lý đa nhiệm.

Bất kỳ tác vụ nào có thể phá hủy dữ liệu của một tác vụ khác đều cần phải được tách biệt một cách chính xác. Việc truy cập tới các dữ liệu chia sẻ có thể được quản lý bằng một số kỹ thuật đồng bộ hóa như hàng đợi thông điệp (message queues), semaphores … Vì những phức tạp nói trên nên một giải pháp thường được đưa ra đó là sử dụng một hệ điều hành thời gian thực. Lúc đó, các

nhà lập trình có thể tập trung vào việc phát triển các chức năng của thiết bị chứ không cần quan tâm đến các dịch vụ của hệ điều hành nữa.

1.4.5. Vi nhân (Microkernel) và nhân ngoại (Exokernel

Khái niệm vi nhân (microkernel) là một bước tiếp cận gần hơn tới khái niệm hệ điều hành thời gian thực. Lúc này, nhân hệ điều hành thực hiện việc cấp phát bộ nhớ và chuyển CPU cho các luồng thực thi. Còn các tiến trình người dùng sử dụng các chức năng chính như hệ thống file, giao diện mạng lưới,… Nói chung, kiến trúc này thường được áp dụng trong các hệ thống mà việc chuyển đổi và giao tiếp giữa các tác vụ là nhanh.

Còn nhân ngoại (exokernel) tiến hành giao tiếp hiệu quả bằng cách sử dụng các lời gọi chương trình con thông thường. Phần cứng và toàn bộ phần mềm trong hệ thống luôn đáp ứng và có thể được mở rộng bởi các ứng dụng.

1.4.6. Nhân khối (monolithic kernels)

Trong kiến trúc này, một nhân đầy đủ với các khả năng phức tạp được chuyển đổi để phù hợp với môi trường nhúng. Điều này giúp các nhà lập trình có được một môi trường giống với hệ điều hành trong các máy để bàn như Linux hay Microsoft Windows và vì thế rất thuận lợi cho việc phát triển. Tuy nhiên, nó lại đòi hỏi đáng kể các tài nguyên phần cứng làm tăng chi phí của hệ thống. Một số loại nhân khối thông dụng là Embedded Linux và Windows CE. Mặc dù chi phí phần cứng tăng lên nhưng loại hệ thống nhúng này đang tăng trưởng rất mạnh, đặc biệt là trong các thiết bị nhúng mạnh như Wireless router hoặc hệ thống định vị GPS. Lý do của điều này là:

• Hệ thống này có cổng để kết nối đến các chip nhúng thông dụng.

• Hệ thống cho phép sử dụng lại các đoạn mã sẵn có phổ biến như các trình điều khiển thiết bị, Web Servers, Firewalls, …

• Việc phát triển hệ thống có thể được tiến hành với một tập nhiều loại đặc tính, chức năng còn sau đó lúc phân phối sản phẩm, hệ thống có thể được cấu

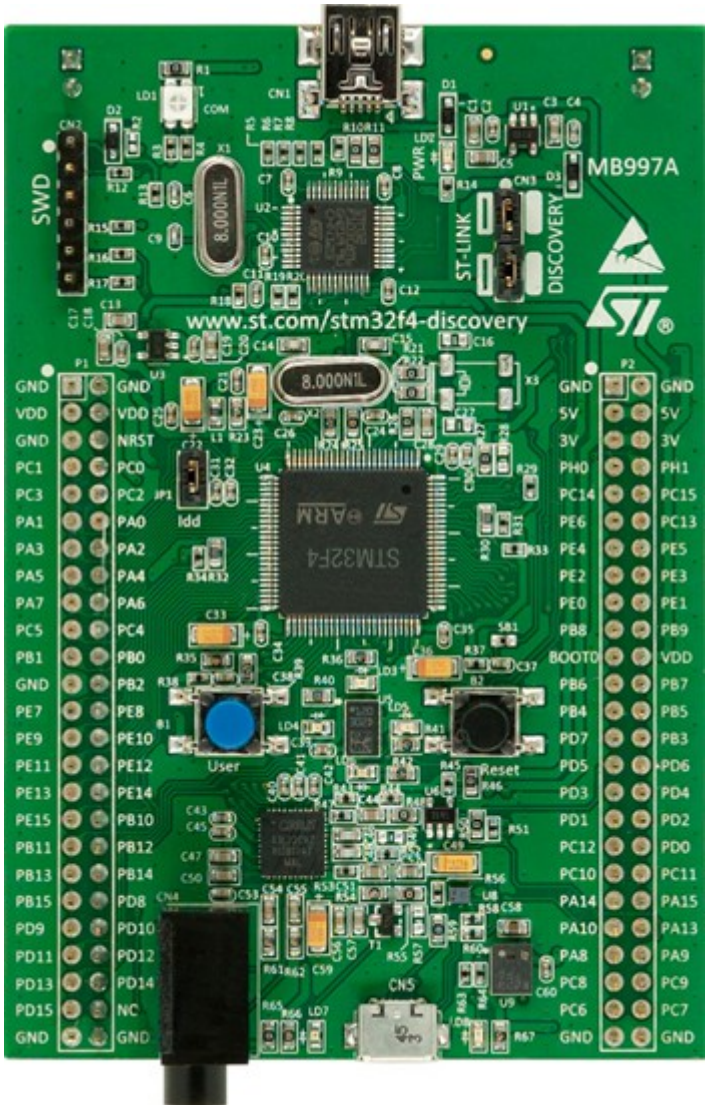
hình để loại bỏ một số chức năng không cần thiết. Điều này giúp tiết kiệm được những vùng nhớ mà các chức năng đó chiếm giữ.

• Hệ thống có chế độ người dùng để dễ dàng chạy các ứng dụng và gỡ rối. Nhờ đó, qui trình phát triển được thực hiện dễ dàng hơn và việc lập trình có tính linh động hơn.

• Có nhiều hệ thống nhúng thiếu các yêu cầu chặt chẽ về tính thời gian thực của hệ thống quản lý. Còn một hệ thống như Embedded Linux có tốc độ đủ nhanh để trả lời cho nhiều ứng dụng. Các chức năng cần đến sự phản ứng nhanh cũng có thể được đặt vào phần cứng.

Chương 2. Giới thiệu về vi điều khiển STM32

2.1.Giới thiệu chung về dòng ARM cortex



Dòng ARM Cortex là một bộ xử lí thế hệ mới đua ra một kiến trúc chuẩn cho nhu cầu đa dạng về công nghệ. Không giống nhu các chip ARM khác, dòng Cortex là một lõi xử lí hoàn thiện, dua ra một chuẩn CPU và kiến trúc hệ thống chung.

Dòng Cortex gồm có 3 phân nhánh chính: dòng A dành cho các ứng dụng cao cấp, dòng R dành cho các ứng dụng thời gian thực nhu các đầu đọc và dòng M dành cho các ứng dụng vi điều khiển và chi phí thấp. STM32 được thiết kế dựa trên dòng Cortex-M3, dòng Cortex-M3 được thiết kế đặc biệt để nâng cao

hiệu suất hệ thống, kết hợp với tiêu thụ nang lượng thấp, CortexM3 đuợc thiết kế trên nền kiến trúc mới, do đó chi phí sản xuất đủ thấp để cạnh tranh với các dòng vi điều khiển 8 và 16-bit truyền thống.

Các chip ARM7 và ARM9 được các nhà sản xuất bán dẫn thiết kế với giải pháp riêng của mình, đặc biệt là phần xử lí các các ngắt đặc biệt (exception) và các ngắt thông thuờng (interrupt). Cortex-M3 đưa ra một lõi vi điều khiển chuẩn nhằm cung cấp phần tổng quát, quan trọng nhất của một vi điều khiển, Trang 6 Author: ARMVN Kiến trúc co bản của SMT32 - ARM Cortex M3 www.arm.vn bao gồm hệ thống ngắt (interrupt system), SysTick timer (đuợc thiết kế cho hệ điều hành thời gian thực), hệ thống kiểm lỗi (debug system) và memory map. Không gian địa chỉ 4Gbyte của Cortex-M3 được chia thành các vùng cho mã chuong trình, SRAM, ngoại vi và ngoại vi hệ thống.

Không giống với ARM7 đuợc thiết kế theo kiến trúc Von Neumann (bộ nhớ chuong trình và bộ nhớ dữ liệu chung với nhau), Cortex-M3 đuợc thiết kế dựa theo kiến trúc Harvard (bộ nhớ chuong trình và bộ nhớ dữ liệu tách biệt với nhau), và có nhiều bus cho phép thực hiện các thao tác song song với nhau, do đó làm tang hiệu suất của chip. Không giống với các kiến trúc ARM trước đó, dòng Cortex cho phép truy cập dữ liệu không xếp hàng (unaligned data, vì chip ARM là kiến trúc 32bit, do đó tất cả các dữ liệu hoặc mã chuong trình đều được sắp sếp khít với vùng bộ nhớ là bội số của 4byte). Đặc điểm này cho phép sử dụng hiệu quả SRAM nội. Dòng Cortex còn hỗ trợ việc đặt và xoá các bit bên trong hai vùng 1Mbyte của bộ nhớ bằng phuong pháp gọi là bit banding. Đặc điểm này cho phép truy cập hiệu quả tới các thanh ghi ngoại vi và các cờ được dùng trên bộ nhớ SRAM mà không cần một bộ xử lí luận lí (Boolean processor).

2.2.Giới thiệu tổng quan về vi điều khiển STM32F407VG

2.2.1. Tính năng

Vi điều khiển STM32F407VGT6 được trang bị chip 32-bit ARM Cortext ®với nhân FTU, 1MB bộ nhớ Flash, 192Kb RAM trong gói LQFP100.

ST-LINK/V2 được tích hợp trực tiếp trên board.

Nguồn cung cấp cho board: thông qua cáp USB hoặc nguồn 5V.

Nguồn điện có thể cung cấp cho các ứng dụng bên ngoài: 5V hoặc 3V.

LIS302DL or LIS3DSH ST MEMS 3-axis gia tốc kế.

Có cảm ứng âm thanh với microphone kỹ thuật sô omni-directional.

CS43L22 audio DAC với bộ điều khiển loa class D được tích hợp.

Cung cấp 8 LED:

LD1(đỏ/xanh) dành cho giao tiếp USB.

LD2(đỏ) hiển thị nguồn điện 3.3V.

4 đèn LED dành cho người dùng: LD3(cam), LD4 (xanh lá), LD5(đỏ), LD6(cam)

2 USB OTG LED LD7(xanh) Vbus và LD8(đỏ)

2 nút bấm (user và reset)

USB OTG FS với cổng kết nối Micro AB.

Các chân I/O mở rộng để kết nối nhanh nhằm mục đích tạo các bảng mạch mẫu và dễ dàng tìm kiếm.

Đi kèm các phần mềm miễn phí bao gồm nhiều ví dụ, một phần của gói STM32CubeF4 hoặc STSW-STM32068 nhằm kế thừa các thư viện thông thường.

2.2.2. Mô tả

STM32F4DISCOVERY giúp khám phá các dòng tính năng của STM32F407/417 và phát triển các ứng dụng một cách dễ dàng. Nó bao gồm tất cả những gì cần thiết cho người mới bắt đầu và cả những người đã có kinh nghiệm để có thể bắt đầu một cách nhanh chóng.

Dựa trên STM32F407VGT6, nó bao gồm cả công cụ ST-LINK/V2, 2 gia tốc kế kỹ thuật số ST MEMS và mic kỹ thuật số, 1 CS43L22 audio DAC với bộ điều khiển loa class D được tích hợp, các đèn LED và các nút bấm cổng kết nối USB OTG microAB.

Để mở rộng các chức năng của kit STM32F4 với kết nối mạng, màn hình LCD và các thiết bị khác, truy cập www.st.com/stm32f4disexpansion.

2.2.3. Yêu cầu hệ thống

- Windows PC (XP, 7, 8)
- USB type A to Mini-B cable.

2.2.4. Các công cụ phát triển.

- IAR EWARW (IAR Embedded Workbench®)
- Keil ®MDK-ARM™
- GCC-based IDE (ARM® Atollic®TrueSTUDIO®,...).

2.3.Tổng quan về các công nghệ được sử dụng

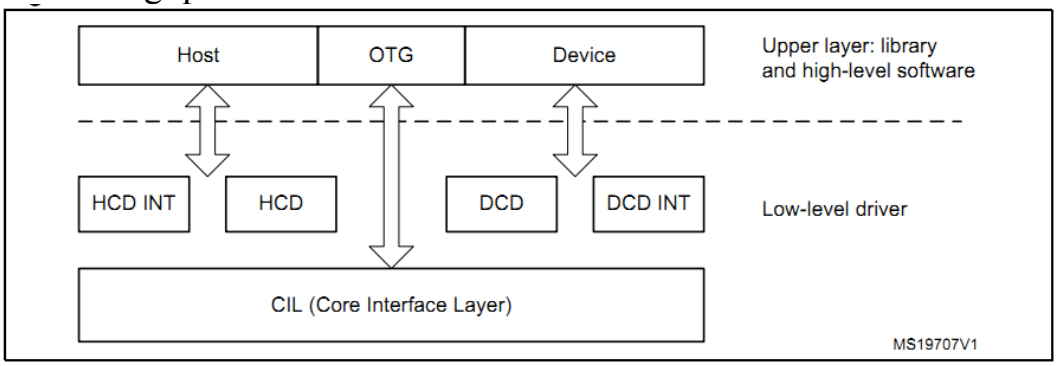
2.3.1. USB host

Các USB On-The-Go Host và Device Library là một phần mềm và gói ứng dụng phần mềm cho USB máy chủ và các thiết bị. Gói này bao gồm các ví dụ và phần mềm trình diễn để phát triển các ứng dụng bằng cách sử dụng USB tốc độ đầy đủ và loại tốc độ truyền cao (kiểm soát, ngắt, số lượng lớn và thời gian).

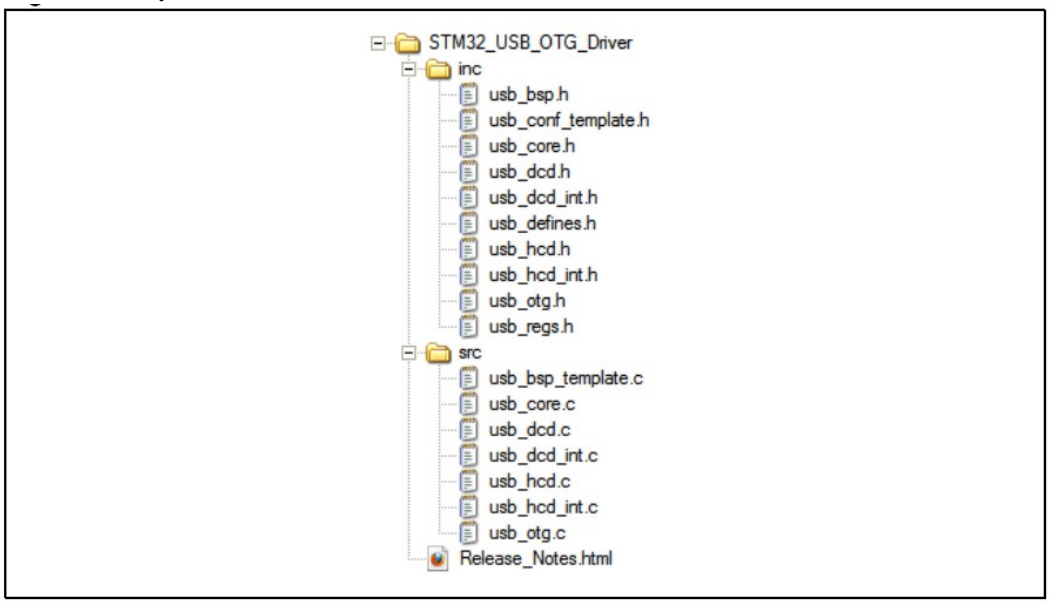
Mục đích của USB OTG Host và thư viện Device là để cung cấp ít nhất một firmware ví dụ minh chứng cho từng loại chuyển USB. Thư viện này được thiết kế để sử dụng với các dòng chip sau:

- STM3210C-EVAL evaluation board (UM0600) for STM32F105/7 devices.
- STM3220G-EVAL evaluation board (UM1057) for STM32F20x devices
- STM3221G-EVAL evaluation board (UM1065) for STM32F21x devices
- STM3240G-EVAL evaluation board (UM1461) for STM32F40x devices
- STM3241G-EVAL evaluation board (UM1460) for STM32F41x devices

➢ Sơ đồ tổng quan kiến trúc driver của USB host.

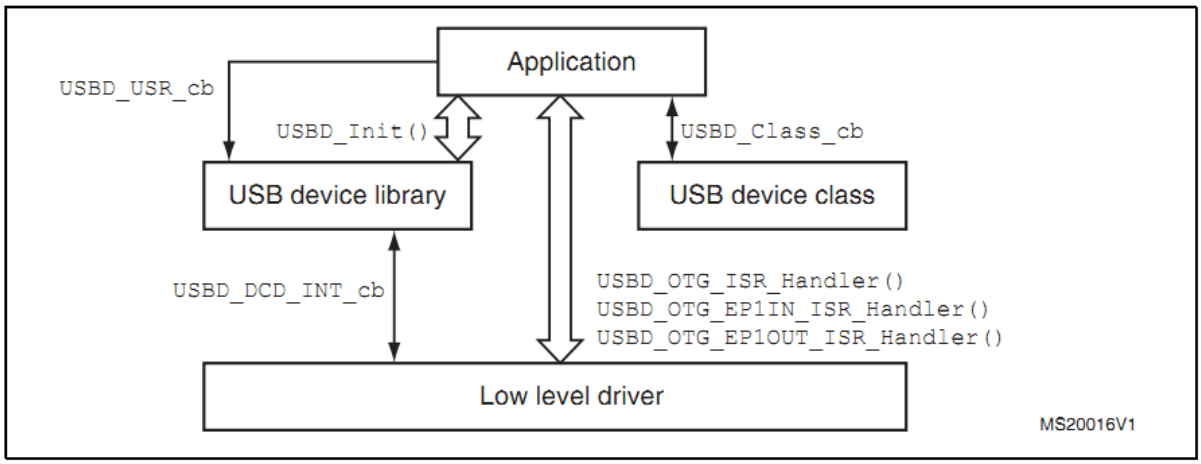


➢ Các thư viện driver của USB host



Cơ chế	File	Mô tả
Common	usb_core.c/h	Tập tin này có chứa các lớp trừu tượng phần cứng và hoạt động giao tiếp USB.
	usb_core.c/h	Tập tin này có chứa các cấu hình lõi cho máy chủ, thiết bị và chế độ OTG: kích thước Transmit FIFO, Nhận kích thước FIFO, Core, chế độ và các tính năng được lựa chọn ... vv. Tập tin này sẽ được sao chép vào thư mục ứng dụng vàbiến đổi tùy thuộc vào nhu cầu ứng dụng.

	usb_bsp_template.c	Tập tin này có chứa các cấu hình mức thấp lõi (ngắt, GPIO). Tập tin này sẽ được sao chép vào thư mục ứng dụng và biến đổi tùy thuộc vào nhu cầu ứng dụng.
Host	usb_hcd.c/h	Tập tin này có chứa các lớp giao diện máy chủ được sử dụng bởi các thư việnđể truy cập vào lõi.
	usb_hcd_int.c/h	Tập tin này có chứa các chương trình con ngắt cho chế độ Host.
Device	usb_dcd.c/h	Tập tin này có chứa các lớp giao diện thiết bị được sử dụng bởi cácthư viện để truy cập vào lõi.
	usb_dcd_int.c/h	Tập tin này có chứa các chương trình con ngắt cho chế độ Device.
OTG	usb_otg.c/h	Tập tin này có chứa các thực hiện của SRP và HNP giao thức và các ngắt tương đối so với các chế độ OTG.



Tầng ứng dụng chỉ để gọi một chức năng (USBD_Init) để khởi tạo các trình điều khiển cấp thấp USB, thư viện thiết bị USB, phần cứng trên bảng sử dụng (BSP) và bắt đầu thư viện. Ứng dụng này cũng có sử dụng các ISR USB nói chung và thủ tục con EP1 cụ thể khi USB_OTG_HS_DEDICATED_EP1_ENABLED xác định trong file usb_conf.h.

Tuy nhiên các chức năng USBD_Init cần cơ cấu sử dụng gọi lại để thông báo cho người sử dụng lớp của các tiểu bang khác nhau của thư viện và các tin nhắn và các cấu trúc lớp gọi lại để bắt đầu giao diện lớp.Các trình điều

khiển USB mức thấp có thể được liên kết với các thư viện thiết bị USB thông qua cấu trúc USBD_DCD_INT_cb. Cấu trúc này đảm bảo một số độc lập giữa các thư viện thiết bị USB và các trình điều khiển cấp thấp; tạo điều kiện cho các trình điều khiển cấp thấp để được sử dụng bởi bất kỳ thư viện thiết bị khác.

- Cấu hình thư viện cho thiết bị USB
  Để cấu hình các thư viện cho USB ta sẽ sử dụng file usbd_conf.h như sau:

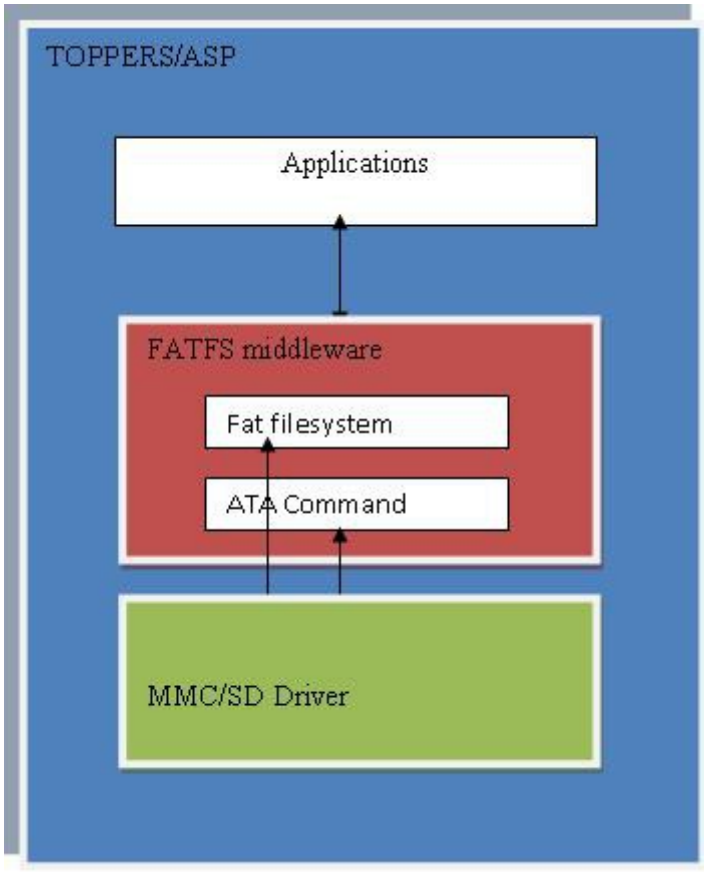
```
#define USBD_CFG_MAX_NUM              1
#define USB_MAX_STR_DESC_SIZ        64

/****  USB_MSC_Class_Layer_Parameter *********/
#define MSC_IN_EP                        0x81
#define MSC_OUT_EP                      0x01
#define MSC_MAX_PACKET          512
#define MSC_MEDIA_PACKET      4096

/****  USB_HID Class_Layer_Parameter *********/
#define HID_IN_EP                        0x81
#define HID_OUT_EP                      0x01
#define HID_IN_PACKET             4
#define HID_OUT_PACKE            4
```

2.3.2. Modul giao tiếp thẻ nhớ FATFs

FATFs là bộ mã nguồn miễn phí hỗ trợ định dạng FAT(File Allocation Table) được sử dụng rộng rãi trong hệ điều hành Windows. Tổ chức của FatFS được mô tả như sau:



FatFS sẽ cung cấp giao diện các hàm cơ bản để thực thi các thao tác file trên thẻ MMC/SD tương tự như lập trình file trên máy tính.

- FatFS hỗ trợ các kiểu format FAT12, FAT16, FAT32.
- Các hàm giao diện lập trình: f_mount, f_open, f_close, f_read, f_write,f_lseek, f_sync, f_opendir, f_readdir, f_getfree, f_stat, f_mkdir, f_untrnk, f_chmod, f_rename, f_mkfs.

FatFS gồm 2 phần chính là phần FAT bao gồm các hàm liên quan đến File Allocation Table, và phần ATA liên quan đến xử lý các lệnh ATA để giao tiếp với thẻ nhớ.

Tổ chức file chính của FatFs gồm:

- file ff.c: gồm các hàm hỗ trợ FAT.
- file ata.c: gồm các hàm giao tiếp ATA command.

Để giao tiếp với phần điều khiển (driver) của thẻ MMC/SD, FatFS yêu cầu 5 hàm giao diện:

- disk_initialize: hàm khởi tạo kết nối tới thẻ MMC/SD, đồng thời kích hoạt cho thẻ ở trạng thái sẵn sàng hoạt động.
- disk_write: ghi dữ liệu vào một vùng sector nhất định.
- disk_read: đọc dữ liệu từ một vùng sector cho trước.
- disk_status: lấy trạng thái của thẻ.
- disk_ioctl: các hàm xử lý các yêu cầu khác sẽ được bố trí xử lý ở đây.

Kiến trúc của FatFS đơn giản do đó rất tiện cho việc "port" sang một hệ nhúng khác. Người dùng không cần có kiến thức sâu về FAT vẫn có thể sử dụng FatFS vào hệ thống của mình.

Bảng các FatFS API commands.

Function	Description
f_mount	Register/Unregister a work area.
f_open	Open/Create a file.
f_close	Close a file.
f_read	Read file.
f_write	Write file.
f_lseek	Move read/write pointer, expand file size.
f_truncate	Truncate file size.
f_sync	Flush cached data.
f_opendir	Open a directory.
f_readdir	Read a directory item.
f_getfree	Get free clusters.
f_stat	Get file status.
f_mkdir	Create a directory.
f_unlink	Remove a file or directory.
f_chmod	Change attribute.
f_utime	Change timestamp.
f_rename	Rename/Move a file or directory.
f_mkfs	Create a file system on the drive.
f_forward	Forward file data to the stream directly.
f_chdir	Change current directory.
f_chdrive	Change current drive.
f_getcwd	Retrieve the current directory.
f_gets	Read a string.
f_putc	Write a character.
f_puts	Write a string.
f_printf	Write a formatted string.

2.3.3. Bộ giải mã Mp3 của Real Network

Các bộ giải mã MP3 Helix cung cấp giải mã MPEG-compliant nội dung MP3. Cả hai bộ giải mã gồm dạng dấu chấm động và dạngnhị phândều được triển khai để để dụng. Các bộ giải mã nhị phân được tối ưu đặc biệt cho bộ vi xử lý ARM nhưng có thể chạy trên bất kỳ bộ xử lý điểm cố định 32-bit có thể thực hiện nhiều phép dài (hai đầu vào 32-bit tạo ra một kết quả 64-bit) và dài nhân-tích lũy (dài nhân với 64-bit accumulator).

Các tính năng chính:

- Mã code C cho porting đến các nền tảng mới
- Tối ưu hóa cho các bộ xử lý ARM
- Tùy chọn C ++ API để tương thích với các khách hàng Helix
- Được thiết kế cho hiệu suất cao và tiêu thụ điện năng thấp trong các thiết bị cầm tay và điện thoại di động
- Hỗ trợ đầy đủ cho 3 lớp:
  - MPEG1 lớp 3 - tần số lấy mẫu: 48 KHz, 44.1 KHz, 32 KHz
  - Lớp MPEG2 3 - tần số lấy mẫu: 24 KHz, 22,05 KHz, 16 KHz
  - Lớp MPEG2.5 3 - tần số lấy mẫu: 12 KHz, 11,025 KHz, 8 KHz
- Hỗ trợ bitrate đổi, bitrate biến, và các chế độ bitrate miễn phí
- Hỗ trợ mono và mọi chế độ stereo (stereo bình thường, âm thanh stereo doanh, dual-mono)
- Tùy chọn sử dụng Intel® IPP thư viện thực hiện (nếu có)
- Dễ dàng để liên kết trong thư viện hoặc IPP hoặc mã Helix

➢ Đặc điểm kỹ thuật

- Mức độ tiêu thụ CPU trung bình

Sample Rate	Channels	Bit Rate	Processor Model (1)					
			ARM7TDMI	ARM9TDMI-REV2	ARM920T	ARM9E	StrongARM1	XScale
48.0 KHz	2	320 Kbps	30 MHz	24 MHz	27 MHz	20 MHz	20 MHz	20 MHz
44.1 KHz	2	128 Kbps	26 MHz	21 MHz	24 MHz	17 MHz	17 MHz	17 MHz

- Mức độ tiêu thụ bộ nhớ

ROM = 13446 Bytes (const globals)

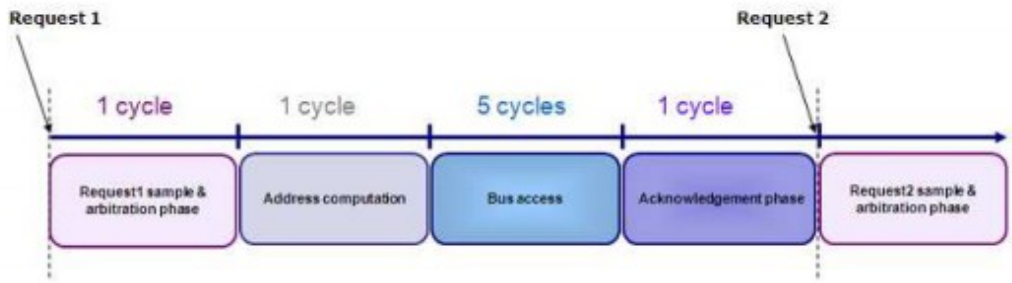
RAM = 23816 Bytes (heap)

Total Data Memory = 37262 Bytes

Code Size = 21000 Bytes (approximately - depends on compiler)

2.3.4. Cơ chế DMA

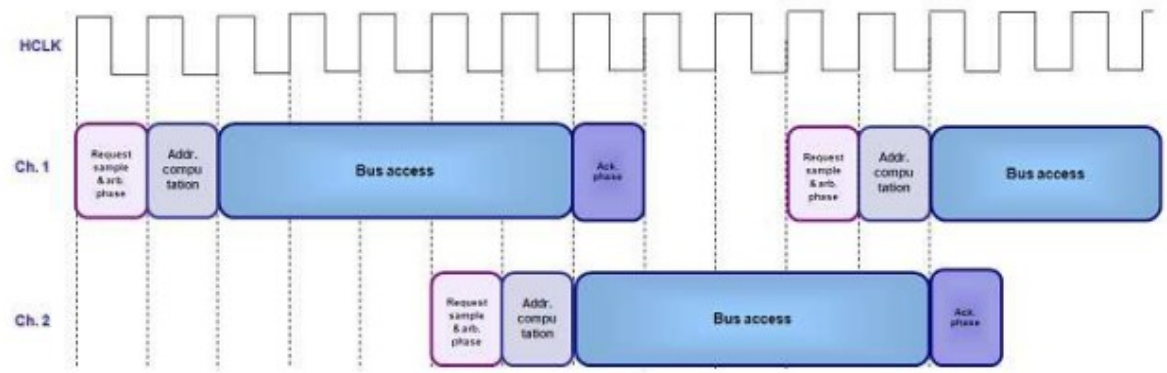
Mặc dù có thể sử dụng chính nhân Cortex để trao đổi dữ liệu giữa các thiết bị ngoại vi và SRAM nội, tuy nhiên chúng ta có thể hoàn toàn sử dụng cơ chế tự động cho việc này với bộ quản lý DMA. STM32 có 7 kênh độc lập dùng để truyền dữ liệu từ bộ nhớ sang bộ nhớ, ngoại vi đến bộ nhớ, bộ nhớ tới ngoại vi và ngoại vi đến ngoại vi. Trong trường hợp trao đổi dữ liệu từ bộ nhớ đến bộ nhớ, tốc độ dữ liệu phụ thuộc vào tốc độ của kênh DMA quản lý nó. Còn với việc giao tiếp dữ liệu với ngoại vi thì tốc độ phụ thuộc vào bộ điều khiển ngoại vi và hướng dữ liệu di chuyển. Cùng với chuyển dữ liệu theo luồng, DMA của STM32 còn hỗ trợ bộ đệm theo vòng, vì hầu hết các thiết bị ngoại vi hiện nay không có bộ nhớ FIFO, mỗi bộ DMA sẽ lưu dữ liệu vào bộ nhớ SRAM. Bộ DMA của STM32 được thiết kế dành riêng cho dữ liệu truyền tốc độ cao và nhỏ.



Each DMA memory to memory transfer is made up of four phases each 1 cycle long except for the bus access phase that takes five cycles for each word transferred

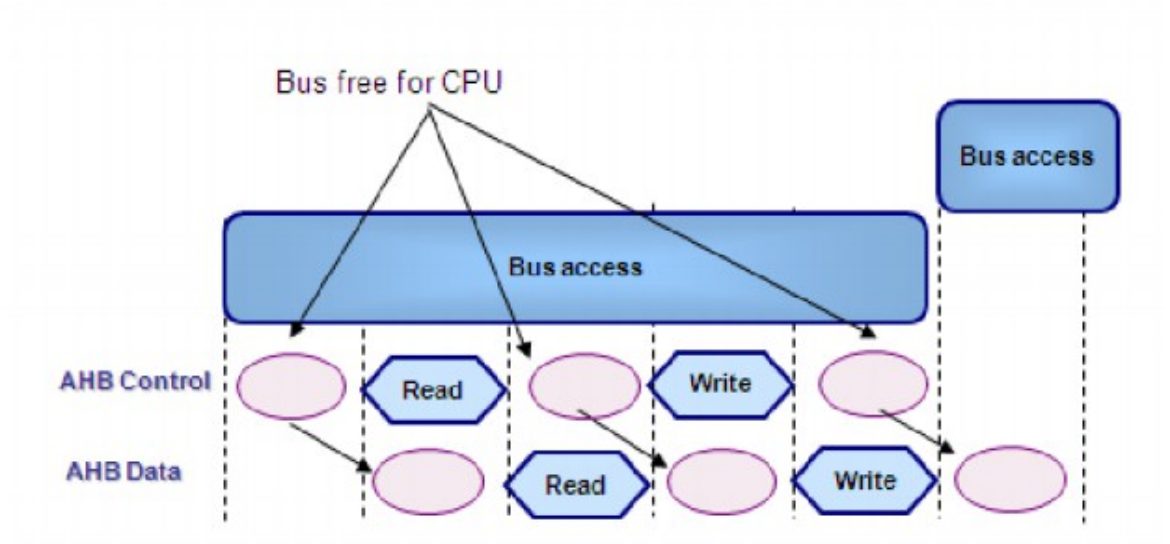
Mỗi thao tác của của bộ nhớ DMA gồm 4 giai đoạn, đó là : lấy mẫu và phân xử, tính toán địa chỉ, truy cập đường truyền, và cuối cùng là hoàn tất. Mỗi giai đoạn thực hiện trong một chu kỳ lệnh, riêng truy cập đường truyền mất 5 chu kỳ lệnh. Ở giai đoạn truy cập đường truyền thực chất là giai đoạn dữ liệu được truyền, mỗi từ (WORD) sẽ mất 3 chu kỳ lệnh. Bộ DMA và CPU được thiết kế cùng lúc có thể hoạt động mà không có tranh chấp tài nguyên lẫn nhau. Giữa 2 kênh DMA khác nhau, sẽ có sự ưu tiên mức hoạt động, dựa trên đó bộ phận phân xử sẽ quyết định kênh DMA có mức ưu tiên cao hơn sẽ lấy được tài nguyên trước. Nếu 2 kênh DMA có cùng mức ưu tiên, lại đang ở trạng thái chờ

để truy cập tài nguyên, thì kênh DMA có số thứ tự nhỏ hơn sẽ được sử dụng tài nguyên trước.



Bộ DMA được thiết kế cho truyền dữ liệu tốc độ và kích thước nhỏ. Bộ DMA chỉ sử dụng bus dữ liệu khi ở giai đoạn truy cập đường truyền.

Bộ DMA có thể thực hiện việc phân xử tài nguyên và tính toán địa chỉ trong khi bộ DMA khác đang ở giai đoạn truy cập đường truyền như mô tả ở hình trên. Ngay khi bộ DMA thứ nhất kết thúc việc truy cập đường truyền, bộ DMA 2 có thể ngay lập tức sử dụng đường truyền dữ liệu. Điều này vừa làm tăng tốc độ truyền dữ liệu, tối đa hóa việc sử dụng tài nguyên.



Ở giai đoạn Bus Access CPU sẽ có 3 chu kỳ rảnh. Khi chuyển dữ liệu từ vùng nhớ sang vùng nhớ điều này sẽ đảm bảo nhân Cortex-M3 sử dụng 60% dung lượng của đường truyền dữ liệu cho dù bộ DMA vẫn hoạt động liên tục.

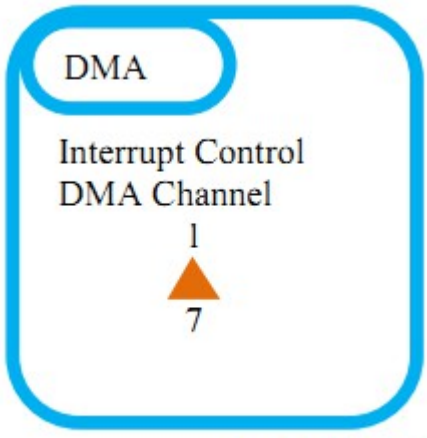
Trong trường hợp trao đổi dữ liệu từ vùng nhớ sang vùng nhớ mỗi kênh DMA chỉ sử dụng đường truyền dữ liệu ở giai đoạn Bus Access và 5 chu kỳ CPU để chuyển 2 bytes dữ liệu. Trong đó một chu kỳ để đọc và 1 chu kỳ để ghi, 3 chu kỳ còn lại nhằm giải phóng đường truyền dữ liệu cho nhân Cortex. Điều đó có nghĩa là bộ DMA chỉ sử dụng tối đa 40% băng thông của đường dữ liệu. Tuy nhiên giai đoạn Bus Access hơi phức tạp ở trường hợp dữ liệu truyền giữa thiết bị ngoại vi và bộ nhớ do liên quan đến AHB và APB. Trao đổi trên bus AHB sử dụng sử dụng 2 chu kỳ và xung nhịp của AHB, trên bus APB sẽ sử dụng 2 chu kỳ xung nhịp của APB cộng thêm 2 chu kỳ xung nhịp của AHB. Mỗi lần trao đổi dữ liệu, bộ DMA sẽ sử dụng bus AHB, bus APB và 1 chu kỳ xung nhịp AHB. Ví dụ để chuyển dữ liệu từ bus SPI tới SRAM chúng ta sẽ sử dụng :

SPI đến SRAM sử dụng DMA = SPI transfer(APB) + SRAM transfer(AHB) + free cycle(AHB)

= (2 APB cycles + 2 AHB cycles) + (2 AHB cycles) + (1 AHB cycle)

= (2 APB cycles) + (5 AHB cycles)

*Lưu ý : quá trình trên chỉ sử dụng với nhân Cortex sử dụng đường I-bus để nạp lệnh cho nhân xử lý.

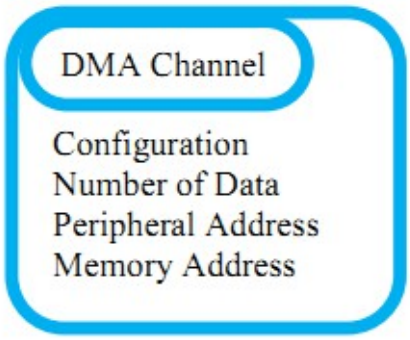


STM32 có 7 bộ DMA độc lập với nhau

Việc sử dụng DMA rất đơn giản. Đầu tiên là kích hoạt đồng hồ xung nhịp:

```
RCC->AHBENR |= 0x00000001; //enable DMA clock
```

Một khi được cấp nguồn khối DMA sẽ được điều khiển bởi 4 thanh ghi điều khiển. 2 thanh ghi điều khiển địa chỉ đích và nguồn ngoại vi và vùng nhớ. Kích thước dữ liệu truyền và cấu hình tổng quan DMA được lưu trong 2 thanh ghi còn lại.

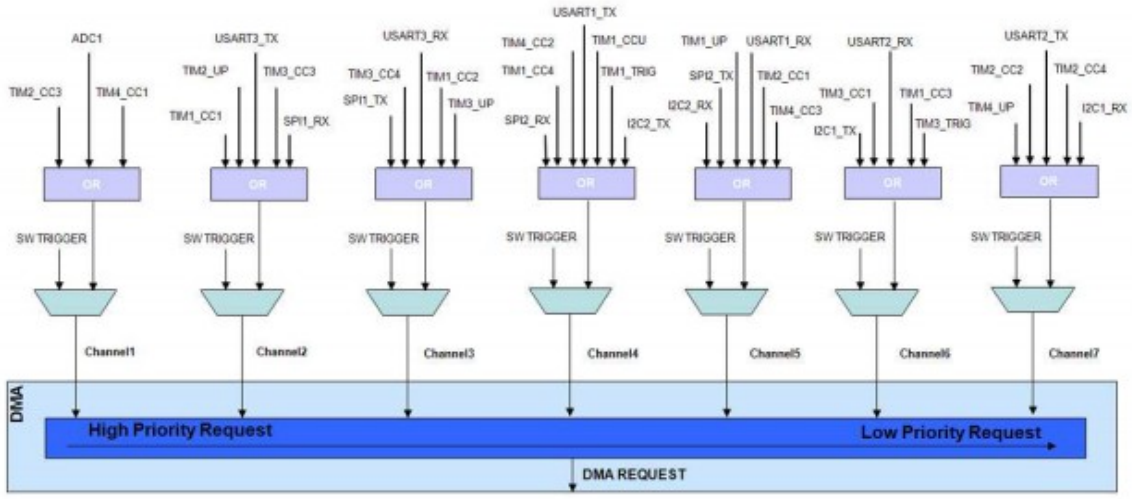


Mỗi bộ DMA có 4 thanh ghi điều khiển, 3 nguồn tín hiệu interrupt: hoàn tất, hoàn tất một nửa, lỗi.

Mỗi kênh DMA có thể được gắn với một mức ưu tiên : cao, rất cao, trung bình , thấp. Kích cỡ dữ liệu được truyền có thể điều chỉnh để phù hợp cho ngoại vi vùng nhớ. Ví dụ với bộ nhớ kích cỡ là 32 bit (mất 3 chu kỳ để truyền), chuyển đến ngoại vi URAT là 4 đơn vị dữ liệu mất 8 bit mất 35 chu kỳ thay vì mất 64 chu kỳ thay vì 64 chu kỳ nếu chuyển từng đơn vị dữ liệu riêng lẻ 8 bit. Chúng ta có thể tăng địa chỉ chỉ đích hoặc nguồn trong quá trình truyền dữ liệu. Ví dụ như khi lấy dữ liệu từ bộ ADC, chúng ta có thể tăng địa chỉ vùng nhớ lên để lưu giá trị từ ADC vào mảng dữ liệu. Trên thanh ghi điều khiển có Transfer Direction Bit cho phép ta cấu hình hướng dữ liệu từ ngoại vi vào vùng nhớ trên SRAM, ta kích hoạt bit 14 trên thanh ghi điều khiển. Ngoài việc sử dụng DMA với chế độ vòng lặp chờ, chúng ta có thể sử dụng ngắt cho để theo dõi quá trình chuyển dữ liệu. Có 3 loại ngắt hỗ trợ cho DMA: hoàn thành chuyển dữ liệu, hoàn thành chuyển dữ liệu, hoàn thành một nửa và lỗi. Sau khi cấu hình hoàn tất, chúng ta kích hoạt Channel Enable Bit để thực hiện quá trình chuyển dữ liệu. Ví dụ sau mô tả quá trình chuyển dữ liệu giữa 2 vùng nhớ trên SRAM:


```
DMA_Channel1->CCR = 0x00007AC0; //cấu hình mem to mem
DMA_Channel1->CPAR = (unsigned int)src_array;//địa chỉ nguồn
DMA_Channel1->CMAR=(unsigned int)dst_array;//địa chỉ đích
DMA_Channel1->CNDTR=0x000A;//số lượng dữ liệu
TIM2->CR1 = 1;              //khởi tạo thời gian
DMA_Channel1->CCR |= 0x00000001;//kích hoạt quá trình chuyển dữ liệu
While( !(DMA->ISR & 0x00000001); //chờ cho đến khi hoàn thành
TIM2->CR1 = 0; //ngưng đếm
TIM2->CNT = 0; // thiết lập giá trị đếm bằng 0
TIM2->CR1 = 1; //bắt đầu đếm lại
for(index = 0;index < 0x000A;index++)
        dst_array[index]=src_array[index];
TIM2-CR1 = 0;
```

Ở đoạn mã trên, ta sử dụng TIM2 để đo thời gian (tính theo chu kỳ) chuyển dữ liệu từ 2 vùng nhớ nhớ kích thước 10 WORD. Với DMA quá trình chuyển tiêu tốn 220 chu kỳ, với cách sử dụng CPU tiêu tốn 356 chu kỳ.



Mỗi kênh DMA được gán với ngoại vi nhất định. Khi được kích hoạt, các thiết bị ngoại vi sẽ điều khiển bộ DMA tương ứng.

Kiểu truyền dữ liệu từ bộ nhớ sang bộ nhớ thường hay được dùng để khởi tạo vùng nhớ hay chép các vùng nhớ dữ liệu lớn. Phần lớn tác vụ DMA hay được sử dụng để truyền dữ liệu giữa ngoại vi bộ nhớ và vùng nhớ. Để sử dụng

DMA, đầu tiên ta khởi tạo thiết bị ngoại vi và kích hoạt chế độ DMA trên thiết bị đó, sau đó khởi tạo kênh DMA tương ứng.

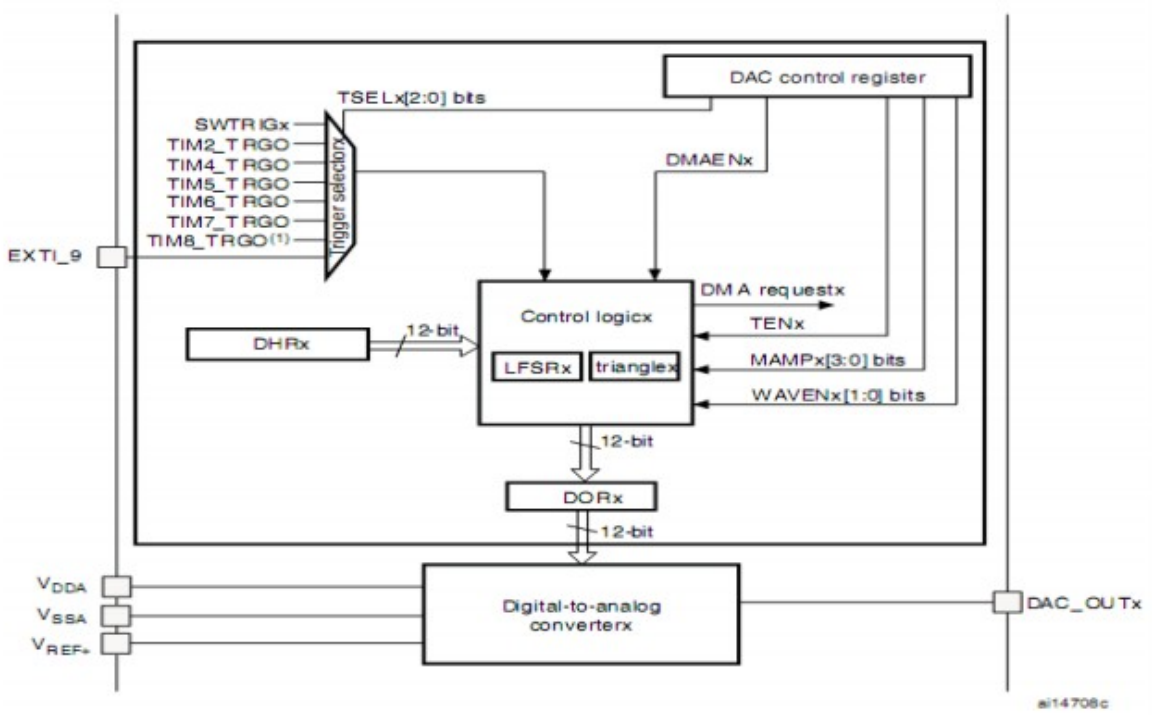
2.3.5. DAC

DAC (Digital-to-analog converter): là bộ chuyển đổi tín hiệu ở dạng số sang dạng tương tự,thường được ứng dụng trong lĩnh vực tiêu dùng. Ví dụ : audio player….

Chuyển đổi số sang tương tự là tiến trình lấy một giá trị được biểu diễn dưới dạng mã số (digitalcode) và chuyển đổi nó thành mức điện thế hoặc dòng điện tỉ lệ với giá trị số.

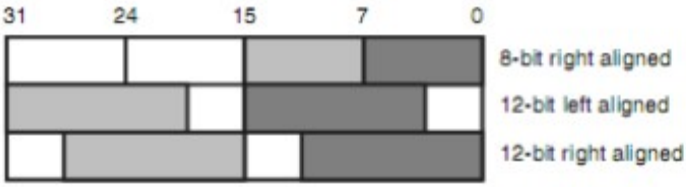
STM32 có 2 bộ DAC, và mỗi bộ có 1 kênh output tương ứng.Các tính năng của bộ DAC gồm có:

- Dữ liệu 12-bit, canh lề trái hoặc phải.
- Khả năng đồng bộ update.
- Sinh ra tín hiệu Noise-wave & triangular-wave.
- Dual DAC (Independent/simultaneous)
- Hỗ trợ DMA cho mỗi kênh
- Hỗ trợ các nguồn kích ngoại
- Điện thế tham chiếu Vref+



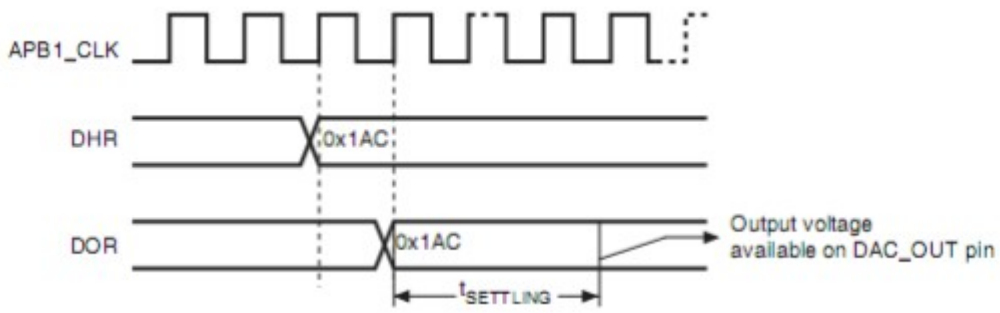
- DAC data format:
  - Single mode

  - Dual mode



- DAC conversion:

Dữ liệu DAC không thể ghi trực tiếp vào thanh ghi DOR mà phải nạp dữ liệu vào thanhghi DHR. Dữ liệu trong DHR sẽ được tự động chuyển qua thanh ghi DOR trong 1 chu kì APB1clock, nếu không có nguồn trigger nào đc kích hoạt, tuy nhiên nếu có nguồn trigger được chọn, sẽtốn 3 chu kì để thực hiện.



- DAC output voltage:
- $DACoutput = V_{REF} \times \frac{DOR}{4095}$
- DMA request:
  - Mỗi kênh DAC đều có khả năng sử dụng bộ DMA. 2 kênh DMA sử dụng cho bộ DAC.
  - Một yêu cầu DAC DMA được tạo ra khi có một nguồn kích ngoại, giá trị của thanh ghiDHR sẽ được chuyển qua thanh ghi DOR.
- Cấu hình DAC của STM32
  - Cấu hình Clock ngoại vi:

```
/* GPIOA Periph clock enable */
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
 /* DAC Periph clock enable */
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
```

  - Cấu hình port IO:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;

GPIO_Init(GPIOA, &GPIO_InitStructure);
```

- Cấu hình bộ DAC:

```
/* DAC channel1 Configuration */
DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Disable;
DAC_Init(DAC_Channel_1, &DAC_InitStructure);
DAC_Cmd(DAC_Channel_1, ENABLE);
```

- Set giá trị cho thanh ghi DHR:

- DAC_SetChannel1Data(DAC_Align_12b_R, DAC_data_output);

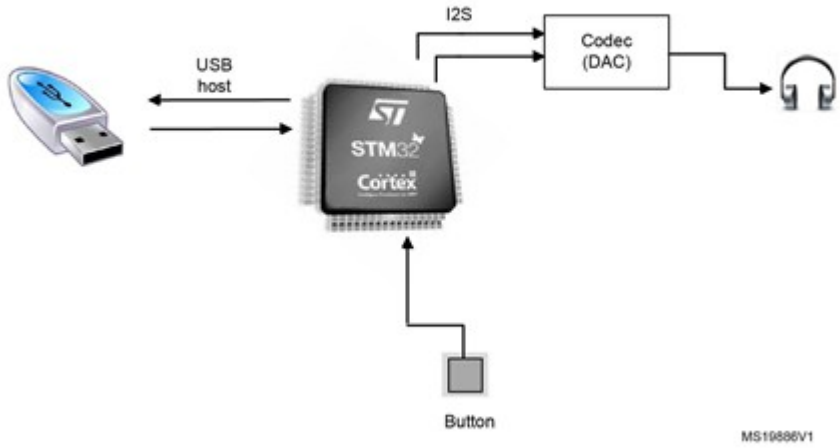
Chương 3. Ứng dụng chơi nhạc MP3

3.1.Giới thiệu chung

Ứng dụng hỗ trợ này sử được hỗ trợ hai kiểu lưu trữ dữ liệu, một là lưu trữ trong bộ nhớ trong Flash và cách hai là lưu trữ dữ liệu tại bộ nhớ ngoài USB.

Hai cách lưu trữ này được định nghĩa trong thư viện main.h.

Driver của phần mềm này có thể đọc dữ liệu được lưu trữ trong bộ nhớ Flash hoặc dữ liệu được lưu trữ từ USB.



Hình 1: Kiến trúc tổng quan của ứng dụng

Ứng dụng này được xây dựng dựa trên chip STM32F407VG.

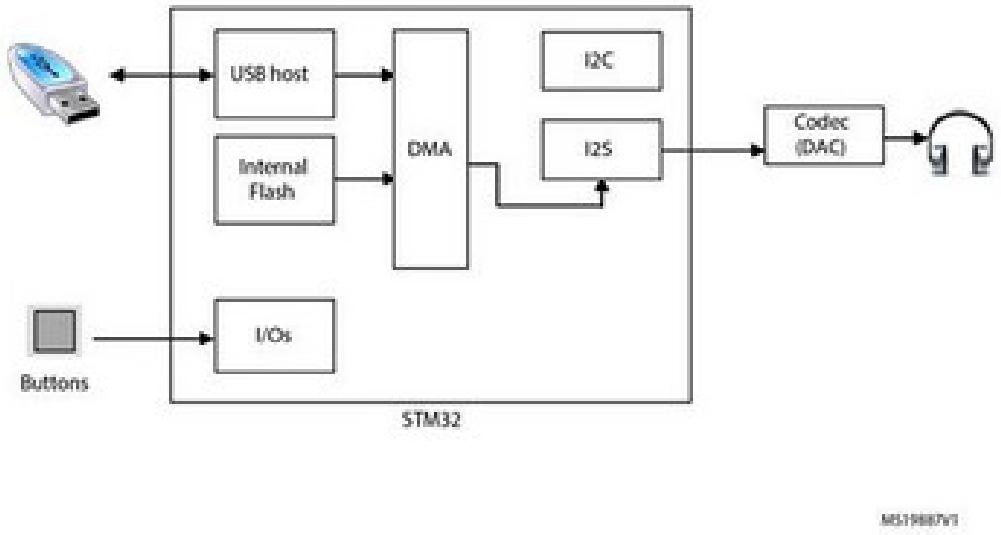
Các tính năng chính của ứng dụng là:

- Audio helix DAC
- Bộ nhớ ngoài – USB
- Bộ gia tốc MEMS.

Các mô-đun STM32 chính được sử dụng là :

- Thiết bị ngoài USB: được cấu hình trên chế độ Host. Lớp thiết bị lưu trữ USB (MSC – Mass Storage Class) được triển khai để truyền và nhận dữ liệu âm thanh đến/đi từ USB.
- Thiết bị ngoại vi I2S: được cấu hình trong chế độ Master Transmitter mode và sử dụng để truyền dữ liệu âm thanh cho DAC.

- DMA được sử dụng để chuyển dữ liệu từ bộ đệm đến các thiết bị ngoại vi, điều này giúp giảm tải cho CPU.
- Thiết bị I2C: được sử dụng để kiểm soát một số các thiết bên ngoài như Codec (DAC) và lấy dữ liệu từ các thiết bị này.
- Thiết bị SPI: kiểm soát các máy đo gia tốc.
- Nhút ấn: được sử dụng để điều khiển ứng dụng.

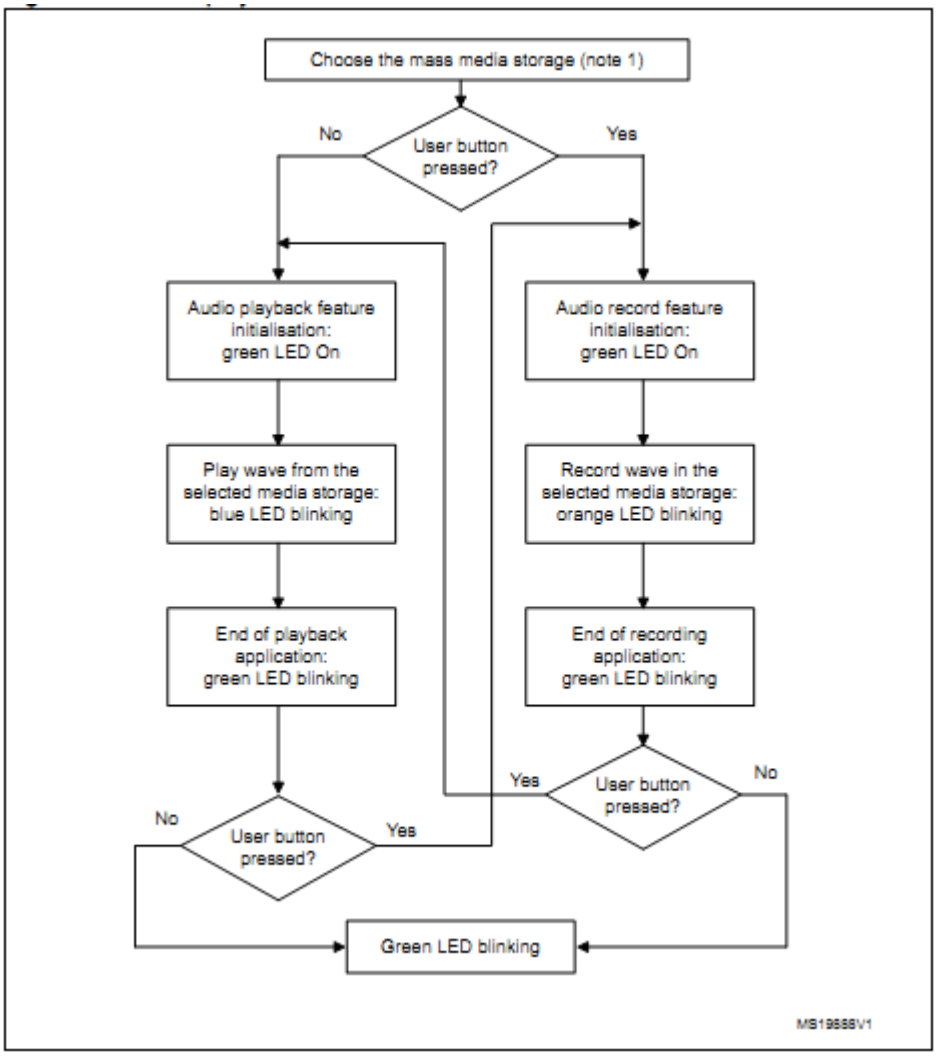


Hình 2: Các mô-đun trong ứng dụng

3.2.Bản mô tả driver của Firmware ứng dụng

Danh sách các lớp chính trong ứng dụng được lưu trữ trong thư mục App:

- Main.c:nó chứa mã khởi tạo và việc bắt đầu ứng dụng.
- Stm32f4xx_it.c: chứa các trình xử lý ngắt cho các ứng dụng.
- Usb_bsp.c: thực hiện gói chỗ trợ cho các thư viện lưu trữ USB.
- Usbh_urc.c : bao gồm các thư viện USB host cho người sử dụng.



Hình 3: Sơ đồ điều khiển ứng dụng

3.3.Cơ chế hoạt động

Bất kỳ tệp tin nào được lưu trữ trên USB key có thể được mở bằng hệ thống tập tin FatFs và được chuyển giao cho các khối (1024 bytes) của bộ nhớ trong SRAM bằng cách sử dụng DMA và giao diện I2S. Và sau đó một DAC âm thanh được kết nối với giao diện I2S để chơi các tập tin lưu trữ âm thanh.

Tên của tệp tin trong USB có thể thay đổi bằng cách thay đổi định nghĩa trong file main.h.

Ứng dụng này đọc tất cả các file sóng từ USB có định dạng như sau:

- Định dạng Audio: PCM (một định dạng dữ liệu sóng nén trong đó mỗi giá trị đại diện cho các biên độ tín hiệu lấy mẫu).
- Tốc độ lấy mẫu: 8000, 11025, 16000, 22050, 44100, 48000 Hz.

- Số lượng bit trên mỗi mẫu : 16 bit (giá trị âm thanh mẫu là trong khoảng 0-1024).
- Số kênh : 2 (sterio).

Các sóng được lấy ra từ USB được phân tích để lấy mẫu cho phù hợp với cấu hình I2S.

Chương trình phát nhạc được quản lý với 2 bộ đệm. Bộ đệm đầu tiên được sử dụng để lưu trữ các dữ liệu sóng lấy từ USB, sử dụng hệ thống tập tin FatFs. Khi bộ đệm này được điền:

- Các DMA gửi nội dung của nó đến các thiết bị ngoại vi I2S để chuyển nó thành mã âm thanh DAC.
- Các dữ liệu từ USB được lưu trữ trong bộ đệm thứ 2.

Sau đó 2 bộ đệm này được đổi chỗ vĩnh viễn cho đến khi quá trình này phát âm thanh này kết thúc. Bất cứ lúc nào, USB bị ngắt kết nối với bảng mạch điều khiển, đèn màu xanh sẽ tắt, chương trình phát nhạc sữ ngừng chạy, đèn đỏ sẽ sáng.

Chương 4. Tài liệu tham khảo

1. UM1021User manual: STM32F105xx, STM32F107xx, STM32F2xx and STM32F4xx USB On-The-Go host and device library
2. Kiến trúc cơ bản của ARM Cortex M3 – www.arm.vn
3. https://helixcommunity.org/
4. AN3997Application note: Audio playback and recording using the STM32F4DISCOVERY.
5. https://www.scribd.com/doc/246256873/UART-c%E1%BB%A7a-stm32f4#scribd