Nhúng và phát âm thanh trên STM32 với DMA và PWM

17 Tháng Một, 2021 Đào Nguyện Lập trình nhúng 9

stm32 stream wav

Âm thanh kĩ thuật số được lưu lại bằng cách đọc các tín hiệu điện áp từ micro thông qua bộ ADC rồi lưu lại dưới dạng 1 con số cụ thể. Đó gọi là lấy mẫu, khoảng thời gian giữa các lần lấy mẫu phải đều nhau và được gọi là chu kì lấy mẫu, lấy nghịch đảo của chu kì ta được tần số lấy mẫu.

Tần số lấy mẫu thường là 44.1Khz (tức trong 1 giây sẽ đọc ADC 44100 lần rồi lưu vào bộ nhớ)

Để phát âm thanh, chúng ta sẽ làm ngược lại quá trình trên tức là đọc các con số đã được lưu trong bộ nhớ rồi phát điện áp tương ứng ra ngoài thông qua bộ DAC, tuy nhiên không phải chip nào cũng có sẵn ngoại vi DAC nên chúng ta có thể giả lập nó bằng phương pháp điều chế độ rộng xung – PWM (đương nhiên DAC fake này sẽ không xịn xò bằng DAC hàng read được)

Khoảng thời gian mà bộ vi điều khiển thay đổi giá trị điện áp phải đúng bằng tần số lấy mẫu thì âm thanh mới chuẩn xác giống như lúc thu âm. Nếu khác thì chúng ta sẽ cho ra các âm thanh bị biến dạng, méo mó

Dữ liệu âm thanh được đóng gói theo các chuẩn file khác nhau (MP3, WAV ...) file âm thanh mp3 là định dạng phổ biến, tuy nhiên nó là file nén, chúng ta sẽ sử dụng các tool chuyển đổi để đưa nó về dạng file

- Tần số lấy mẫu của âm thanh
- Âm thanh này là mono, hay stereo
- Độ sâu của âm thanh (bit depth)

Âm thanh mono, hay stereo

Để đơn giản chúng ta sẽ chỉ làm việc với âm thanh mono (tức là 1 kênh pwm phát âm thanh)

Độ sâu của âm thanh

Độ sâu của âm thanh cũng tương đương với độ phân giải của bộ PWM, ví dụ âm thanh lấy mẫu ở 8bit thì bộ PWM (DAC) cũng phải có độ phân giải 8bit, âm thanh 16bit thì bộ PWM cũng phải có độ phân giải 16bit

Do chúng ta đang giả lập PWM như 1 bộ DAC nên tần số của PWM càng cao thì nó sẽ càng mô phỏng gần chính xác bộ DAC hơn, tuy nhiên trong STM32 tần số tối đa của PWM còn phụ thuộc vào độ phân giải của PWM, ví dụ:

Nếu chọn tần số hoạt động của timer là 72Mhz, độ sâu âm thanh là 16bit thì ta có tần số tối đa của PWM là 72M/ (2 mũ 16) = 1,098 Hz
Nếu chọn độ sâu âm thanh là 8bit thì ta có tần số tối đa của PWM là 72M/(2 mũ 8) 281,250 Hz

Tức là nếu bạn cần phát âm thanh có độ sâu 16bit thì bộ giả lập DAC (tức PWM) chỉ có thể hoạt động ở ~1Khz, đây là con số rất thấp và nó gần như không hoạt động được. Do vậy mình sẽ sử dụng file âm thanh có độ sâu âm là 8bit (chúng ta sẽ dùng tool để chuyển đổi sang dạng 8bit)

Tần số lấy mẫu của âm thanh

Tần số lấy mẫu của âm thanh sẽ tương đường với số lần chúng ta thay đổi độ rộng xung PWM (Pulse). Âm thanh thông thường được lưu ở 44.1Khz (tức trong 1 giây chúng ta phải thay đổi độ rộng xung 44100 lần) nó cũng tương đương với mỗi 1 giây của âm thanh sẽ hao tổn 44.1KB bộ nhớ (mono – 8bit – 44.1Khz)

Bộ nhớ chính xác của STM32F103C8T6 là 128KB nên nếu phát âm 44.1Khz thì chỉ phát được hơn 2s là cùng. Do vậy mình sẽ giảm tần số lấy mẫu của âm thanh xuống khoảng 16Khz (lưu được gần 8s) hoặc 8Khz (lưu được gần 16 giây)

Qua kiểm nghiệm thử thì mình thấy âm thanh mono – 8bit – 16Khz là nghe ổn và đỡ tốn bộ nhớ nhất

Chuẩn bị file âm thanh

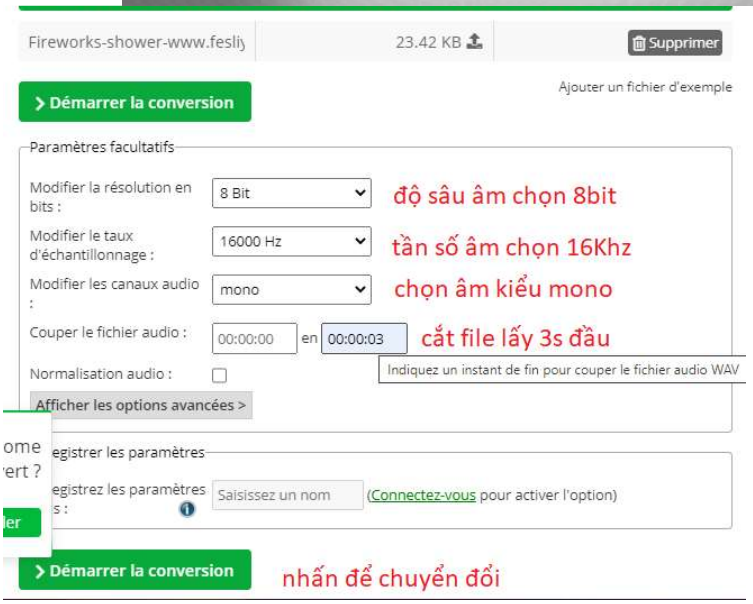
Các bạn tải file âm thanh bất kì về rồi truy cập vào https://audio.online-convert.com/fr/convertir-en-wav để xử lí sang dạng file chúng ta cần

[LCD và GUI] Bài 2: Tìm hiểu màn hình LCD TFT 1.8 inch ST7735 (phần 1)
Bit Band trên STM32
[IoT] Bài 10: Cập nhật chương trình cho esp8266 từ xa qua internet (FOTA)
Bài 0: Làm quen và cài đặt proteus và KeilC 4.0 để lập trình cho các dòng chip 8051
Giao tiếp USB HID trên stm32 với c# window form
Nhúng và phát âm thanh trên STM32 với DMA và PWM
Giao tiếp với module thẻ từ RFID RC522

CHUYÊN MỤC

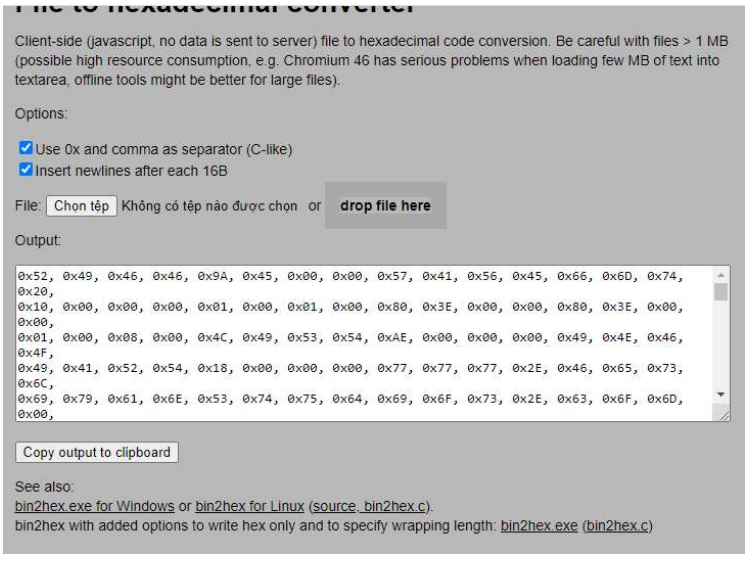
8051
Chưa được phân loại
Ethernet
IoT tutorial
Lập trình nhúng
LCD và GUI
Ma trận LED
Sản phẩm
Vi điều khiển
WIFI-ESP8266

PHẢN HỒI GẦN ĐÂY



Sau đó tải về là chúng ta đã có file âm thanh đuôi WAV và được xử lí sang cấu hình mà chúng ta cần. Bây giờ chúng ta sẽ chuyển file này sang dạng mảng dữ liệu trong ngôn ngữ C bằng công cụ chuyển đổi FILE to Hex http://tomeko.net/online_tools/file_to_hex.php?lang=en

Các bạn kéo thả file âm thanh đuôi WAV vào là có được mã hex của âm, đó chính là mảng dữ liệu mà chúng ta sẽ copy và nhúng vào code



Cấu trúc của tệp tin WAV

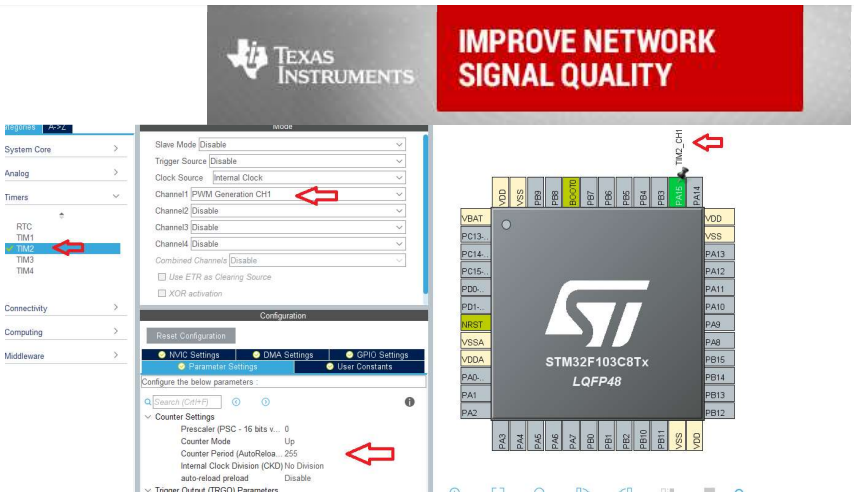
44 byte đầu mô tả các thông tin của file âm thanh, Trong đó, 4 byte đầu luôn là 0x52,0x49,0x46,0x46 . Từ byte thứ 45 là dữ liệu

Các bạn có thể tìm hiểu thêm chi tiết 44 byte header của tệp tin WAV ở trên mạng, trong bài này chúng ta không cần quan tâm lắm vì cứ biết từ byte thứ 45 là dữ liệu là được rồi (vì mặc định là sử dụng file âm thanh mono – 16Khz – 8bit rồi mà)

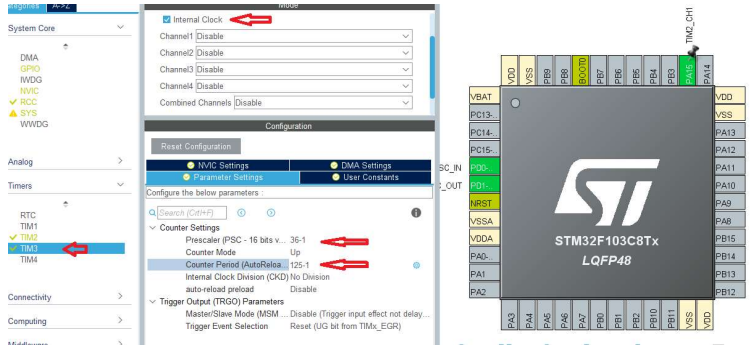
Lập trình

1 cách cực kì đơn giản là sử dụng ngắt timer để đều đặn truy cập thay đổi độ rung xung PWM

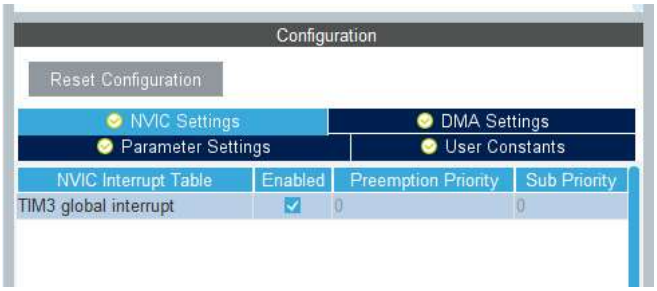
Icd, matrix, thư viện Fontmaker.h
Khách trong [IoT] Bài 10: Cập nhật chương trình cho esp8266 từ xa qua internet (FOTA)
Khách trong [Matrix LED] Bài 4: Hiển thị chữ trên led ma trận 8×32
NK trong [Matrix LED] Bài 15: Điều chế độ rộng xung trên LED ma trận FULL
Khách trong [Matrix LED] Bài 11: Tìm hiểu module LED ma trận P10 Full Color – lập trình led matrix full color
Khách trong [ENC28J60] Bài 7: Giao thức ARP (phần 2)
Khách trong [Matrix LED] Bài 6: Đồng hồ matrix 8×40 hiệu ứng lật trang đơn giản
Khách trong [Matrix LED] Bài 6: Đồng hồ matrix 8×40 hiệu ứng lật trang đơn giản
Khách trong [Matrix LED] Bài 6: Đồng hồ matrix 8×40 hiệu ứng lật trang đơn giản



Bây giờ sẽ cần thêm 1 ngắt timer có tần số ngắt đúng bằng tần số lấy mẫu của âm thanh (16Khz), mình sẽ xài timer 3 và tính toán hệ số chia, bộ đếm sao cho khi timer tràn nó sẽ đếm được 1/16K = 0.0000625s = 0.0625ms = 62.5us
-> Chọn hệ số chia là 36 -> mỗi xung đếm 0.5us -> đếm 125 xung là được 62.5



Đừng quên chuyển sang tab NVIC và kích hoạt ngắt timer 3



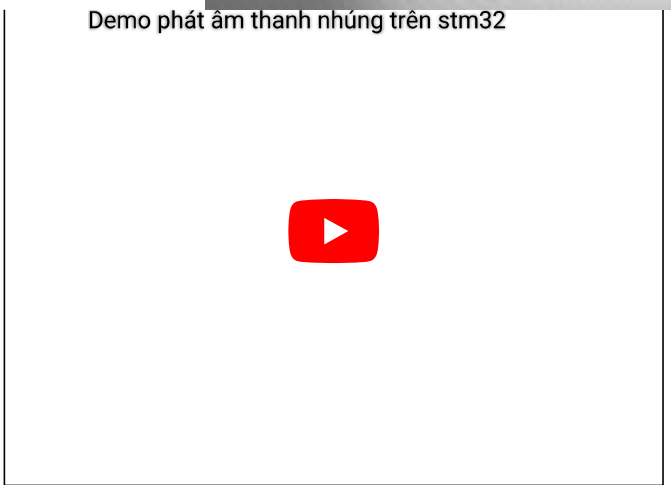
Sinh code và kích hoạt timer, pwm trong hàm main

```
HAL_TIM_PWM_Start(&amp;htim2,TIM_CHANNEL_1); //cho phep PWM (gia lap DAC)
HAL_TIM_Base_Start_IT(&amp;htim3);            //cho phep ngat TIM3 hoat dong
```

Hàm ngắt timer 3 có nhiệm vụ lấy dữ liệu trong mảng âm thanh đưa vào thay đổi độ rộng xung của PWM, hãy nhớ dữ liệu bắt đầu từ 44 nhé ! Mình sẽ đặt tên cho mảng dữ liệu chứa âm thanh là uint8_t const amthanh[]={.....};

```
const uint8_t amthanh[]={
0x52, 0x49, 0x46, 0x46, 0xB8, 0x16, 0x01, 0x00, 0x57, 0x41, 0x56, 0x45, 0x66, 0x6D, 0x74, 0x20,
0x10, 0x00, 0x00, 0x00, 0x01, 0x00, 0x01, 0x00, 0x80, 0x3E, 0x00, 0x00, 0x80, 0x3E, 0x00, 0x00,
0x01, 0x00, 0x08, 0x00, 0x4C, 0x49, 0x53, 0x54, 0x1A, 0x00, 0x00, 0x00, 0x49, 0x4E, 0x46, 0x4F,
0x49, 0x53, 0x46, 0x54, 0x0E, 0x00, 0x00, 0x00, 0x4C, 0x61, 0x76, 0x66, 0x35, 0x38, 0x2E, 0x35,
0x34, 0x2E, 0x31, 0x30, 0x30, 0x00, 0x64, 0x61, 0x74, 0x61, 0x71, 0x16, 0x01, 0x00, 0x80, 0x80,
0x7F, 0x7F, 0x7F, 0x80, 0x80, 0x7F, 0x7F, 0x81, 0x7F, 0x7E, 0x7F, 0x81, 0x7E, 0x7E, 0x80, 0x82,
0x81, 0x7C, 0x7F, 0x87, 0x7E, 0x74, 0x89, 0x88, 0x72, 0x7B, 0x90, 0x7A, 0x72, 0x8E, 0x83, 0x6F,
0x82, 0x8F, 0x72, 0x7A, 0x8F, 0x7B, 0x72, 0x8A, 0x8B, 0x6F, 0x83, 0x8B, 0x76, 0x7A, 0x8D, 0x7D,
0x75, 0x8B, 0x81, 0x75, 0x84, 0x8A, 0x72, 0x80, 0x8E, 0x77, 0x75, 0x8E, 0x83, 0x6E, 0x8A, 0x8A,
```

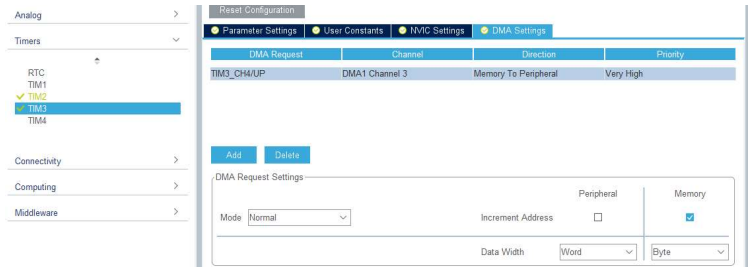
```
uint32_t count=44;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) //16Khz timer
{
   TIM2->CCR1=amthanh[count++];
   if(count>=sizeof(amthanh)) count=44; count=44;
}
```



Sử dụng với DMA

DMA sẽ giúp giải phóng CPU khỏi ngắt timer giúp tối ưu hiệu suất của chip, các bạn có thể thoải mái làm việc khác trong khi vẫn đang phát nhạc. Lúc này, timer3 thay vì tạo ra tín hiệu ngắt vào CPU thì sẽ tạo ra tín hiệu trigger vào DMA

Tắt ngắt Timer 3 và vào tab DMA Setting để bật DMA TIM UP và cấu hình như sau:



Hướng di chuyển của data rõ ràng là M2P rồi (memorry to peripheral), ưu tiên để cao nhất.
Dộ dài của mảng dữ liệu là uint8_t (tức là 1byte) còn độ dài của thanh ghi độ rộng xung là 4byte (word) nên chúng ta phải setup cho đúng nhé !

Sinh code và xóa hàm ngắt timer đi ! trong hàm main tiến hành khởi tạo lại:

```
1 HAL_TIM_PWM_Start(&htim2,TIM_CHANNEL_1); //cho phep PWM (gia lap DAC)
2 __HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_UPDATE); //bat DMA tim up cho timer3
3 HAL_DMA_Start_IT(&hdma_tim3_up,(uint32_t)(amthanh+44),(uint32_t)&(TIM2->CCR1),sizeof(amthan
```

Vậy là xong, rất đơn giản phải không ! ngoài ra các bạn có thể bắt 1 tín hiệu callback khi âm thanh phát hết bằng hàm tạo callback

```
1 HAL_DMA_RegisterCallback(&hdma_tim3_up,HAL_DMA_XFER_CPLT_CB_ID,&DMA_ngat); //connect callbc
```

Như vậy hàm DMA_ngat sẽ tự động được gọi mỗi khi DMA truyền xong data (phát xong âm thanh), giờ minh sẽ viết hàm DMA_ngat để cho nó phát lại âm thanh khi phát xong

```
1 void DMA_ngat(DMA_HandleTypeDef * _hdma) //ngat truyen xong data
2 {
3  HAL_DMA_Abort(&hdma_tim3_up);            //kết thúc truyền
4  HAL_DMA_Start_IT(&hdma_tim3_up,(uint32_t)(amthanh+44),(uint32_t)&(TIM2->CCR1),sizeof(amtha
5 }
```

Chúc các bạn thành công !

Related posts:

Lập trình LED dây UCS1903 với stm32f103c8t6

Thuật toán FFT cho mạch nháy theo nhạc

Giao tiếp USB HID trên stm32 với c# window form

Hiệu chỉnh Gamma

Từ tác giả:

Nếu có bất kì thắc mắc nào trong bài viết, vui lòng để lại comment dưới mỗi bài ! Mình sẽ không trả lời thắc mắc của các bạn ở facebook hay email !

Nếu trong phần code bạn nhìn thấy nhưng thứ kiểu như & thì đó là lỗi hiển thị, cụ thể 3 kí tự < > & bị biến đổi thành như thế
& là &
< là <
> là >

Giới thiệu Đào Nguyện › 71 bài viết

DIY,chế cháo, viết blog chia sẽ kiến thức về lập trình,điện tử - IoT. Rất mong được giao lưu, kết bạn với các bạn cùng đam mê. Địa chỉ Facebook: https://www.facebook.com/nguyendao207

9 BÌNH LUẬN

Dat

20 THÁNG MỘT, 2021 TẠI 5:10 CHIỀU

anh ơi cho em hỏi 2 hàm này viết vào đâu vậy anh.
HAL_DMA_RegisterCallback(&hdma_tim3_up,HAL_DMA_XFER_CPLT_CB_ID,&DMA_ngat); và hàm void DMA_ngat(DMA_HandleTypeDef * _hdma)

và cho em hỏi đối số như này &hdma_tim3_up hay là như này ạ &hdma_tim3_ch4_up.

TRẢ LỜI

Khách

28 THÁNG MỘT, 2021 TẠI 3:33 SÁNG

hàm DMA_ngat là 1 hàm do bạn tự tạo ra
Hàm HAL_DMA_RegisterCallback dùng để đăng kí rằng khi có sự kiện ngắt truyền xong DMA thì hãy tự động gọi hàm DMA_ngat
đối số hdma_tim3_up hay hdma_tim3_ch4_up còn tùy thuộc vào MCU bạn dùng

TRẢ LỜI

Khách

2 THÁNG HAI, 2021 TẠI 12:43 SÁNG

Mj cảm ơn bạn

TRẢ LỜI

Khách

2 THÁNG HAI, 2021 TẠI 12:45 SÁNG

Khách

2 THÁNG HAI, 2021 TẠI 12:46 SÁNG

2 hàm này vị trí nằm ở đâu vậy bạn

TRẢ LỜI

Khách

29 THÁNG MỘT, 2021 TẠI 12:23 CHIỀU

Cảm ơn bài viết của anh

TRẢ LỜI

Khách

1 THÁNG HAI, 2021 TẠI 12:07 CHIỀU

Chào anh, hôm nào a có thể viết bài về giao tiếp thẻ sd card và các hàm thao tác với files được không ạ

TRẢ LỜI

Khách

20 THÁNG HAI, 2021 TẠI 9:59 CHIỀU

Cảm ơn bạn

TRẢ LỜI

Khách

12 THÁNG TÁM, 2021 TẠI 10:23 CHIỀU

cho mình hỏi có nhất thiết tần số lấy mẫu ADC phải là 44.1Khz ko nhỉ ? nếu như lấy cao hơn thì chất lượng âm tốt hơn đúng k nhỉ ? có nhược điểm và ưu điểm gì khi lấy mẫu tần số cao hơn k nhỉ

TRẢ LỜI

Để lại bình luận

Nhập bình luận ở đây...

IOT47 BLOG

Blog chuyên chia sẽ kiến thức về IoT, lập trình nhúng, được lập vào tháng 8/2019 bởi Đào Nguyện

LIÊN HỆ – HỖ TRỢ

daonguyen20798@gmail.com
0394733311
F fb.com/nguyendao207

LIÊN KẾT

▶ Kênh DIY, làm mạch điện tử
▶ Kênh tổng hợp video từ blog
▶ https://qstech.vn/

ỦNG HỘ

Nếu thấy cái bài viết hữu ích, các bạn có thể ủng hộ cho mình tại tài khoản:
VIETCOMBANK CHƯƠNG DƯƠNG
STK: 0541000289275
DAO VAN NGUYEN