

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN - ĐIỆN TỬ
Khoa Điện tử



BÁO CÁO MÔN HỌC VI XỬ LÝ

ĐỀ TÀI:

**THIẾT KẾ MẠCH TRUYỀN SỐ LIỆU SỬ
DÙNG ÂM THANH**

GVHD: TS. Hàn Huy Dũng

Mã lớp: 154166

Sinh viên thực hiện

Nguyễn Trung Mạnh	20224054
Ngô Tiến Lộc	20224040
Trần Minh Quân	20224112
Ngô Thanh Lâm	20223713

LỜI NÓI ĐẦU

Trong thời đại công nghệ ngày càng phát triển, nhu cầu truyền thông tin giữa các thiết bị vi điều khiển ngày càng trở nên phổ biến và đa dạng. Bên cạnh các phương thức truyền thông truyền thống như UART, SPI, I2C, hay không dây như Wi-Fi và Bluetooth, việc khai thác sóng âm – một phương tiện truyền tín hiệu phi điện từ – mở ra những hướng đi mới cho các ứng dụng trong môi trường đặc thù, tiết kiệm chi phí hoặc đơn giản về mặt phần cứng.

Đề tài "*Xây dựng mạch truyền số liệu dùng âm thanh giữa hai ESP32*" được thực hiện với mục tiêu nghiên cứu, thiết kế và xây dựng một hệ thống truyền dữ liệu đơn giản sử dụng sóng âm giữa hai vi điều khiển ESP32. Hệ thống sử dụng loa để phát và micro để thu tín hiệu âm thanh mang dữ liệu, kết hợp với các kỹ thuật điều chế và giải điều chế tín hiệu cơ bản để đảm bảo việc truyền – nhận dữ liệu hiệu quả.

Qua đề tài này, nhóm thực hiện mong muốn tìm hiểu sâu hơn về nguyên lý truyền dữ liệu bằng sóng âm, ứng dụng kỹ thuật xử lý tín hiệu cơ bản trên ESP32, đồng thời nâng cao kỹ năng thiết kế, lập trình và tích hợp phần cứng – phần mềm trong các hệ thống nhúng. Bọn em hy vọng rằng kết quả của đề tài sẽ là tiền đề cho các nghiên cứu và ứng dụng mở rộng trong tương lai, đặc biệt trong các môi trường hạn chế kết nối không dây hoặc yêu cầu mức độ đơn giản cao.

Cấu trúc dự án bao gồm 5 chương:

Chương 1: Trình bày tổng quan đề tài về đặt vấn đề, phương pháp thực hiện, đưa ra mục tiêu, phạm vi của đề tài cũng như các tiêu chí đánh giá hệ thống.

Chương 2: Trình bày mô hình tổng quan hệ thống, đưa ra các yêu cầu chức năng, phi chức năng của hệ thống.

Chương 3: Trình bày phương pháp thiết kế triển khai hệ thống.

Chương 4: Triển khai sản phẩm.

Chương 5: Kiểm thử hệ thống bằng các bài test, đưa ra các đánh giá về hệ thống.

MỤC LỤC

LỜI NÓI ĐẦU	1
MỤC LỤC HÌNH VẼ.....	5
MỤC LỤC BẢNG	6
CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI.....	9
1.1. Đặt vấn đề:.....	9
1.2. Ý tưởng, mục tiêu và giải pháp:	9
1.2.1. Ý tưởng:	9
1.2.2. Mục tiêu đề tài:.....	10
1.2.3. Giải pháp thực hiện:	10
1.2.4. Phạm vi bài toán:.....	11
CHƯƠNG 2. PHÂN TÍCH YÊU CẦU KỸ THUẬT	12
2.1. Cơ sở lý thuyết:	12
2.1.1. ESP32:.....	12
2.1.2. GPIO:	14
2.1.3. ADC:	18
2.1.4. I2S:	25
2.1.5. I2C:.....	27
2.1.6. PWM:	32
2.1.7. DMA:	32
2.1.8. Mã hóa Reed-Solomon.....	35
2.1.9. DSS (Direct Sequence Spread)	35
2.2. Nguyên lý hoạt động:	37
2.2.1. Nguyên tắc truyền dữ liệu bằng sóng âm:.....	38
2.2.2. Truyền phát, thu nhận dữ liệu bằng sóng âm:.....	39
2.2.2.1. Phát dữ liệu:	39
2.2.2.2. Thu dữ liệu:	39
2.2.3. Các phương pháp điều chế tín hiệu:.....	40

2.2.4. Mã hóa và điều chế tín hiệu:	41
2.2.5. Giải điều chế và xử lý tín hiệu:	42
2.3. Yêu cầu chức năng:.....	44
2.4. Yêu cầu phi chức năng:	44
CHƯƠNG 3. THIẾT KẾ HỆ THỐNG.....	46
3.1. Tổng quan hệ thống:.....	46
3.2. Phương án thiết kế:.....	48
3.2.1. Thiết kế phần cứng:.....	48
3.2.2. Thiết kế phần mềm:.....	49
CHƯƠNG 4. TRIỂN KHAI SẢN PHẨM	53
4.1. Triển khai phần cứng:	53
4.2. Triển khai phần mềm:	54
4.2.1. Đầu phát:	54
4.2.1.1. Tiền xử lý dữ liệu:	55
4.2.1.2. Thêm mã sửa lỗi (Reed-Solomon)	56
4.2.1.3. Chuyển dữ liệu thành chỉ số âm thanh (tone indices).....	58
4.2.1.4. Chuyển tone index sang tần số Hz:	59
4.2.1.5. Tạo chuỗi âm thanh	61
4.2.2. Đầu thu	63
4.2.2.1. Thu tín hiệu âm thanh - FFT và phổ công suất	64
4.2.2.2. Chuẩn hóa phổ (Quantization)	65
4.2.2.3. Phát hiện tông âm (Tone Detection)	66
4.2.2.4. Kết hợp nibble → byte	66
4.2.2.5. Giải mã Reed-Solomon	66
4.2.2.6. Khử trải phổ DSS	67
4.2.2.7. Xuất kết quả cuối cùng.....	67
CHƯƠNG 5. KIỂM THỬ	68

MỤC LỤC HÌNH VẼ

Hình 2.1 Sơ đồ PinOut của ESP32.....	12
Hình 2.2 Sơ đồ chân GPIO của ESP32	14
Hình 2.3 Nguyên lý hoạt động GPIO.....	15
Hình 2.4 GPIO MUX	15
Hình 2.5 Các chế độ hoạt động GPIO.....	16
Hình 2.6 GPIO output HIGH	16
Hình 2.7 GPIO output LOW	17
Hình 2.8 GPIO Mode open drain	17
Hình 2.9 Sơ đồ hoạt động ADC	18
Hình 2.10 Quá trình lấy mẫu ADC	19
Hình 2.11 Quá trình lượng tử hóa ADC.....	19
Hình 2.12 Flash ADC.....	20
Hình 2.13 Sơ đồ mạch Fash ADC.....	22
Hình 2.14 Sơ đồ mạch Dual-Slope ADC	22
Hình 2.15 Sơ đồ nguyên lý SAR.....	23
Hình 2.16 Mã hóa trong ADC.....	24
Hình 2.17 Nguyên lý hoạt động DMA.....	33
Hình 2.18 UART hoạt động	33
Hình 2.19 Khung truyền cơ bản của một UART	34
Hình 2.20 UART kênh truyền dữ liệu.....	37
Hình 2.21 UART truyền dữ liệu.....	38
Hình 2.22 UART Kênh nhận dữ liệu	38
Hình 3.1 Sơ đồ khối hệ thống	46
Hình 3.2 Sơ đồ tổng quan thiết kế mạch.....	48
Hình 3.3 Sơ đồ trạng thái mạch phát tín hiệu	49
Hình 3.4 Sơ đồ trạng thái mạch thu tín hiệu	50
Hình 3.5 Lưu đồ thuật toán phân phát.....	51
Hình 3.6 Lưu đồ thuật toán phân thu	51
Hình 3.7 Quy trình hoạt động của phân phát... Error! Bookmark not defined.	
Hình 4.1 Mạch triển khai thực tế.....	53
Hình 4.2 QR Source code phân phát.....	63
Hình 4.3 QR Source code phân thu.....	67

MỤC LỤC BẢNG

Bảng 2.1 Thông số cơ bản của ESP32 WROOM DEVKIT	13
Bảng 2.2 Các phương pháp điều chế tín hiệu	40
Bảng 2.3 Bảng so sánh các loại điều chế	41

ĐÓNG GÓP TỪNG THÀNH VIÊN

	Công việc		Tỉ lệ đóng góp
Nguyễn Trung Mạnh	Phần thu		25%
Ngô Thanh Lâm		<ul style="list-style-type: none"> - Nghiên cứu thuật toán xử lý tín hiệu thu: FFT, sửa lỗi ECC, Reed-solomon decode - Mô hình hóa các bước đầu thu thành sơ đồ - Triển khai giải điều chế MFSK - Phân tích kết quả thu: Đo tín hiệu sau khi thu qua mic, phân tích phổ tần số bằng matlab 	25%
Trần Minh Quân	Phần phát	<ul style="list-style-type: none"> - Triển khai các thuật toán mã hóa dữ liệu: DSS (Direct Sequence Spreading), ECC (Reed-Solomon), tách nibble → tone index. - Hiểu và chỉnh sửa thuật toán tạo tần số, ánh xạ tone index → tần số phát. - Phân tích kết quả phát: test thực tế với loa, đo phổ tín hiệu bằng công cụ matlab. 	25%
Ngô Tiến Lộc		<ul style="list-style-type: none"> - Tổng hợp cách hoạt động toàn bộ phần phát: pipeline xử lý dữ liệu, âm thanh. - Phân tích các ngoại vi được dùng: PWM, loa, mã hóa bit. - Tham gia hỗ trợ mã hóa 	25%

		<p>DSS, ECC, tone index (cùng Quân), đảm bảo hiệu và sửa lỗi.</p> <p>- Thử nghiệm phát từng giai đoạn (chỉ DSS, chỉ ECC), so sánh phổ để chứng minh hiệu quả thuật toán.</p>	
--	--	--	--

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

1.1. Đặt vấn đề:

Trong các hệ thống nhúng và Internet of Things (IoT), việc truyền dữ liệu giữa các thiết bị là một thành phần quan trọng, quyết định hiệu quả và tính ứng dụng của hệ thống. Các phương thức truyền thông phổ biến hiện nay như Wi-Fi, Bluetooth, Zigbee... đều sử dụng sóng điện từ và yêu cầu phần cứng chuyên biệt, đi kèm với chi phí, mức tiêu thụ năng lượng và sự phụ thuộc vào môi trường truyền dẫn.

Tuy nhiên, trong một số tình huống:

- Môi trường có nhiễu từ mạnh, khó triển khai bằng RF.
- Khoảng cách truyền ngắn, yêu cầu kết nối đơn giản.
- Hệ thống cần chi phí thấp, chủ yếu phục vụ học tập, thí nghiệm rộng rãi.

Từ thực tế đó, nhóm nghiên cứu đề xuất xây dựng một hệ thống truyền số liệu giữa hai vi điều khiển ESP32 sử dụng âm thanh làm kênh truyền dẫn. Hệ thống bao gồm một ESP32 phát dữ liệu dưới dạng âm thanh thông qua loa, và một ESP32 khác thu nhận âm thanh bằng micro để giải mã lại dữ liệu gốc.

Đề tài không chỉ giúp tìm hiểu khả năng truyền dữ liệu qua sóng âm – một hướng đi ít phổ biến – mà còn rèn luyện kỹ năng lập trình xử lý tín hiệu, giao tiếp giữa các thiết bị, và khai thác tối đa tiềm năng phần cứng của ESP32. Đây cũng là bước đầu để mở rộng sang các hướng ứng dụng khác như: truyền dữ liệu qua giọng nói, liên lạc giữa thiết bị mà không cần mạng, hay các hệ thống tương tác âm thanh – vi điều khiển

1.2. Ý tưởng, mục tiêu và giải pháp:

1.2.1. Ý tưởng:

Truyền dữ liệu đơn giản bằng âm thanh dùng ESP32 thay vì các giao thức không dây như WiFi, Lora, ...

Mạch phát sẽ phát tín hiệu đã được điều chế bằng FSK dưới dạng âm thanh qua loa.

Mạch thu sẽ nhận tín hiệu từ micro, thực hiện giải điều chế và đưa tín hiệu đã giải điều chế ra màn hình hiển thị.

1.2.2. Mục tiêu đề tài:

- Xây dựng một hệ thống truyền dữ liệu giữa hai vi điều khiển ESP32 sử dụng tín hiệu âm thanh.
- Thiết kế và hiện thực quá trình mã hóa dữ liệu thành âm thanh ở phía phát (ESP32 + loa).
- Thiết kế hệ thống thu và giải mã tín hiệu âm thanh ở phía thu (ESP32 + micro).
- Áp dụng các kỹ thuật điều chế và giải điều chế đơn giản như FSK hoặc ASK.
- Đảm bảo hệ thống hoạt động ổn định trong khoảng cách truyền ngắn (dưới 1m).
- Tận dụng phần cứng đơn giản, chi phí thấp, dễ chế tạo và triển khai.
- Góp phần mở rộng hướng nghiên cứu truyền dữ liệu bằng phương pháp phi truyền thông (sóng âm).

1.2.3. Giải pháp thực hiện:

Hệ thống sẽ hoạt động như thế nào?

- Hệ thống gồm hai vi điều khiển ESP32, trong đó một ESP32 đóng vai trò phát dữ liệu và một ESP32 khác thu dữ liệu. ESP32 phát sẽ mã hóa dữ liệu nhị phân thành tín hiệu âm thanh (dạng sóng vuông hoặc điều chế đơn giản) và phát ra qua loa. ESP32 thu sẽ sử dụng micro để ghi nhận âm thanh, xử lý tín hiệu và giải mã để thu lại dữ liệu ban đầu. Dữ liệu sau khi thu sẽ được hiển thị hoặc xử lý tùy mục đích sử dụng.

Khi nào hệ thống sẽ hữu ích?

- Hệ thống đặc biệt hữu ích trong các môi trường không thể dùng Wi-Fi/Bluetooth như nơi bị nhiễu sóng điện từ, không gian kín, hay trong các ứng dụng cần truyền dữ liệu tạm thời, đơn giản mà không muốn thiết lập mạng phức tạp. Ngoài ra, nó còn phù hợp trong giáo dục hoặc nghiên cứu các phương thức truyền dữ liệu thay thế, sáng tạo.

Ai lại người cần hệ thống này?

- Giải pháp này phù hợp với sinh viên, giảng viên, kỹ sư, và nhà nghiên cứu trong lĩnh vực hệ thống nhúng, truyền thông dữ liệu và tự động hóa. Nó cũng hữu ích cho các cá nhân/đơn vị muốn phát triển các thiết bị giao tiếp đơn giản qua âm thanh, ví dụ trong thiết bị thông minh, truyền tín hiệu giữa các robot, hay thiết bị IoT chi phí thấp.

1.2.4. Phạm vi bài toán:

- Truyền dữ liệu ở khoảng cách ngắn (dưới 1 mét) trong môi trường không quá nhiễu âm.
- Sử dụng tín hiệu âm thanh đơn giản (âm thanh nghe được, tần số cố định hoặc thay đổi) để truyền dữ liệu nhị phân.
- Chỉ tập trung vào giao tiếp **một chiều**: từ ESP32 phát → ESP32 thu.
- Dữ liệu truyền là dạng chuỗi ký tự hoặc số đơn giản, tốc độ thấp.
- Áp dụng kỹ thuật điều chế/giải điều chế cơ bản, không dùng thuật toán xử lý tín hiệu phức tạp (như FFT, lọc thích nghi...).
- Sử dụng phần cứng rẻ tiền: micro analog thường, loa mini, không sử dụng thiết bị thu/phát âm thanh chuyên dụng.
- Không giải quyết các vấn đề về bảo mật, mã hóa dữ liệu hay chống nhiễu chuyên sâu.

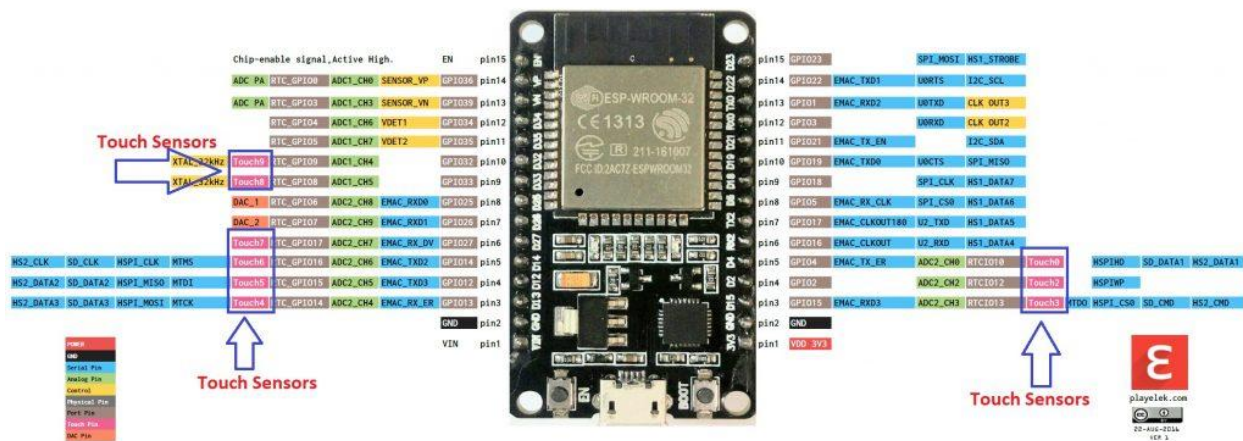
CHƯƠNG 2. PHÂN TÍCH YÊU CẦU KỸ THUẬT

2.1. Cơ sở lý thuyết:

Vi điều khiển và các ngoại vi:

2.1.1. ESP32:

DOIT ESP32 DEVKIT V1 PINOUT



Hình 2.1 Sơ đồ PinOut của ESP32

ESP32 là một bộ vi điều khiển thuộc danh mục vi điều khiển trên chip công suất thấp và tiết kiệm chi phí. Hầu hết tất cả các biến thể ESP32 đều tích hợp Bluetooth và Wi-Fi chế độ kép, làm cho nó có tính linh hoạt cao, mạnh mẽ và đáng tin cậy cho nhiều ứng dụng.

Nó là sự kế thừa của vi điều khiển NodeMCU ESP8266 phổ biến và cung cấp hiệu suất và tính năng tốt hơn. Bộ vi điều khiển ESP32 được sản xuất bởi Espressif Systems và được sử dụng rộng rãi trong nhiều ứng dụng khác nhau như IoT, robot và tự động hóa.

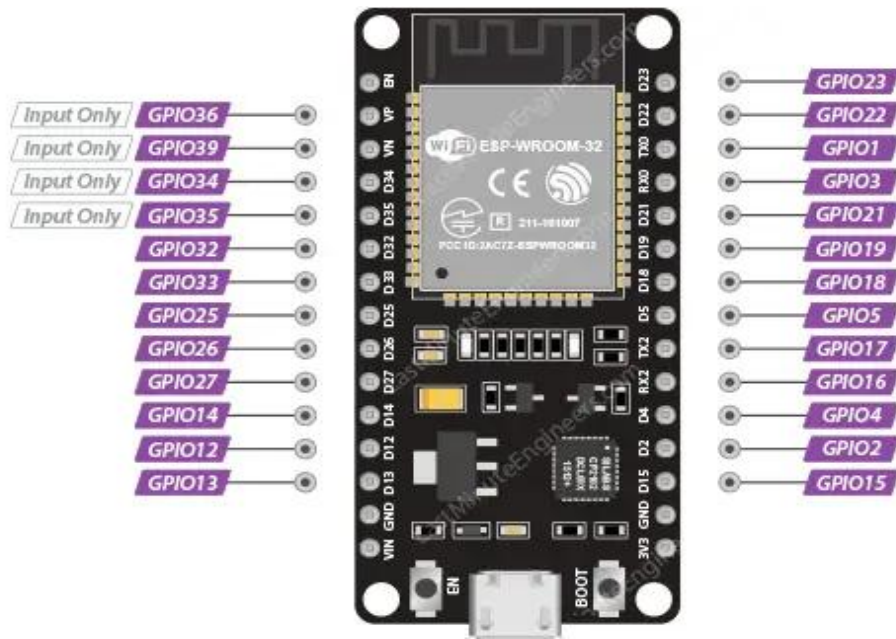
ESP32 cũng được thiết kế để tiêu thụ điện năng thấp, lý tưởng cho các ứng dụng chạy bằng pin. Nó có hệ thống quản lý năng lượng cho phép nó hoạt động ở chế độ ngủ và chỉ thức dậy khi cần thiết, điều này có thể kéo dài tuổi thọ pin rất nhiều.

Thông số	Giá trị / Mô tả
Vi xử lý	Dual-core Xtensa® 32-bit LX6
Tốc độ xung nhịp	Lên đến 240 MHz
RAM	~520 KB SRAM
Flash	Hỗ trợ ngoài (thường 4 MB hoặc 8 MB tùy module)
GPIO	Tối đa 34 chân GPIO (tùy theo module)
ADC	12-bit, lên đến 18 kênh
DAC	8-bit, 2 kênh
PWM	Có, điều khiển được nhiều kênh
Giao tiếp	UART, SPI, I2C, I2S, CAN
Điện áp hoạt động	3.0V – 3.6V (thường 3.3V)

Bảng 2.1 Thông số cơ bản của ESP32 WROOM DEVKIT

ESP32 được lựa chọn làm vi điều khiển chủ đạo cho mạch truyền dữ liệu bằng âm thanh nhờ khả năng tích hợp bộ xử lý mạnh mẽ với CPU dual-core 240 MHz, hỗ trợ đa nhiệm và xử lý tín hiệu phức tạp như mã hóa FSK, điều khiển PWM và giao tiếp I2S với micro âm thanh. Bên cạnh đó, ESP32 có sẵn phần cứng hỗ trợ giao tiếp I2S chất lượng cao và PWM độ phân giải cao, giúp việc thu nhận và phát sóng âm thanh được chính xác và ổn định. Thêm vào đó, ESP32 sở hữu giá thành thấp, phổ biến rộng rãi cùng cộng đồng phát triển lớn, kèm theo nhiều thư viện hỗ trợ như ggwave, giúp tăng tốc quá trình phát triển và triển khai sản phẩm một cách hiệu quả và tiết kiệm.

2.1.2. GPIO:

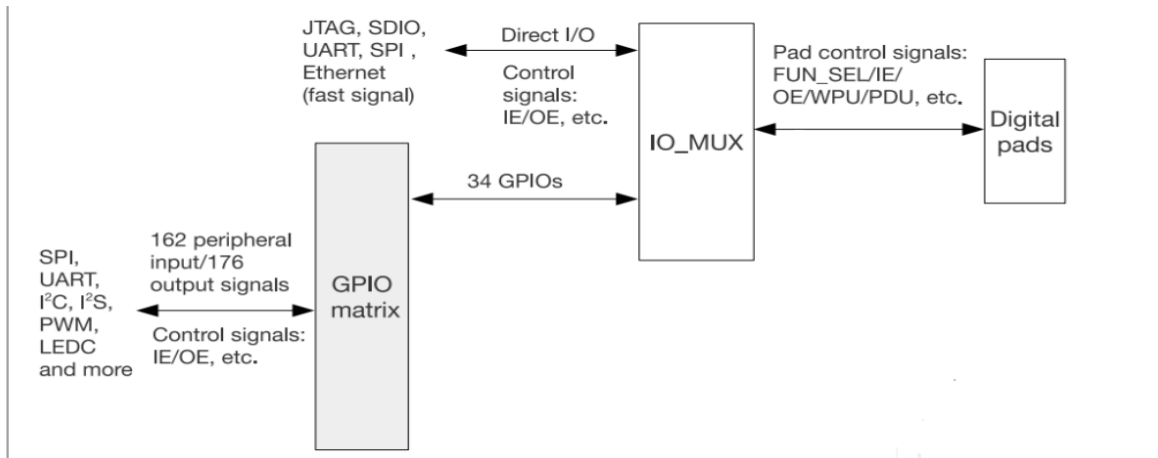


Hình 2.2 Sơ đồ chân GPIO của ESP32

Giới thiệu:

- **GPIO (General Purpose Input/Output)** là các chân tín hiệu đa năng trên vi điều khiển, dùng để giao tiếp và điều khiển các thiết bị ngoại vi bên ngoài như cảm biến, nút bấm, đèn LED, động cơ, v.v.
- Các chân GPIO có thể được cấu hình ở chế độ **input** để đọc tín hiệu hoặc **output** để điều khiển thiết bị.
- GPIO trên ESP32 có khả năng hỗ trợ các chức năng đặc biệt như PWM, ADC, DAC, I2C, SPI, UART, giúp linh hoạt trong thiết kế hệ thống.

Nguyên tắc hoạt động:

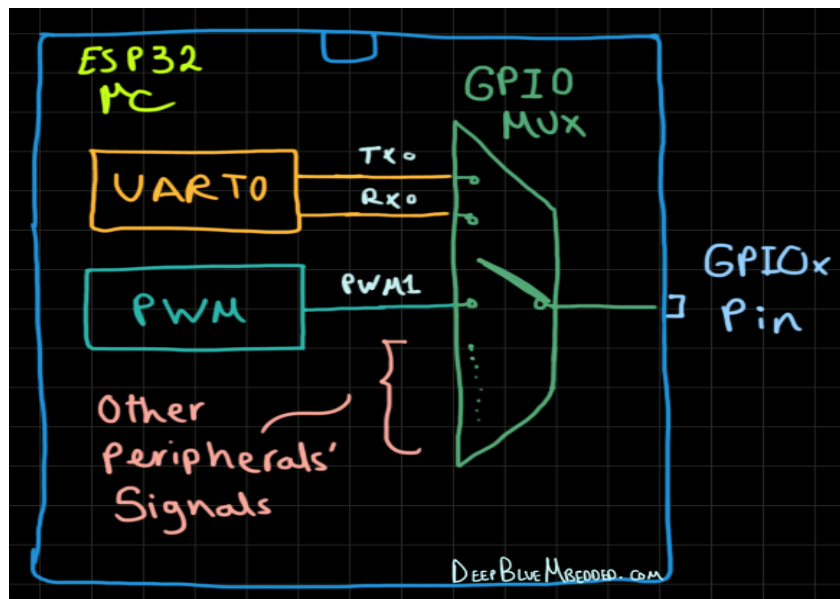


Hình 2.3 Nguyên lý hoạt động GPIO

Các chân GPIO của ESP32 được kết nối thông qua một bộ MUX giúp lập

trình viên lựa chọn các chức năng phù hợp cho dự án.

Bộ MUX GPIO trên ESP32 là thành phần cho phép chọn lựa chức năng cho từng chân GPIO. Mỗi chân có thể đảm nhận nhiều nhiệm vụ khác nhau như



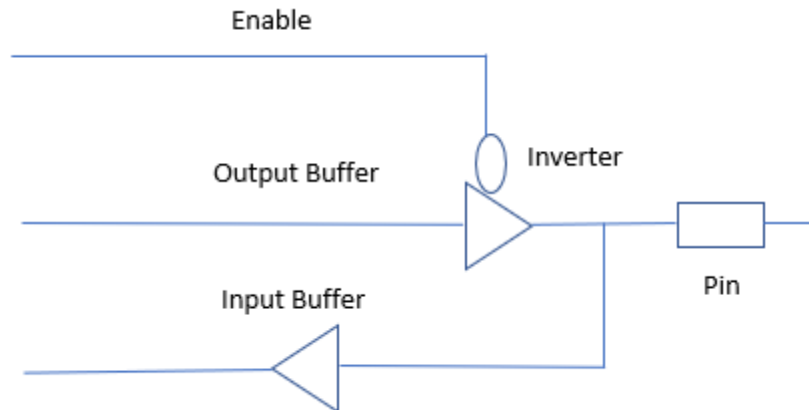
làm đầu vào hoặc đầu ra số, PWM, ADC, hoặc giao tiếp UART, SPI, I2C.

Hình 2.4 GPIO MUX

Bộ MUX hoạt động bằng cách chuyển đổi chức năng chân GPIO dựa trên tín hiệu điều khiển nội bộ, giúp tận dụng tối đa khả năng của chân và giảm số lượng chân vật lý cần thiết. Nhờ vậy, hệ thống có thể linh hoạt hơn và tiết kiệm tài nguyên phần cứng.

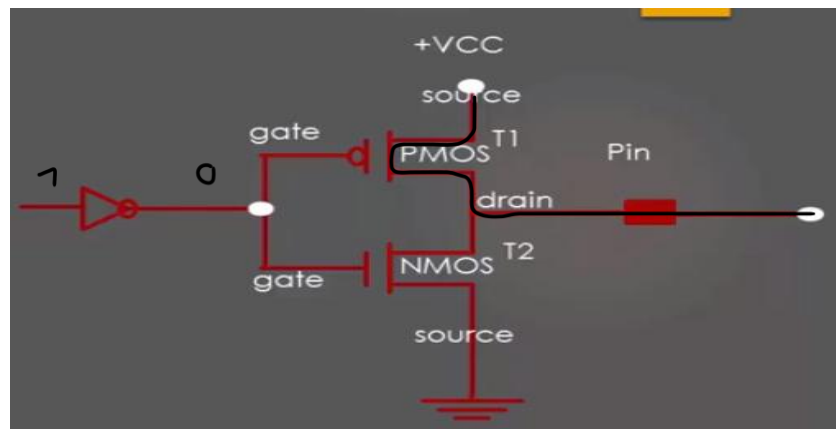
Phương pháp hoạt động chung của chế độ GPIO:

Mỗi chân bao gồm hai bộ đệm - bộ đệm đầu vào và đầu ra cùng với một dòng enable. Khi dòng enable là 0, bộ đệm đầu ra được kích hoạt và bộ đệm đầu vào sẽ bị vô hiệu hóa.



Hình 2.5 Các chế độ hoạt động GPIO

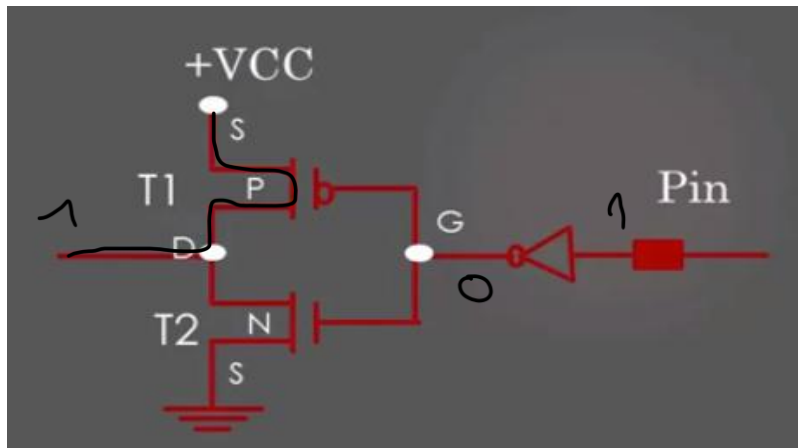
Khi dòng enable là 0, bộ đệm đầu ra được kích hoạt và bộ đệm đầu vào sẽ



Hình 2.6 GPIO output HIGH

bị vô hiệu hóa.

Khi chúng ta ghi 1 từ phần mềm, inverter đầu tiên sẽ đặt nó là 0 và mạch PMOS (T1) sẽ hoạt động (NMOS (T2) sẽ bị vô hiệu hóa) do đó chân sẽ được kéo lên VCC.



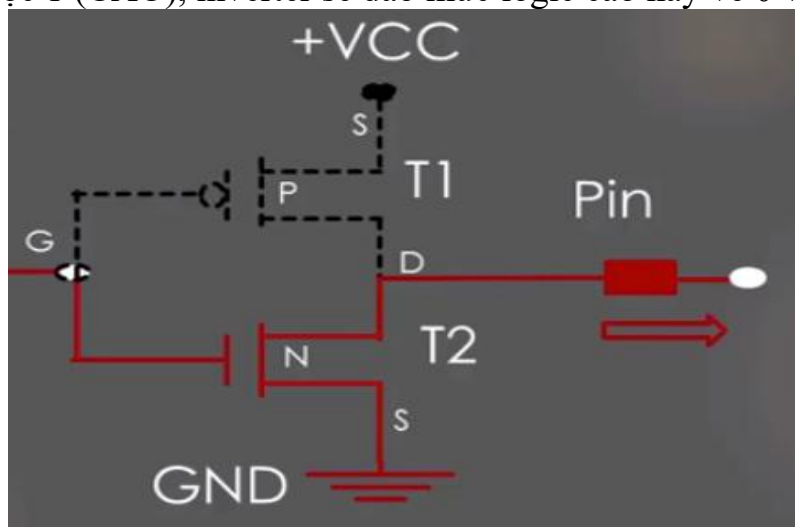
Hình 2.7 GPIO output LOW

Khi chúng ta ghi 0 từ phần mềm, biến tần đầu tiên sẽ đặt nó là 1 và NMOS sẽ được kích hoạt do đó chân sẽ được kéo xuống 0.

Đây là cấu hình mặc định của chân GPIO và còn được gọi là đầu ra push-pull.

Nếu chúng ta đặt dòng enable là 1, thì chân sẽ được cấu hình ở chế độ đầu vào bằng cách kích hoạt bộ đệm đầu vào. Bộ đệm đầu vào bao gồm hai transistor CMOS được kết nối theo kiểu dưới đây

Khi chân đọc 1 (CAO), inverter sẽ đảo mức logic cao này về 0 và transistor



Hình 2.8 GPIO Mode open drain

T1 PMOS sẽ BẬT. Vì nó được kéo lên VCC, phần mềm đọc logic là 1. Khi chân đọc 0 (LOW), inverter đảo mức logic thấp này thành 1 và T2 NMOS sẽ được kích hoạt và nó sẽ được kéo xuống 0, do đó phần mềm đọc 0. Dòng kích hoạt này được cấu hình bởi phần mềm.

Trong chế độ open-drain đầu ra, transistor PMOS trên cùng không có mặt. Sơ đồ trông giống như sau:

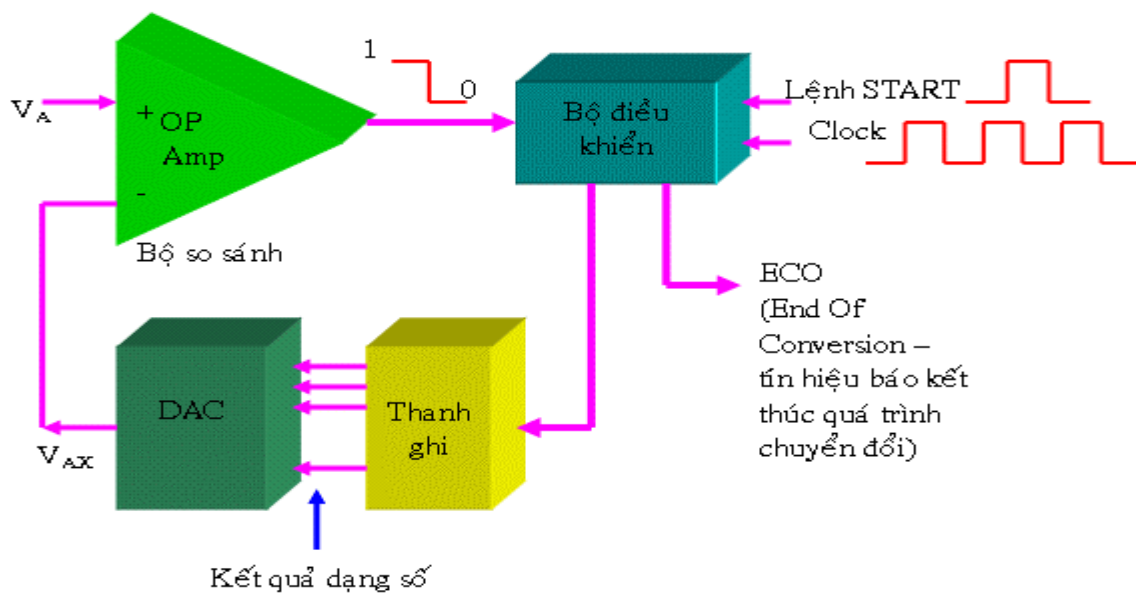
Khi T2 bật, chân sẽ được kéo xuống mức thấp (gnd). Khi T2 tắt, drain (D) của transistor sẽ nổi hoặc mở, do đó đầu ra sẽ nổi. Do đó, cấu hình open-drain chỉ có thể kéo chân xuống, không thể kéo chân lên. Vì open-drain chỉ có hai mức logic, logic 0 hoặc nổi, bạn cần cung cấp khả năng kéo lên cho chân này bằng cách kích hoạt điện trở kéo lên bên trong (có thể được thực hiện thông qua thanh ghi GPIO) hoặc đưa điện trở kéo lên bên ngoài. Trong một số thiết bị ngoại vi, GPIO được định cấu hình là open-drain, ví dụ, thiết bị ngoại vi I2C định cấu hình các chân SDA và SCL trong cấu hình open-drain.

2.1.3. ADC:

Trong các hệ thống nhúng việc đo lường các đại lượng vật lý là các đại lượng tương tự rất quan trọng ví dụ như đo lường nhiệt độ, độ ẩm, ánh sáng hay đo lường các dòng điện, điện áp mà ta thường xuyên gặp phải các đại lượng mang giá trị liên tục

Vì điều khiển là thiết bị số(digital) chỉ làm việc với các giá trị 0 và 1 vì vậy chúng ta cần có các phương án để chuyển đổi các đại lượng vật lý này để truyền tín hiệu về cho vi điều khiển cho nên ở đây chúng ta có bộ chuyển đổi tương tự số ADC

Sơ đồ tổng quát của bộ chuyển đổi tương tự sang số ADC



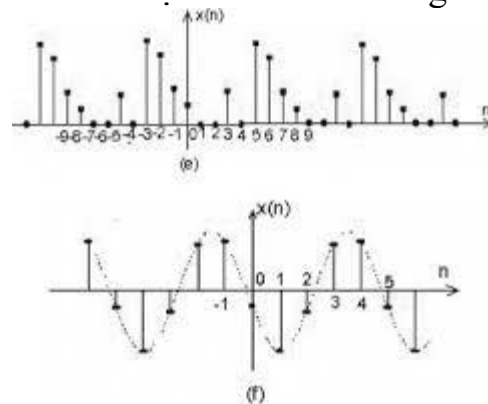
Hình 2.9 Sơ đồ hoạt động ADC

Đầu tiên sơ đồ tổng quát bao gồm các bộ so sánh sử dụng op-amp để so sánh điện áp tham chiếu từ các thiết bị ngoại vi điển hình là các cảm biến(sensor) sẽ đưa các điện áp giá trị tương tự so sánh với 1 điện áp mẫu nào đó sau đó sẽ tạo ra các xung nếu $V_A > V_{AX}$ thì đầu ra của op-amp sẽ là 1 nếu $V_A < V_{AX}$ thì đầu ra của op-amp sẽ là 0 sẽ được đưa vào bộ điều khiển với các lệnh start và xung Clock sau đó EOC(End of conversion) sẽ kết thúc quá trình chuyển đổi rồi sẽ đưa tín hiệu về các thanh ghi từ đó có thể đọc được giá trị từ cảm biến đưa vào

Các bước chuyển đổi ADC

1. Lấy mẫu

Là quá trình rời rạc hóa tín hiệu theo miền thời gian



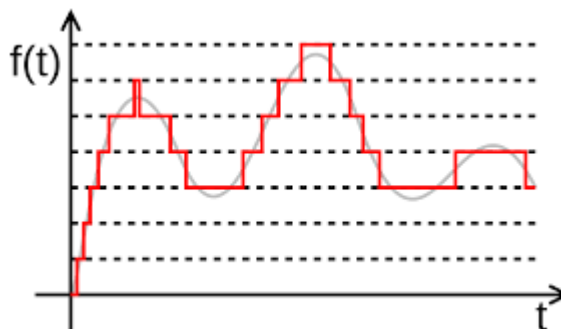
Hình 2.10 Quá trình lấy mẫu ADC

2. Lượng tử hóa

Là quá trình rời rạc hóa tín hiệu theo miền giá trị

Một giá trị bất kỳ của tín hiệu số đều phải biểu thị bằng bội số nguyên lần giá trị đơn vị nào đó, giá trị này nhỏ nhất được chọn

Đơn vị được chọn theo quy định này gọi là đơn vị lượng tử (bước lượng tử)



Hình 2.11 Quá trình lượng tử hóa ADC

3. Mã hóa

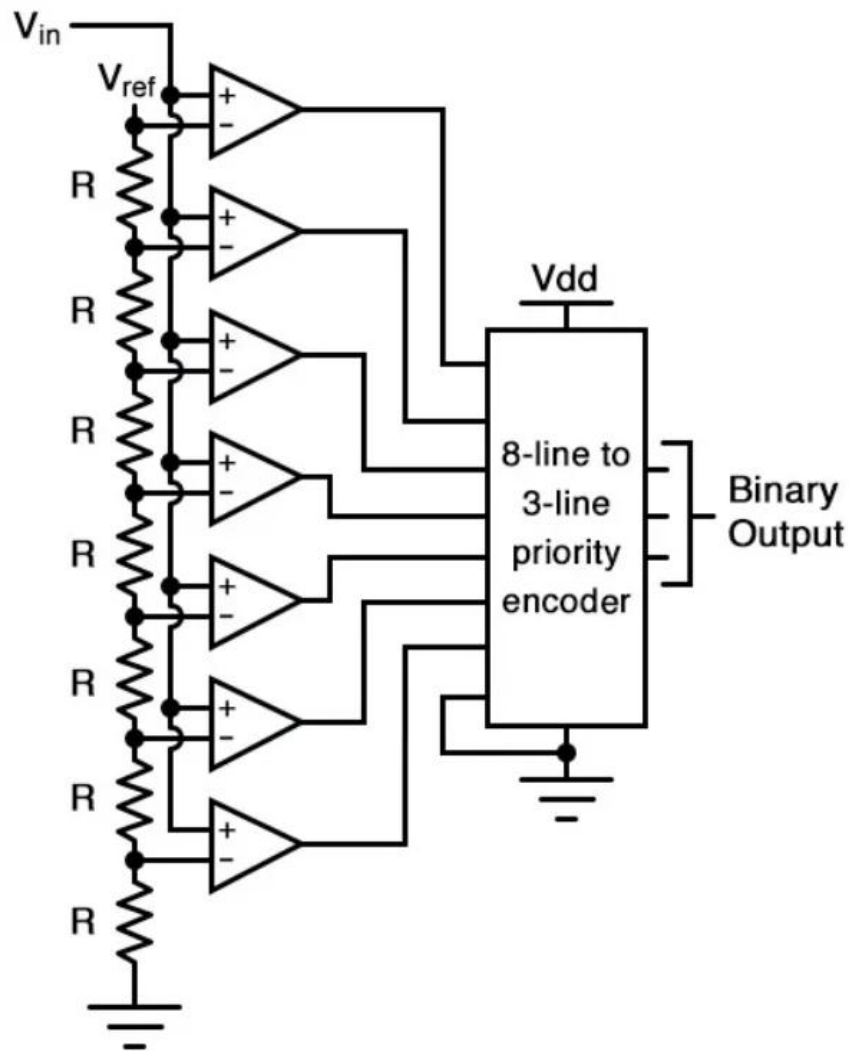
Là quá trình dùng mã nhị phân biểu thị giá trị tín hiệu số sau khi đã được lượng tử hóa

Mã nhị phân có được sau quá trình mã hóa chính là tín hiệu đầu ra của chuyển đổi ADC

Hiện nay có rất nhiều loại ADC

1. Flash ADC(ADC nhanh)

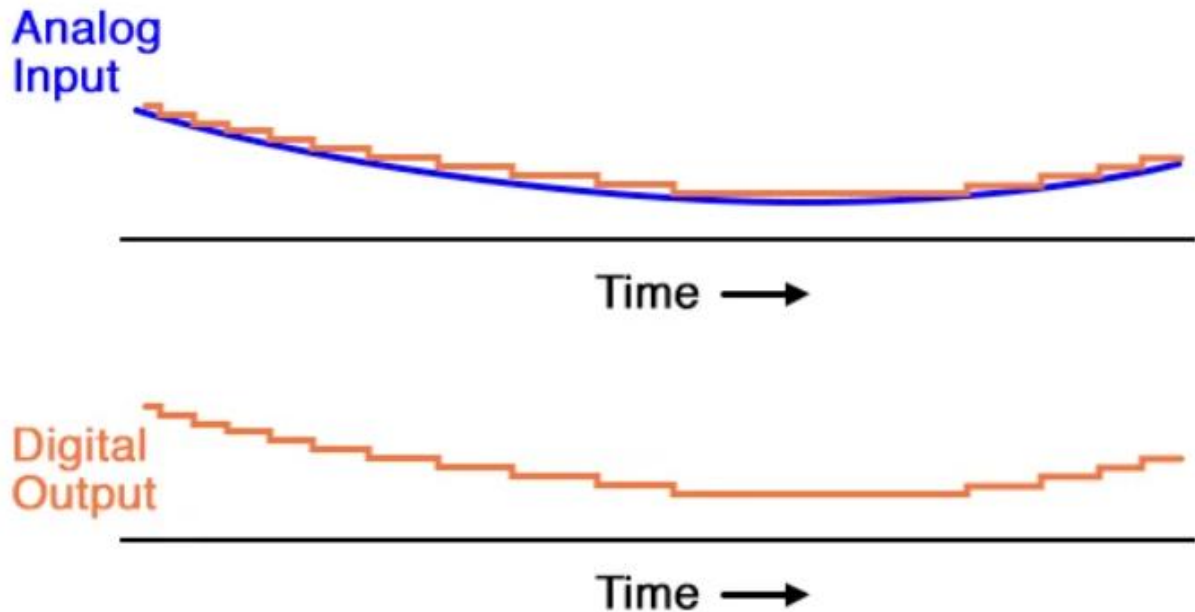
Đặc điểm là có cấu trúc song song các bộ chuyển đổi song song nên sẽ nhanh hơn các loại khác đầu vào sẽ bao gồm các op-amp có cấu trúc nối tầng với nhau



Hình 2.12 Flash ADC

V_{ref} là điện áp tham chiếu được thiết kế từ 1 bộ ổn áp có đầu ra ổn định khi đầu vào tương tự vượt quá điện áp V_{ref} thì lúc này mạch op amp hoạt động như 1 bộ so sánh đầu ra của bộ so sánh ở tích cực mức cao chạm ngưỡng bão hòa. Bộ mã hóa ưu tiên tạo ra một số nhị phân dựa trên đầu vào hoạt động bậc cao nhất, bỏ qua tất cả các đầu vào hoạt động khác.

Khi hoạt động, ADC flash sẽ tạo ra kết quả đầu ra trông giống như thế này:



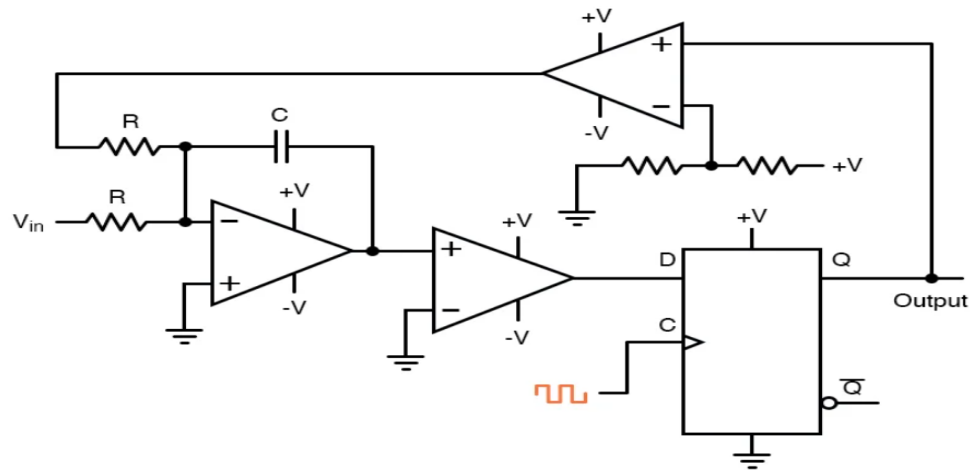
Tín hiệu số (Digital Output) bám sát tín hiệu analog nhưng có dạng bậc (vì số bit hữu hạn).

Cứ mỗi khoảng thời gian rất nhỏ (điểm lấy mẫu), ADC sẽ cập nhật một mức digital gần nhất

Ứng dụng: Radar, truyền hình số, oscilloscope tốc độ cao

2. Delta-Sigma ($\Delta\Sigma$) ADC

Sơ đồ mạch



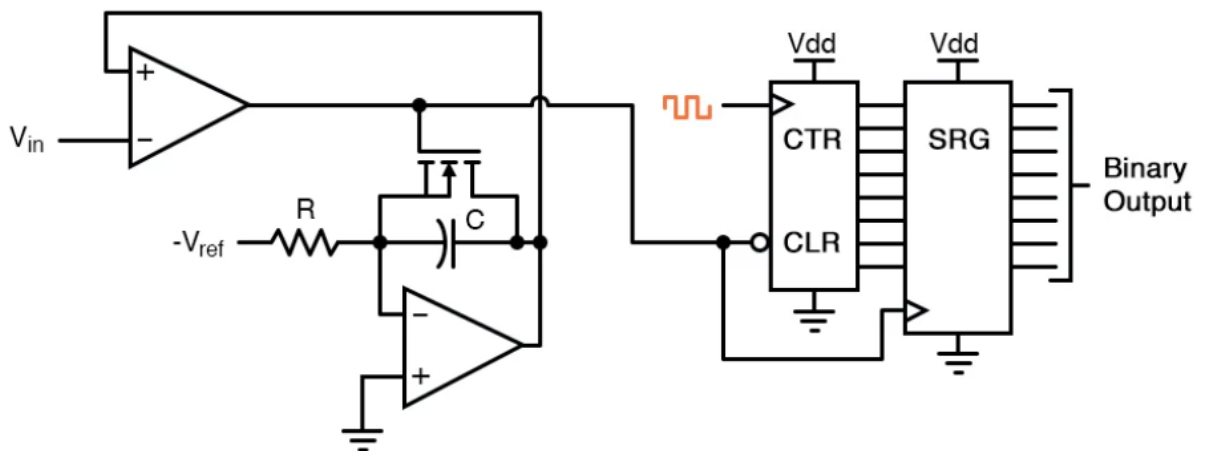
Hình 2.13 Sơ đồ mạch Fash ADC

Dùng trong các vi điều khiển nhúng chính xác cao (ví dụ MSP430, STM32L4)

Ưu điểm: độ phân giải rất cao (16–24 bit), lọc nhiễu tốt

Nhược điểm: tốc độ chậm hơn SAR

3. Dual-Slope ADC / Integrating ADC



Hình 2.14 Sơ đồ mạch Dual-Slope ADC

Dùng trong các vi điều khiển nhúng thiết bị đo (multimeter, đồng hồ số)

Ưu điểm: chính xác, chống nhiễu tốt

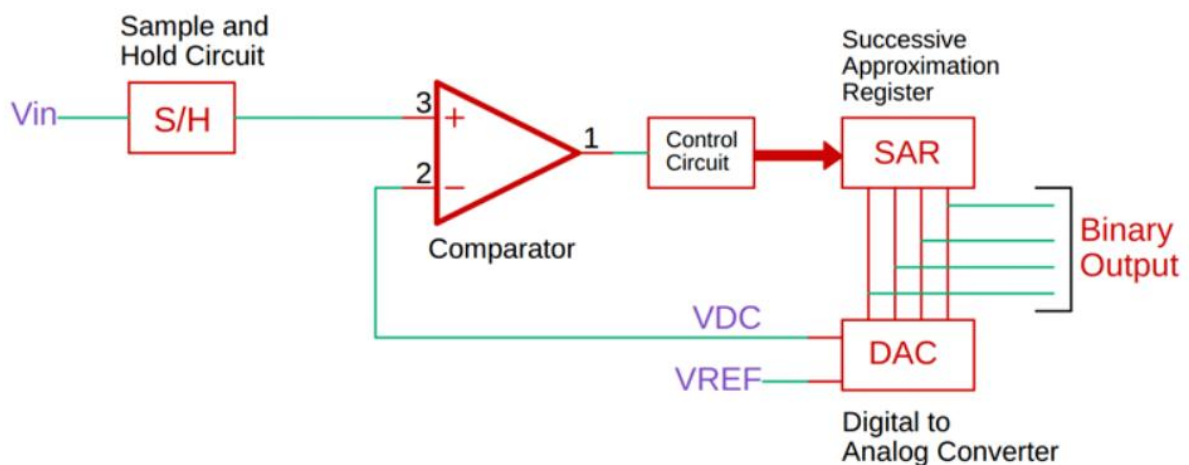
Nhược điểm: tốc độ rất chậm

4. SAR (Successive Approximation Register)

Trong Esp32 chúng ta sẽ dùng loại ADC này

ADC xấp xỉ liên tiếp là ADC được lựa chọn cho các ứng dụng có độ phân giải trung bình đến cao giá thành thấp, độ phân giải cho ADC SAR dao động từ 8 - 18 bit, với tốc độ lấy mẫu lên đến 5 mega-mẫu mỗi giây (MSPS). Ngoài ra, nó có thể được chế tạo theo dạng hệ số nhỏ với mức tiêu thụ điện năng thấp, đó là lý do tại sao loại ADC này được sử dụng cho các thiết bị chạy bằng pin di động.

Như tên gọi của nó, ADC này áp dụng thuật toán tìm kiếm nhị phân để chuyển đổi các giá trị, đó là lý do tại sao mạch bên trong có thể chạy ở vài MHz nhưng tốc độ lấy mẫu thực tế lại thấp hơn nhiều do thuật toán xấp xỉ liên tiếp.

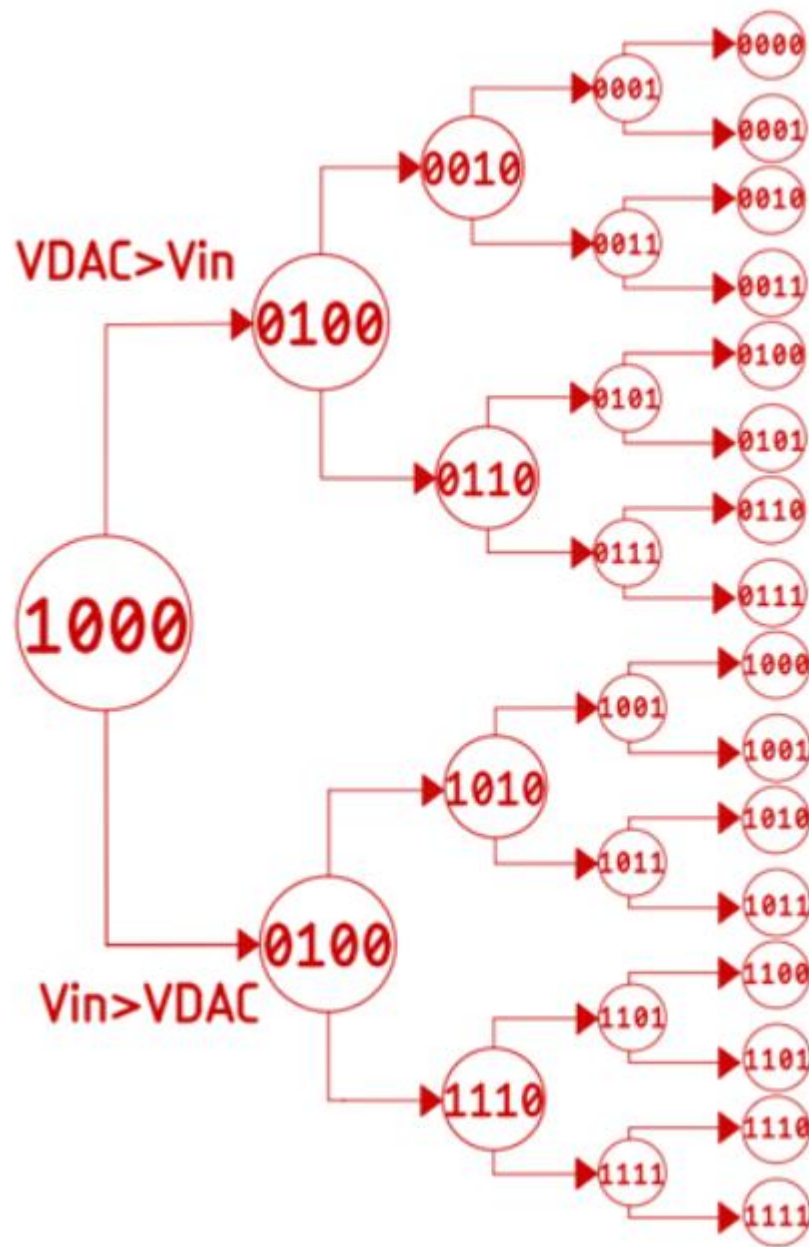


Hình 2.15 Sơ đồ nguyên lý SAR

Nếu V_{in} lớn hơn đầu ra của DAC, bit quan trọng nhất sẽ giữ nguyên và bit tiếp theo sẽ được đặt cho một phép so sánh mới. Ngược lại, nếu điện áp đầu vào nhỏ hơn giá trị DAC, bit quan trọng nhất sẽ được đặt thành 0 và bit tiếp theo sẽ được đặt thành 1 cho một phép so sánh mới. Bây giờ nếu bạn thấy hình ảnh bên dưới, điện áp DAC là 5V và vì nó nhỏ hơn điện áp đầu vào, bit tiếp theo trước bit quan trọng nhất sẽ được đặt thành 1 và các bit khác sẽ được đặt thành 0, quá trình này sẽ tiếp tục cho đến khi đạt đến giá trị gần nhất với điện áp đầu vào.

Đây là cách ADC xấp xỉ liên tiếp thay đổi 1 bit tại một thời điểm để xác định điện áp đầu vào và tạo ra giá trị đầu ra. Và bất kể giá trị nào có thể là

trong bốn lần lặp, chúng ta sẽ nhận được mã kỹ thuật số đầu ra từ giá trị đầu vào. Cuối cùng, danh sách tất cả các kết hợp có thể có cho ADC xấp xỉ liên tiếp bốn bit được hiển thị



Hình 2.16 Mã hóa trong ADC

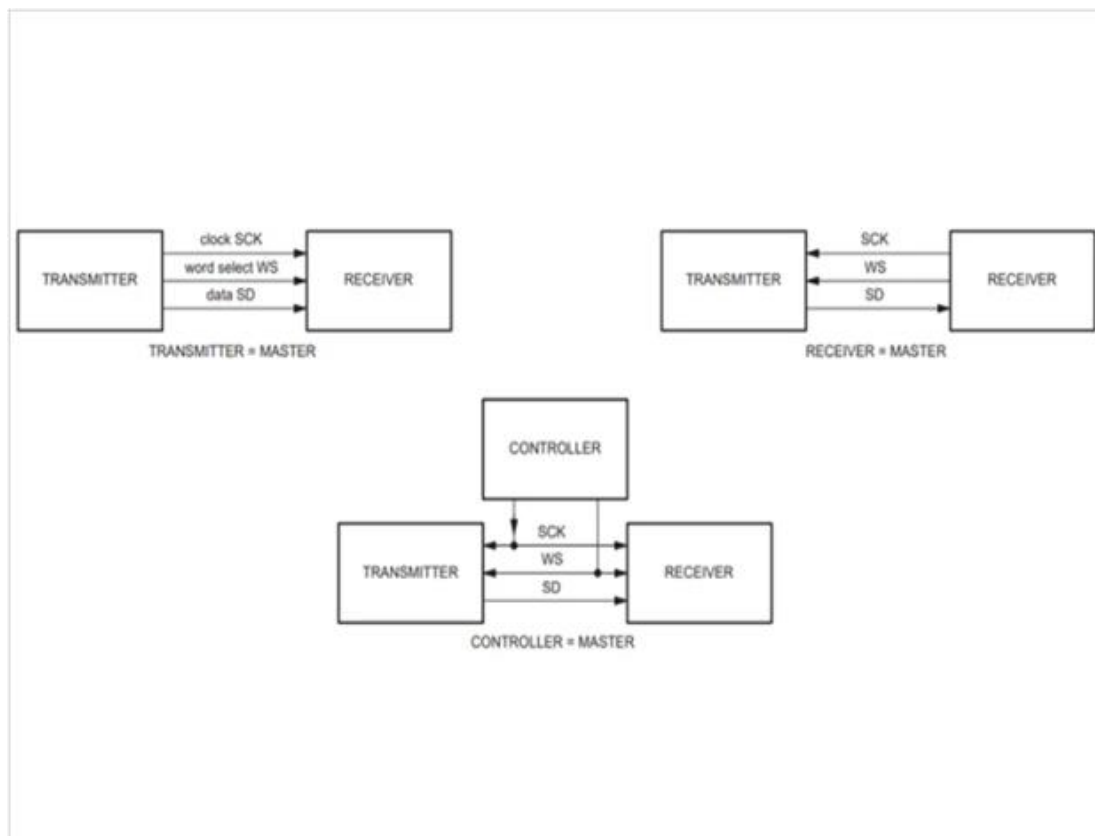
2.1.4. I2S:

Ta thấy sự tương đồng về 2 giao thức I2C và I2S đều bắt đầu bằng I2 tức là giao tiếp giữa các IC. Tuy nhiên I2S được phát hành sau I2C và I2S được thiết kế để truyền dữ liệu âm thanh

I2S được tạo ra vào những năm 1980, khi kỹ thuật số bắt đầu chinh phục thị trường âm thanh tiêu dùng. Mục đích được nêu của I2S là tạo điều kiện thuận lợi cho sự phát triển của thiết bị điện tử âm thanh thông qua giao diện chuẩn hóa để truyền dữ liệu kỹ thuật số giữa các ADC, DAC, bộ lọc kỹ thuật số, bộ xử lý tín hiệu kỹ thuật số và các loại IC khác được sử dụng trong hệ thống âm thanh. Về bản chất, đây là giao thức hai kênh, vì nó được thiết kế cho âm thanh nổi

1. Đặc điểm của I2S

Sơ đồ sau đây mô tả ba cấu hình được I2S hỗ trợ.



Dữ liệu được truyền trên đường SD, trạng thái của đường WS tương ứng với kênh âm thanh (phải hoặc trái) hiện đang được truyền và đường xung nhịp mang xung nhịp nối tiếp. Như bạn có thể thấy trong sơ đồ, tín hiệu WS và SCK có thể được tạo ra bởi bộ phát, bộ thu hoặc thành phần bộ điều khiển của bên thứ ba

2. Dữ liệu nối tiếp (SD)

Các giá trị số được truyền MSb trước.

Máy phát và máy thu không cần phải có độ dài từ đã thỏa thuận; máy phát sẽ gửi những gì nó có và máy thu sẽ lấy những gì nó có thể sử dụng.

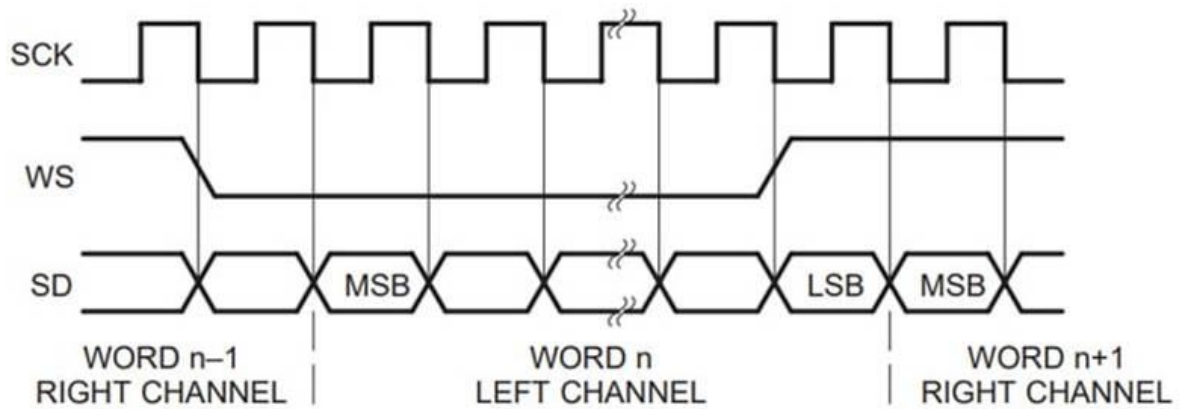
Các bit dữ liệu mới có thể được đồng hồ ra ở cạnh tăng hoặc giảm của đồng hồ. Tuy nhiên, chúng phải được đồng hồ vào ở cạnh tăng, vì vậy cách tiếp cận đơn giản hơn ở đây là cách sắp xếp được hiển thị trong sơ đồ bên dưới—tức là chúng ta đồng hồ ra ở cạnh giảm và chúng ta đồng hồ vào ở cạnh tăng.

Giao thức không bao gồm các khoảng thời gian xung nhịp chưa sử dụng giữa các từ; LSB của một từ sẽ được theo ngay sau MSb của từ tiếp theo.

3. Chọn từ (WS)

Mức logic thấp trên WS cho biết từ hiện đang được truyền là một phần của luồng dữ liệu cho kênh âm thanh bên trái; mức logic cao trên WS cho biết âm thanh kênh bên phải.

Để tạo điều kiện xử lý dữ liệu ở cả phía máy phát và máy thu, tín hiệu WS chuyển tiếp 1 chu kỳ xung nhịp trước khi hoàn tất 1 từ dữ liệu

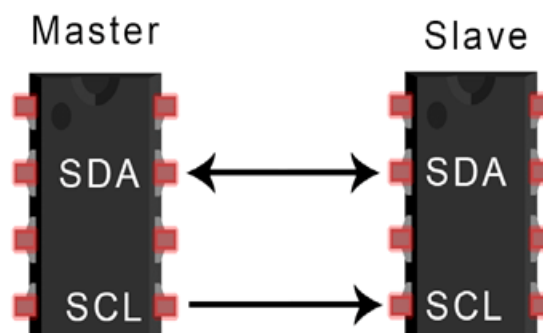


4. Clock

Giao thức không chỉ định tốc độ dữ liệu tối đa
Đồng hồ chạy liên tục

2.1.5. I2C:

I2C kết hợp các tính năng tốt nhất của SPI và UART. Với I2C, bạn có thể kết nối nhiều slave với một master duy nhất (như SPI) và bạn có thể có nhiều master điều khiển một hoặc nhiều slave. Điều này thực sự hữu ích khi bạn muốn có nhiều hơn một vi điều khiển ghi dữ liệu vào một thẻ nhớ duy nhất hoặc hiển thị văn bản trên một màn hình LCD.



Giống như giao tiếp UART, I2C chỉ sử dụng hai dây để truyền dữ liệu giữa các thiết bị:

Nhóm 16 - 154166

SDA (Serial Data) - đường truyền cho master và slave để gửi và nhận dữ liệu.

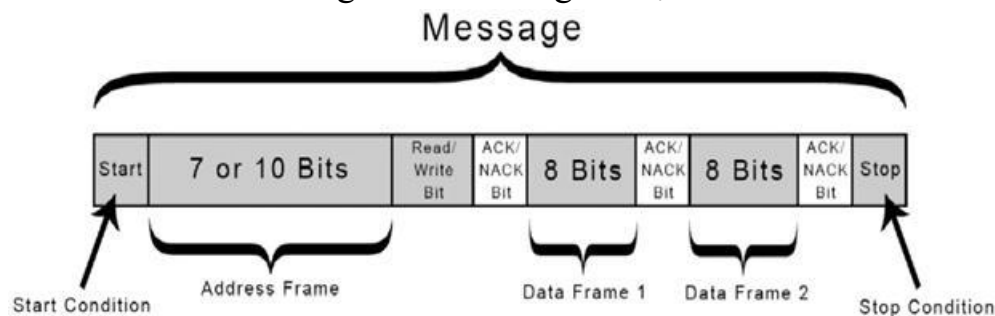
SCL (Serial Clock) - đường mang tín hiệu xung nhịp.

I2C là một giao thức truyền thông nối tiếp, vì vậy dữ liệu được truyền từng bit dọc theo một đường duy nhất (đường SDA).

Giống như SPI, I2C là đồng bộ, do đó đầu ra của các bit được đồng bộ hóa với việc lấy mẫu các bit bởi một tín hiệu xung nhịp được chia sẻ giữa master và slave. Tín hiệu xung nhịp luôn được điều khiển bởi master.

Cách hoạt động của I2C

Với I2C, dữ liệu được truyền trong các tin nhắn. Tin nhắn được chia thành các khung dữ liệu. Mỗi tin nhắn có một khung địa chỉ chứa địa chỉ nhị phân của địa chỉ slave và một hoặc nhiều khung dữ liệu chứa dữ liệu đang được truyền. Thông điệp cũng bao gồm điều kiện khởi động và điều kiện dừng, các bit đọc / ghi và các bit ACK / NACK giữa mỗi khung dữ liệu:



- Điều kiện khởi động: Đường SDA chuyển từ mức điện áp cao xuống mức điện áp thấp trước khi đường SCL chuyển từ mức cao xuống mức thấp.
- Điều kiện dừng: Đường SDA chuyển từ mức điện áp thấp sang mức điện áp cao sau khi đường SCL chuyển từ mức thấp lên mức cao.
- Khung địa chỉ: Một chuỗi 7 hoặc 10 bit duy nhất cho mỗi slave để xác định slave khi master muốn giao tiếp với nó.
- Bit Đọc / Ghi: Một bit duy nhất chỉ định master đang gửi dữ liệu đến slave (mức điện áp thấp) hay yêu cầu dữ liệu từ nó (mức điện áp cao).
- Bit ACK / NACK: Mỗi khung trong một tin nhắn được theo sau bởi một bit xác nhận / không xác nhận. Nếu một khung địa chỉ hoặc khung dữ liệu được nhận thành công, một bit ACK sẽ được trả lại cho thiết bị gửi từ thiết bị nhận.

Địa chỉ

I2C không có các đường Slave Select như SPI, vì vậy cần một cách khác để cho slave biết rằng dữ liệu đang được gửi đến slave này chứ không

phải slave khác. Nó thực hiện điều này bằng cách định địa chỉ. Khung địa chỉ luôn là khung đầu tiên sau bit khởi động trong một tin nhắn mới.

Master gửi địa chỉ của slave mà nó muốn giao tiếp với mọi slave được kết nối với nó. Sau đó, mỗi slave sẽ so sánh địa chỉ được gửi từ master với địa chỉ của chính nó. Nếu địa chỉ phù hợp, nó sẽ gửi lại một bit ACK điện áp thấp cho master. Nếu địa chỉ không khớp, slave không làm gì cả và đường SDA vẫn ở mức cao.

Bit đọc / ghi:

Khung địa chỉ bao gồm một bit duy nhất ở cuối tin nhắn cho slave biết master muốn ghi dữ liệu vào nó hay nhận dữ liệu từ nó. Nếu master muốn gửi dữ liệu đến slave, bit đọc / ghi ở mức điện áp thấp. Nếu master đang yêu cầu dữ liệu từ slave, thì bit ở mức điện áp cao.

Khung dữ liệu:

Sau khi master phát hiện bit ACK từ slave, khung dữ liệu đầu tiên đã sẵn sàng được gửi.

Khung dữ liệu luôn có độ dài 8 bit và được gửi với bit quan trọng nhất trước. Mỗi khung dữ liệu ngay sau đó là một bit ACK / NACK để xác minh rằng khung đã được nhận thành công. Bit ACK phải được nhận bởi master hoặc slave (tùy thuộc vào cái nào đang gửi dữ liệu) trước khi khung dữ liệu tiếp theo có thể được gửi.

Sau khi tất cả các khung dữ liệu đã được gửi, master có thể gửi một điều kiện dừng cho slave để tạm dừng quá trình truyền. Điều kiện dừng là sự chuyển đổi điện áp từ thấp lên cao trên đường SDA sau khi chuyển tiếp từ thấp lên cao trên đường SCL, với đường SCL vẫn ở mức cao.

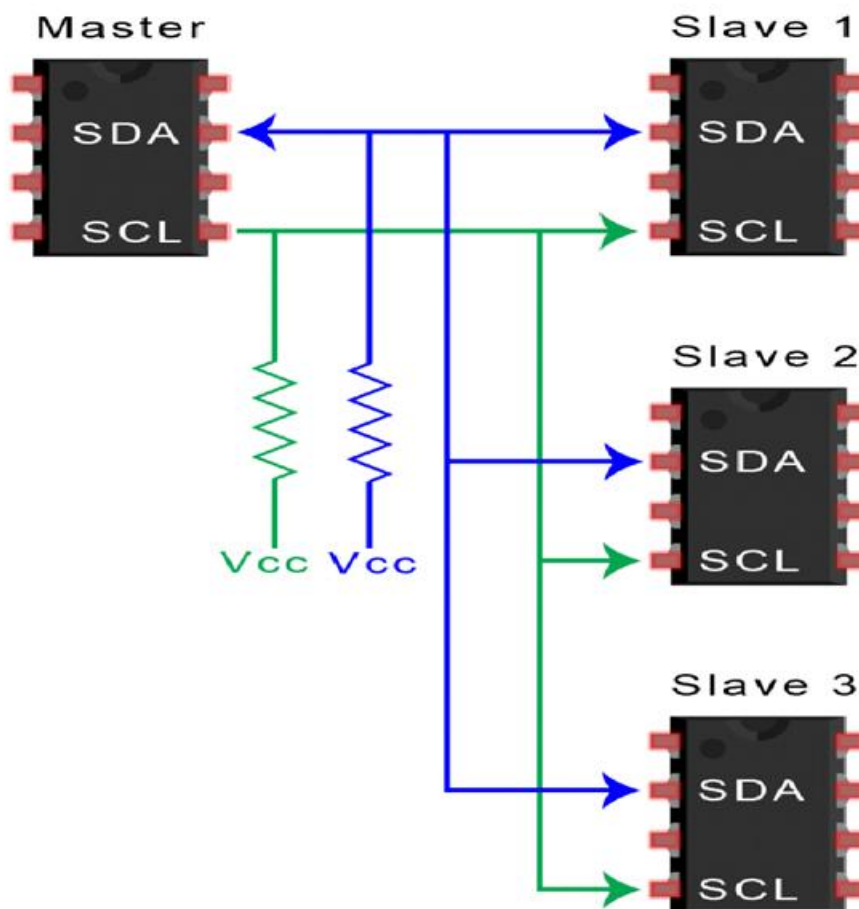
Các bước truyền dữ liệu I2C:

1. Master gửi điều kiện khởi động đến mọi slave được kết nối bằng cách chuyển đường SDA từ mức điện áp cao sang mức điện áp thấp trước khi chuyển đường SCL từ mức cao xuống mức thấp.
2. Master gửi cho mỗi slave địa chỉ 7 hoặc 10 bit của slave mà nó muốn giao tiếp, cùng với bit đọc / ghi.
3. Mỗi slave sẽ so sánh địa chỉ được gửi từ master với địa chỉ của chính nó. Nếu địa chỉ trùng khớp, slave sẽ trả về một bit ACK bằng cách kéo dòng SDA xuống thấp cho một bit. Nếu địa chỉ từ master không khớp với địa chỉ của slave, slave rời khỏi đường SDA cao.
4. Master gửi hoặc nhận khung dữ liệu.

5. Sau khi mỗi khung dữ liệu được chuyển, thiết bị nhận trả về một bit ACK khác cho thiết bị gửi để xác nhận đã nhận thành công khung.
6. Để dừng truyền dữ liệu, master gửi điều kiện dừng đến slave bằng cách chuyển đổi mức cao SCL trước khi chuyển mức cao SDA.

Một master với nhiều slave

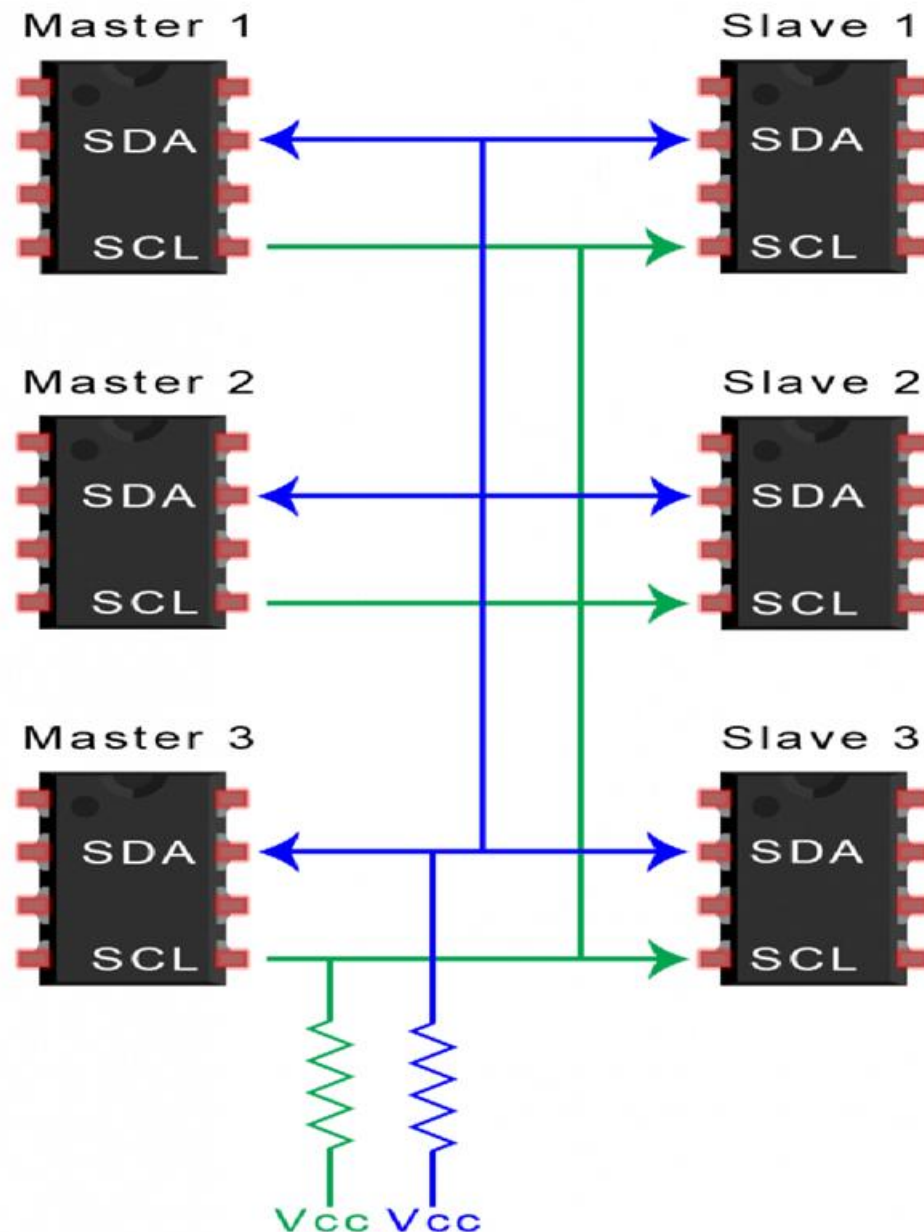
Vì I2C sử dụng định địa chỉ nên nhiều slave có thể được điều khiển từ một master duy nhất. Với địa chỉ 7 bit sẽ có 128 (2^7) địa chỉ duy nhất. Việc sử dụng địa chỉ 10 bit không phổ biến, nhưng nó cung cấp 1.024 (2^{10}) địa chỉ duy nhất. Để kết nối nhiều slave đến một master duy nhất, bạn có thể đấu dây như thế này, với điện trở kéo lên 4,7K Ohm kết nối đường SDA và SCL với Vcc:



Nhiều master với nhiều slave

Nhiều master có thể được kết nối với một slave hoặc nhiều slave. Sự cố với nhiều master trong cùng một hệ thống xảy ra khi hai master cố gắng gửi hoặc nhận dữ liệu cùng một lúc qua đường SDA. Để giải quyết vấn đề này, mỗi master cần phải phát hiện xem đường SDA thấp hay cao trước khi truyền tin

nhấn. Nếu đường SDA thấp, điều này có nghĩa là một master khác có quyền điều khiển bus và master đó phải đợi để gửi tin nhắn. Nếu đường SDA cao thì có thể truyền tin nhắn an toàn. Để kết nối nhiều master với nhiều slave, hãy sử dụng sơ đồ sau, với các điện trở kéo lên 4,7K Ohm kết nối các đường SDA và SCL với Vcc:



Ưu điểm và nhược điểm của I2C

Có rất nhiều điều ở I2C có thể khiến nó nghe có vẻ phức tạp so với các giao thức khác, nhưng có một số lý do chính đáng khiến bạn có thể muốn hoặc không muốn sử dụng I2C để kết nối với một thiết bị cụ thể:

Ưu điểm

- Chỉ sử dụng hai dây
- Hỗ trợ nhiều master và nhiều slave
- Bit ACK / NACK xác nhận mỗi khung được chuyển thành công
- Phần cứng ít phức tạp hơn so với UART
- Giao thức nổi tiếng và được sử dụng rộng rãi

Nhược điểm

- Tốc độ truyền dữ liệu chậm hơn SPI
- Kích thước của khung dữ liệu bị giới hạn ở 8 bit
- Cần phần cứng phức tạp hơn để triển khai so với SPI

2.1.6. PWM:

PWM là phương pháp điều chế độ rộng xung-tức là tạo ra 1 tín hiệu hình chữ nhật(On/Off) có: tần số cố định, độ rộng xung(duty cycle) thay đổi theo giá trị điều khiển.

Ứng dụng của PWM: điều khiển độ sáng LED, điều tốc động cơ DC, servo, tạo âm thanh...

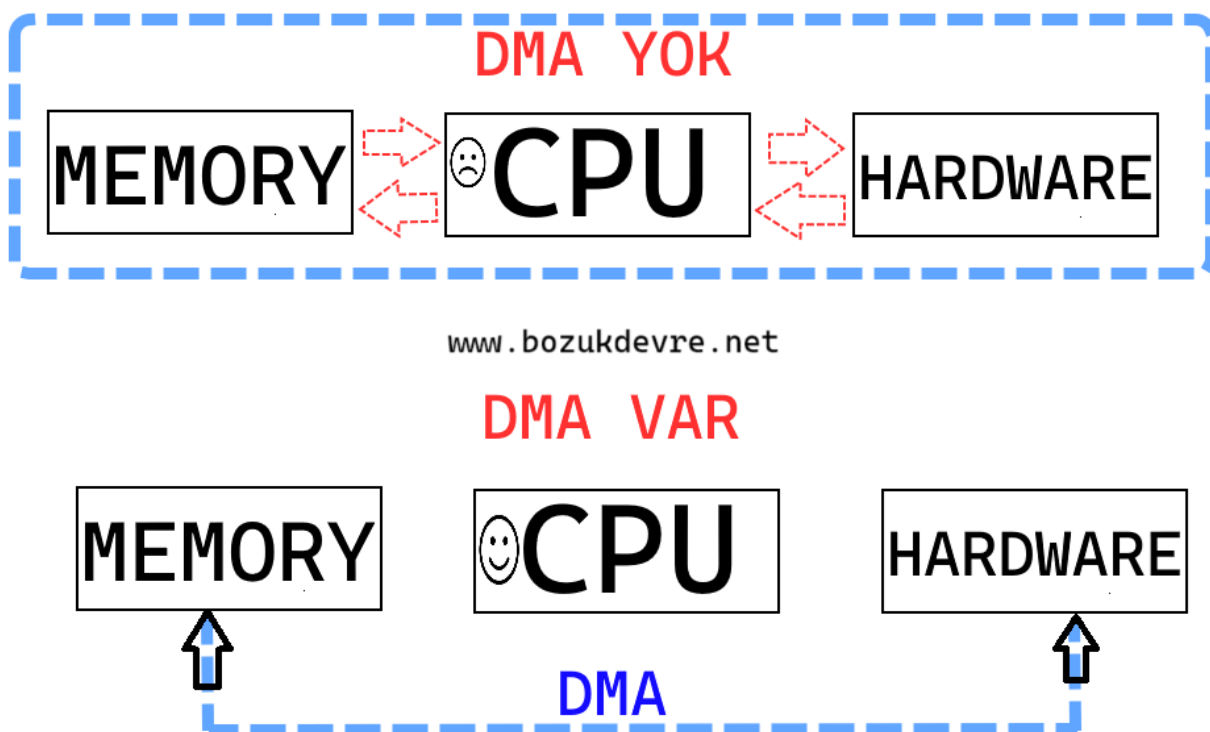
PWM trong ESP32 - sử dụng LEDC (LED Controller): ESP32 sử dụng phần cứng LEDC để phát PWM: hỗ trợ 16 kênh PWM độc lập, mỗi kênh có thể gán cho bất kỳ chân GPIO nào, hỗ trợ độ phân giải lên tới 20 - bit, tần số PWM từ vài Hz đến hàng trăm kHz.

Cấu trúc điều khiển PWM: Timer(4 timer 0-3) quyết định tần số và độ phân giải, Channel (16 channel) điều khiển duty cycle và gán GPIO. Mỗi channel liên kết với 1 timer để tạo tín hiệu PWM.

2.1.7. DMA:

Thông thường, CPU sẽ điều khiển việc transfer data giữa Peripheral (UART, I2C, SPI, ...) và bộ nhớ (RAM) qua các đường bus. Cơ chế này được hiểu như cơ chế Master - Slave, với CPU đóng vai trò là Master, Peripheral và Memory đóng vai trò như các Slave. Nên việc giao tiếp giữa 2 Slave sẽ do Master điều khiển.

Tuy nhiên khi sử dụng nhiều ngoại vi hay nhiều dòng lệnh làm tiêu tốn tài nguyên CPU, khi đó hệ thống sẽ không thể hoạt động chính xác được nữa do tiêu tốn thời gian thực hiện việc truyền dữ liệu từ ngoại vi – RAM (Trong thời gian đó CPU sẽ không thực hiện các thao tác khác bên ngoài).

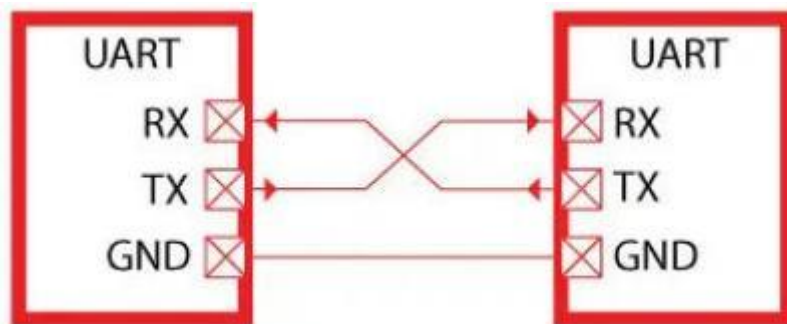


Hình 2.17 Nguyên lý hoạt động DMA

DMA - Direct Memory Access, hay truy cập trực tiếp bộ nhớ, được biết đến như một thành phần trong vi điều khiển, có vai trò như một Master, dùng để điều khiển việc truy cập trực tiếp vào bộ nhớ mà không thông qua CPU. DMA có thể làm giảm tải "áp lực làm việc" cho CPU.

DMA là một phần cứng chuyên biệt, có tác dụng làm thay cho CPU các thao tác truyền nhận dữ liệu, không có các chức năng tính toán hay đưa ra các lệnh điều khiển.

a) UART:

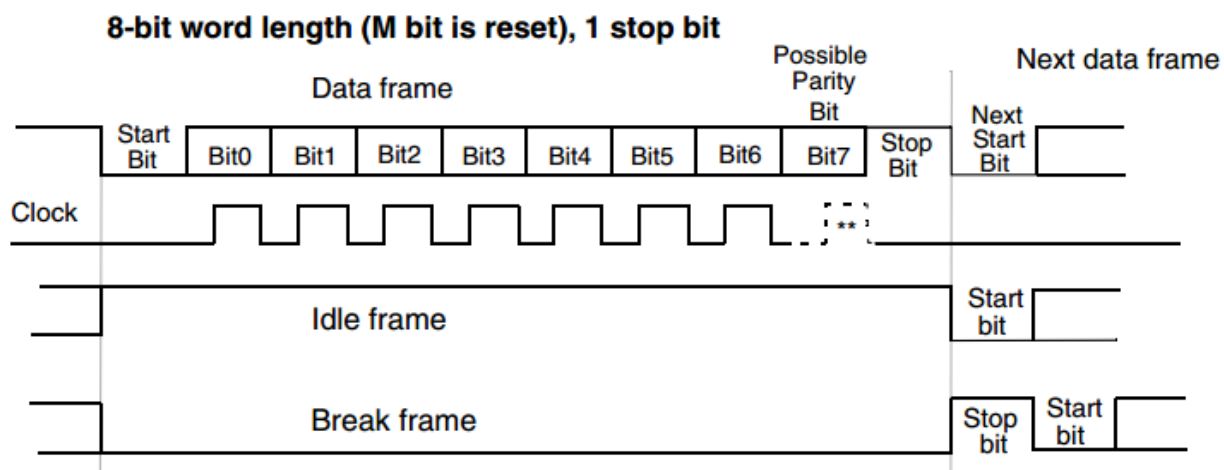


Hình 2.18 UART hoạt động

UART (Bộ thu phát không đồng bộ phổ dụng) là một chuẩn giao tiếp nối tiếp (serial communication protocol) được sử dụng rộng rãi trong lĩnh vực hệ thống nhúng và vi điều khiển. UART cho phép truyền và nhận dữ liệu theo từng bit nối tiếp, thường được dùng để giao tiếp giữa vi điều khiển và

các thiết bị ngoại vi như: máy tính, module Bluetooth, module GPS, cảm biến,...

Không giống như các chuẩn đồng bộ (như SPI hoặc I2C), UART hoạt động không đồng bộ, tức là không sử dụng tín hiệu xung nhịp (clock) chung giữa hai thiết bị. Thay vào đó, UART sử dụng một số cấu hình thống nhất giữa 2 bên, bao gồm: tốc độ baud (baud rate), số bit dữ liệu, bit chẵn lẻ (parity), và số bit stop để đảm bảo việc truyền dữ liệu chính xác.



Hình 2.19 Khung truyền cơ bản của một UART

Các thông số quan trọng:

- **Baudrate:** Số bit truyền được trong 1s, ở truyền nhận không đồng bộ thì ở các bên truyền và nhận phải thống nhất Baudrate. Các thông số tốc độ Baudrate thường hay sử dụng để giao tiếp với máy tính là 600,1200,2400,4800,9600,14400,19200,38400,56000,57600,115200.
- **Khung truyền (Frame):** Ngoài việc giống nhau của tốc độ baud 2 thiết bị truyền nhận thì khung truyền của bên cũng được cấu hình giống nhau. Khung truyền quy định số bit trong mỗi lần truyền, bit bắt đầu “Start bit”, các bit kết thúc (Stop bit), bit kiểm tra tính chẵn lẻ (Parity), ngoài ra số bit quy định trong một gói dữ liệu cũng được quy định bởi khung truyền. Có thể thấy, khung truyền đóng một vai trò rất quan trọng trong việc truyền thành công dữ liệu.
- **Start bit:** Bit đầu tiên được truyền trong một frame, chức năng báo hiệu bắt đầu truyền một bản tin.
- **Data:** Data hay dữ liệu là thông tin mà chúng ta nhận được trong quá trình truyền và nhận.
- **Parity bit:** Parity dùng để kiểm tra dữ liệu truyền có đúng hay không. Có 2 loại Parity đó là Parity chẵn (even parity) và parity lẻ (odd parity).

parity). Parity bit cần được thống nhất ở cả phần phát và thu, đôi khi ta không cần thiết lập parity bit.

- **Stop bits:** Stop bits là một bit báo cáo để cho bộ truyền/nhận biết được gói dữ liệu đã được gửi xong.

Tín hiệu clock trong hình trên không phải là xung clock vì uart hoạt động không cần đến clock của vi điều khiển, đó là tín hiệu clock được chia từ bộ clock của vi điều khiển để đạt được giá trị đúng với giá trị baudrate.

2.1.8. Mã hóa Reed-Solomon

Reed–Solomon (RS) là mã sửa lỗi đối tượng (symbol-based), được phát triển từ lý thuyết BCH, hoạt động trên các “symbol” (ví dụ 8-bit) thành mã dạng khối (block code) hệ tuyến tính. Mã được ký hiệu $RS(n, k)$, với n là độ dài mã sau khi thêm ECC, và k là số symbol dữ liệu gốc. Mỗi symbol thường có độ dài s bit, và số symbol $ECC = n - k = 2t$ cho phép mã khôi phục tối đa t symbol bị lỗi ($2t = \text{số ECC}$).

Tính năng quan trọng: RS là **MDS code** (Maximum-Distance-Separable), nghĩa là độ lệch tối thiểu (minimum distance) đạt $n - k + 1$, đảm bảo tránh lỗi ký tự tối ưu theo ràng buộc Singleton.

Mã hóa: biểu diễn message là đa thức $p(x)$ tại các phần tử khác nhau của trường hữu hạn $GF(q)$. Mỗi evaluation tạo symbol mã. Phương pháp BCH-systematic dùng đa thức phát sinh $g(x)$ để gắn ECC phía sau dữ liệu gốc.

Khả năng sửa lỗi & Cơ chế giải mã

- Với $RS(n, k)$, có thể sửa tối đa $t = (n - k)/2$ symbol lỗi bất kỳ. Số ECC càng nhiều thì độ chịu lỗi càng cao.
- Các thuật toán giải mã phổ biến: Berlekamp–Massey và Peterson–Gorenstein–Zierler giúp xác định đa thức lỗi và giá trị lỗi để khôi phục dữ liệu gốc. RS đặc biệt mạnh trước lỗi theo burst (nhiều bit lỗi trong một symbol), vì lỗi trong symbol tính chung là một symbol lỗi duy nhất.
- Ứng dụng: đĩa CD/DVD, QR code, GPS, vệ tinh, RAID, Bluetooth, VHS, truyền dữ liệu vệ tinh như Voyager, Cassini, Mars Rover...

2.1.9. DSS (Direct Sequence Spread)

DSSS (Direct Sequence Spread Spectrum) là kỹ thuật trải phổ trực tiếp, trong đó tín hiệu số gốc được nhân với một chuỗi mã giả ngẫu nhiên (PN-sequence) có tốc độ chip cao hơn nhiều tốc độ bit dữ liệu. Kết quả là tín hiệu được trải rộng ra dải tần lớn hơn nhiều so với băng gốc, giúp tăng khả năng chống nhiễu và bảo mật.

Cụ thể, nếu tín hiệu gốc có tốc độ bit R_b , chuỗi PN có tốc độ chip R_c với $R_c \gg R_b$, thì tại mỗi bit, tín hiệu được nhân với nhiều chip (thường gọi là chipping). Tín hiệu sau khi nhân xuất ra dải băng rộng theo độ rộng chip. Phương pháp giải mã (despread) dựa trên việc nhân lại với PN code, giúp tín hiệu dữ liệu được phục hồi và nhiễu không liên quan bị giảm mạnh vì không tương quan với PN code đó.

Cơ chế tạo tín hiệu và giải phổ

Trải rộng tín hiệu (Spreading)

1. Người gửi có dữ liệu bit $d[n] \in \{\pm 1\}$.
2. Có chuỗi chip $c[n] \in \{\pm 1\}$, chạy ở tần số cao (R_c).
3. Tín hiệu trải rộng: $s[n] = d[n / (R_c / R_b)] \times c[n]$
4. Kết quả là tín hiệu có băng thông rộng, tương tự như “nhiều trắng” nếu nhìn tổng thể, giúp tránh bị gây nhiễu có chủ ý.

Phục hồi tại bên thu (Despreading)

Bên thu dùng **mã chip giống** để nhân lại tín hiệu: $\hat{d}[n] = s[n] \times c[n] = d[\dots] \times c[n]^2 = d[\dots]$

Sau đó tích hợp hoặc lọc trung bình để khôi phục bit dữ liệu gốc. Vì $c[n]^2 = 1$ nên tín hiệu gốc được thu lại đầy đủ. Với nhiễu không tương quan mã, ảnh hưởng sẽ giảm đi theo **process gain**.

Tính chất của chuỗi PN: Chuỗi **pseudorandom** (PN), ngắn hạn giống ngẫu nhiên nhưng thực ra định sẵn (deterministic).

Các tính chất quan trọng:

- **Tự tương quan cao** (auto-correlation): giúp đồng bộ tín hiệu, hỗ trợ giải mã.
- **Tương quan chéo thấp** (cross-correlation): nhiều thiết bị có thể dùng chung băng tần với các mã khác nhau mà không gây nhiễu (CDMA).

Process Gain (Lợi ích trải phổ) biểu diễn khả năng chống nhiễu:

$$G_p = \frac{W_{\text{spread}}}{W_{\text{data}}} = \frac{R_c}{R_b}$$

Giải thích:

- Nếu băng tín hiệu gốc 1 kHz và trải rộng đến 100 kHz, $G_p = 100 \rightarrow$ khoảng 20 dB lợi ích SNR.
- Lợi ích đạt được nhờ giảm “độ ồn nhiễu” nhìn qua mã trải rộng. Tuy nhiên **không cải thiện SNR với nhiễu trắng rộng toàn phổ**, chỉ cải thiện khi có nhiễu hẹp dải.

Công thức toán học (trích từ Springer)

Nhóm 16 - 154166

Tín hiệu truyền đi:

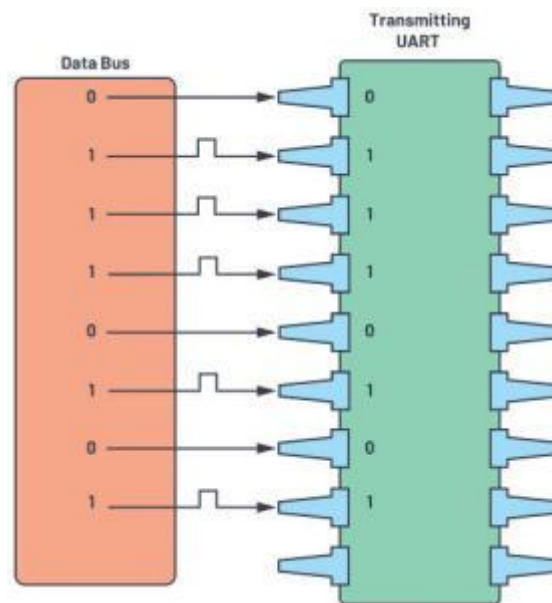
$$s(t) = A_s \cdot D(t) \cdot R\{g(t)\} \cdot \cos(2\pi f_0 t + \theta) - A_s \cdot D(t) \cdot I\{g(t)\} \cdot \sin(2\pi f_0 t + \theta)$$

Trong đó:

- A_s : biên độ
- $D(t)$: dữ liệu gốc
- $g(t)$: tín hiệu chip trải phổ (thường phức hợp)
- f_0 : tần số mang, θ : pha bắt đầu.

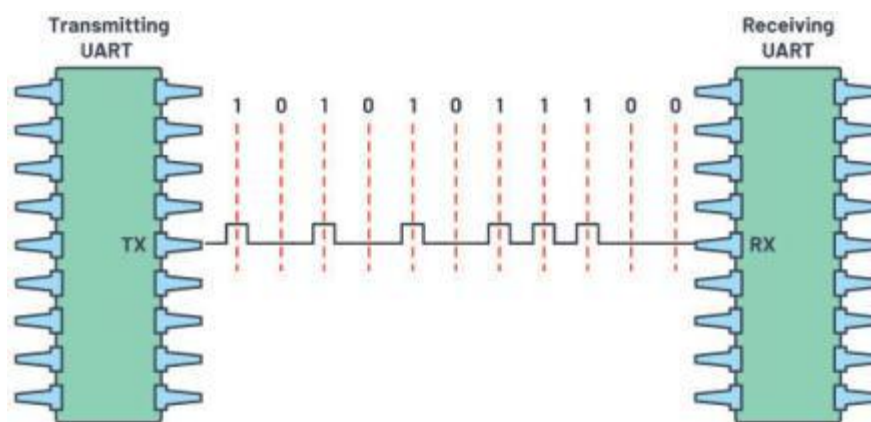
2.2. Nguyên lý hoạt động:

1. Khi muốn truyền 1 dữ liệu, máy sẽ mã hóa thành mã nhị phân 8 bit rồi đưa vào thanh ghi chứa data, Uart nhận dữ liệu song song từ bus dữ liệu.



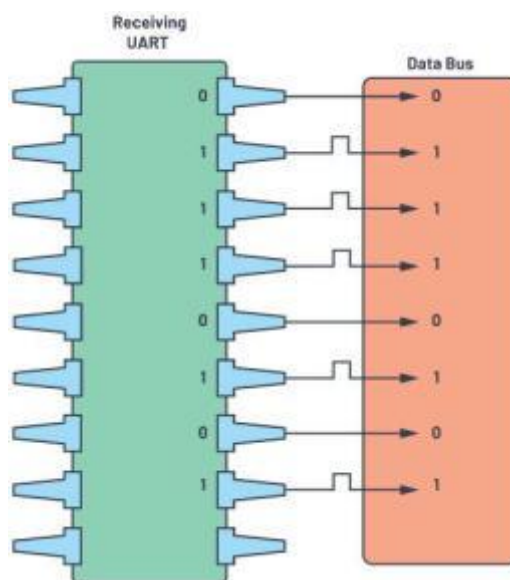
Hình 2.20 UART kênh truyền dữ liệu

2. Sau đó dữ liệu sau mã hóa được truyền đi tuần tự theo dạng nối tiếp từng bit một sang phần thu đến khi gặp stop bit.



Hình 2.21 UART truyền dữ liệu

3. Phần thu nhận được start bit tiến hành bắt đầu ghi dữ liệu 8 bit trên thanh ghi data vào thanh ghi lưu trữ và giải mã dữ liệu thu được.



Hình 2.22 UART Kênh nhận dữ liệu

2.2.1. Nguyên tắc truyền dữ liệu bằng sóng âm:

Cơ sở vật lý:

Bản chất sóng âm: Sóng âm là sóng cơ học dọc lan truyền qua môi trường đàn hồi (không khí, nước, chất rắn) nhờ sự nén dãn của các phân tử.

Thông số chính:

- **Tần số (f):** Số dao động/giây (Hz). Tai người nghe ~20Hz-20kHz. Truyền dữ liệu thường dùng siêu âm (>20kHz) để tránh nhiễu âm thanh nghe được.
- **Biên độ (A):** "Độ lớn" của sóng, liên quan đến cường độ âm thanh.
- **Bước sóng (λ):** Khoảng cách giữa hai điểm nén/dãn liên tiếp ($\lambda = v/f$, với v là vận tốc âm thanh).
- **Vận tốc (v):** Phụ thuộc môi trường (~343 m/s trong không khí ở 20°C).

Truyền trong không gian: Sóng âm suy hao theo khoảng cách (tỉ lệ nghịch với bình phương khoảng cách trong không gian tự do) và bị hấp thụ, tán xạ bởi vật cản, nhiệt độ, độ ẩm, gió. Phản xạ (gây tiếng vang) cũng ảnh hưởng đến chất lượng tín hiệu thu.

2.2.2. Truyền phát, thu nhận dữ liệu bằng sóng âm:

2.2.2.1. Phát dữ liệu:

Thiết bị phát âm thanh:

- **Loa (Speaker):** Biến đổi tín hiệu điện thành rung động cơ học tạo sóng âm. Phổ biến cho dải tần số rộng.
- **Bộ chuyển đổi siêu âm (Ultrasonic Transducer - thường dùng gốm Piezo):** Hiệu quả cao ở tần số siêu âm cụ thể (ví dụ: 40kHz). Rung khi có điện áp đặt vào.
- **Có nhiều loại thiết bị phát tuy nhiên trong mô hình của đề tài chúng ta sẽ sử dụng loa.**

Phát như thế nào:

- Tín hiệu dữ liệu số (0/1) được **mã hóa và điều chế** thành tín hiệu điện tương tự phù hợp.
- Tín hiệu điện này được **khuếch đại**.
- Tín hiệu điện khuếch đại được đưa đến **thiết bị phát** (loa/transducer).
- Thiết bị phát **rung động**, tạo ra các **thay đổi áp suất** trong không khí xung quanh, tức là tạo ra **sóng âm** mang thông tin.

Phương tiện truyền phát: Không khí là phương tiện chính. Có thể qua nước hoặc vật liệu rắn nhưng đòi hỏi thiết bị phát/thu chuyên dụng.

2.2.2.2. Thu dữ liệu:

Thiết bị thu âm thanh:

- **Microphone (Mic):** Biến đổi thay đổi áp suất âm thanh (sóng âm) thành tín hiệu điện. Có nhiều loại (điện dung, động, MEMS).
- **Bộ chuyển đổi siêu âm (Ultrasonic Transducer - gồm Piezo):** Hiệu quả cao ở tần số siêu âm. Tạo ra điện áp khi bị sóng âm tác động làm rung.
- **Có nhiều loại thiết bị thu tuy nhiên trong mô hình của đề tài chúng ta sẽ sử dụng micro.**

Thu như thế nào:

- **Sóng âm** mang thông tin tới **thiết bị thu** (mic/transducer).
- Thiết bị thu **rung động** theo sóng âm.
- Rung động này được chuyển đổi thành **tín hiệu điện tương tự yếu**.
- Tín hiệu điện được **khuếch đại tiền chế (Pre-amplified)**.
- Tín hiệu khuếch đại được đưa qua bộ **lọc** để loại bỏ nhiễu ngoài băng tần quan tâm.
- Tín hiệu đã lọc được đưa đến khối **Giải điều chế & Xử lý tín hiệu** để trích xuất dữ liệu gốc.

2.2.3. Các phương pháp điều chế tín hiệu:

Đặt vấn đề: Dòng bit nhị phân năng lượng thấp không thể truyền đi xa trên các kênh vật lý khác nhau.

→ Vì vậy ta phải điều chế tín hiệu.

Các phương pháp điều chế tín hiệu:

Phương pháp	Tên đầy đủ	Đặc điểm chính
ASK	Amplitude Shift Keying	Điều chế biên độ sóng
FSK	Frequency Shift Keying	Điều chế tần số sóng
PSK	Phase Shift Keying	Điều chế pha sóng
QAM	Quadrature Amplitude Modulation	Kết hợp điều chế biên độ và pha

Bảng 2.2 Các phương pháp điều chế tín hiệu

Thông tin cơ bản về các phương pháp được tổng hợp trong bảng dưới đây:

Tiêu chí	ASK	FSK	PSK	QAM
----------	-----	-----	-----	-----

Nguyên lý	0/1 đại diện bởi biên độ khác nhau	0/1 đại diện bởi tần số khác nhau	0/1 đại diện bởi pha khác nhau	Mỗi nhóm bit biểu diễn bằng tổ hợp pha + biên độ
Số bit/mỗi symbol	1 bit	1 bit	1 bit (BPSK), 2+ bit (QPSK)	4+ bit (tùy bậc điều chế: 16-QAM, 64-QAM,...)
Tốc độ truyền (bitrate)	Thấp	Trung bình	Trung bình đến cao	Cao
Miễn nhiễu	Kém (nhạy với nhiễu âm)	Tốt (phân biệt tần số rõ ràng)	Trung bình (nhạy pha)	Kém hơn PSK nếu nhiễu lớn
Độ phức tạp thu/giải mã	Thấp	Trung bình (cần FFT hoặc lọc tần)	Cao (cần xử lý pha chính xác)	Rất cao (cần định pha + biên độ đồng thời)
Dễ triển khai với âm thanh	Dễ, chỉ cần tăng giảm biên độ	Rất dễ, chỉ cần thay đổi tần số	Khó hơn, cần đồng bộ pha	Khó, đòi hỏi xử lý tín hiệu chính xác
Khả năng phát hiện lỗi	Yếu (biên độ dễ bị biến dạng)	Tốt hơn vì tần số ổn định hơn	Trung bình	Phụ thuộc kỹ thuật mã hóa bổ sung
Tương thích sóng âm môi trường	Tốt trong môi trường yên tĩnh	Rất tốt trong môi trường có tiếng ồn	Phụ thuộc độ ổn định thời gian	Khó giữ độ chính xác khi nhiễu cao

Bảng 2.3 Bảng so sánh các loại điều chế

Trong phạm vi của đề tài sử dụng mic và loa truyền trong khoảng cách ngắn, phương pháp điều chế FSK là phù hợp với yêu cầu. Vì vậy ta sẽ sử dụng phương pháp biến đổi FSK và dưới đây là chi tiết.

2.2.4. Mã hóa và điều chế tín hiệu:

Mục đích: Biến đổi dữ liệu số (các bit 0 và 1) thành dạng tín hiệu tương tự phù hợp để truyền bằng sóng âm, đồng thời giúp chống nhiễu và tăng hiệu quả truyền.

Mã hóa (Coding): Chuyển đổi luồng bit thô thành dạng phù hợp hơn cho điều chế hoặc phát hiện lỗi (ví dụ: Manchester encoding, các mã sửa lỗi FEC).

Điều chế (Modulation - BẮT BUỘC): Thay đổi một thông số của sóng mang (carrier wave - sóng âm tần số cao) theo tín hiệu dữ liệu.

FSK (Frequency Shift Keying - Điều chế dịch tần số):

- **Nguyên lý:** Dùng **hai tần số sóng mang** khác nhau (f_0 và f_1) để đại diện cho **hai trạng thái logic** (thường f_0 cho bit '0', f_1 cho bit '1').
- **Thực hiện trong mạch:**
 - Tạo **sóng mang** tần số cơ bản (thường là tần số trung tâm, ví dụ 40kHz).
 - Dữ liệu số điều khiển một **bộ chuyển mạch tần số (Frequency Synthesizer/VCO - Voltage Controlled Oscillator)**.
 - Khi bit dữ liệu là '0', bộ tạo dao động xuất ra tần số f_0 (ví dụ: 39kHz).
 - Khi bit dữ liệu là '1', bộ tạo dao động xuất ra tần số f_1 (ví dụ: 41kHz).
 - Tín hiệu FSK được khuếch đại và đưa ra thiết bị phát.
- **Ưu điểm:** Tương đối đơn giản, chống nhiễu tốt hơn ASK (Amplitude Shift Keying) vì nhiễu thường ảnh hưởng biên độ hơn là tần số (nếu khoảng cách $|f_1 - f_0|$ đủ lớn).
- **Nhược điểm:** Tốc độ dữ liệu thấp hơn so với một số kỹ thuật khác (như PSK), cần băng thông kênh rộng hơn ASK.

2.2.5. Giải điều chế và xử lý tín hiệu:

Mục đích: Tái tạo lại dữ liệu số gốc từ tín hiệu âm thanh đã thu (đã được khuếch đại và lọc).

Giải điều chế FSK:

- **Nguyên lý:** Phát hiện xem tần số nào (f_0 hay f_1) đang chiếm ưu thế trong tín hiệu thu tại mỗi khoảng thời gian bit (bit period).

Các thuật toán/phương pháp trong mạch:

1. Bộ lọc thông dải kép (Dual Bandpass Filters):

Sử dụng hai bộ lọc thông dải: Một lọc tập trung quanh f_0 , một lọc tập trung quanh f_1 .

Tín hiệu thu được đưa đồng thời qua cả hai bộ lọc.

Đầu ra mỗi bộ lọc được **chỉnh lưu (rectified)** và **lọc lấy bao hình (low-pass filtered)** để lấy biên độ tín hiệu (envelope) ở mỗi tần số.

2. Bộ so sánh (Comparator):

So sánh biên độ đầu ra của bộ lọc f_0 và bộ lọc f_1 .

Nếu biên độ $f_0 > \text{biên độ } f_1 \Rightarrow$ Quyết định bit '0'.

Nếu biên độ $f_1 > \text{biên độ } f_0 \Rightarrow$ Quyết định bit '1'.

3. Vòng giữ pha (PLL - Phase-Locked Loop) + Bộ tách tần số (Frequency Discriminator):

PLL: Khóa vào tần số trung tâm giữa f_0 và f_1 .

Bộ tách tần số: Chuyển đổi sự thay đổi tần số tức thời trong tín hiệu thu (quanh tần số trung tâm) thành sự thay đổi điện áp tương tự.

Bộ so sánh ngưỡng (Threshold Comparator): So sánh điện áp này với một ngưỡng (thường là 0V).

Nếu điện áp $> \text{ngưỡng} \Rightarrow$ Quyết định bit '1'.

Nếu điện áp $< \text{ngưỡng} \Rightarrow$ Quyết định bit '0'.

4. Xử lý tín hiệu số (DSP - Digital Signal Processing):

Tín hiệu tương tự thu được chuyển đổi sang số bằng ADC (Analog-to-Digital Converter).

Sử dụng các thuật toán DSP (ví dụ: **Biến đổi Fourier nhanh FFT** hoặc các bộ lọc số) để phân tích phổ tần số của tín hiệu trong từng khoảng thời gian bit.

Xác định tần số nào (f_0 hay f_1) có năng lượng mạnh nhất trong khoảng thời gian đó để quyết định bit dữ liệu.

Phương pháp này linh hoạt và mạnh mẽ nhất, có thể xử lý nhiễu tốt, nhưng phức tạp hơn về phần cứng và phần mềm.

5. Xử lý tín hiệu tiếp theo (Sau giải điều chế):

Lấy mẫu và Giữ (Sampling & Holding): Lấy mẫu tín hiệu ra từ bộ giải điều chế tại thời điểm tối ưu trong mỗi bit period (thường ở giữa bit) để giảm lỗi.

So sánh ngưỡng (Threshold Comparison - nếu dùng PLL/DSP): Như mô tả ở trên.

Tái tạo xung đồng hồ (Clock Recovery): Đồng bộ hóa thời gian lấy mẫu với tốc độ bit của dữ liệu phát. Rất quan trọng cho truyền dẫn dài/nhiều bit.

Giải mã (Decoding - nếu có mã hóa): Đảo ngược quá trình mã hóa (ví dụ: giải mã Manchester) hoặc sửa lỗi (FEC).

Đệm dữ liệu (Data Buffering): Tạm lưu dữ liệu bit đã khôi phục trước khi gửi đến hệ thống xử lý chính (vi điều khiển, máy tính).

2.3. Yêu cầu chức năng:

Phân phát:

Mã hóa dữ liệu nhị phân (ký tự, số, chuỗi) thành tín hiệu âm thanh sử dụng kỹ thuật điều chế đơn giản (FSK). Xuất tín hiệu ra loa để phát truyền qua không khí.

Phân thu:

Thu sóng âm bằng micro analog, chuyển tín hiệu thành số thông qua ADC. Giải điều chế tín hiệu để khôi phục lại dữ liệu gốc.

Xử lý tín hiệu và hiển thị:

Hiển thị dữ liệu thu được lên màn hình LCD để người dùng dễ theo dõi. Có thể mở rộng sang gửi dữ liệu qua Wi-Fi nếu cần.

Giao tiếp đơn giản – một chiều:

Giao tiếp dữ liệu theo một chiều cố định: từ ESP32 phát → ESP32 thu. Đảm bảo truyền ổn định trong khoảng cách dưới 1m.

Thiết lập tham số cấu hình:

Cho phép thay đổi các thông số như tần số phát, tốc độ truyền (baudrate mô phỏng), thời gian giữa các gói, để thích ứng với các điều kiện môi trường khác nhau.

2.4. Yêu cầu phi chức năng:

Hiệu suất:

Hệ thống cần truyền và nhận dữ liệu trong thời gian ngắn (dưới 1 giây cho mỗi byte), đảm bảo tín hiệu âm thanh không bị chồng lấp hay sai lệch do trễ xử lý.

Độ tin cậy:

Hệ thống phải hoạt động ổn định trong môi trường có nhiễu âm nhẹ, như tiếng quạt hoặc tiếng nói chuyện thông thường. Phải có cơ chế phát hiện lỗi giải mã để tránh truyền sai dữ liệu.

Bảo mật:

Hệ thống cần có cơ chế hạn chế lỗi truyền dữ liệu do nhiễu âm, tránh truyền sai hoặc nhầm lệnh. Có thể áp dụng mã kiểm tra đơn giản (checksum, parity bit...) để xác thực tính toàn vẹn dữ liệu.

Tính dễ sử dụng:

Giao diện hiển thị dữ liệu (qua Serial hoặc màn hình OLED) phải trực quan, dễ quan sát. Hệ thống dễ cấu hình tần số, tốc độ truyền hoặc reset mà không cần thao tác phức tạp.

Khả năng mở rộng:

Cho phép mở rộng để truyền nhiều loại dữ liệu khác nhau (ký tự, số, chuỗi...). Có thể nâng cấp lên giao tiếp hai chiều hoặc tích hợp Wi-Fi/Bluetooth để giám sát từ xa.

Khả năng bảo trì:

Mạch điện đơn giản, sử dụng linh kiện thông dụng, dễ dàng thay thế. Phần mềm được chia module rõ ràng để dễ kiểm tra và nâng cấp khi cần.

Tiết kiệm năng lượng:

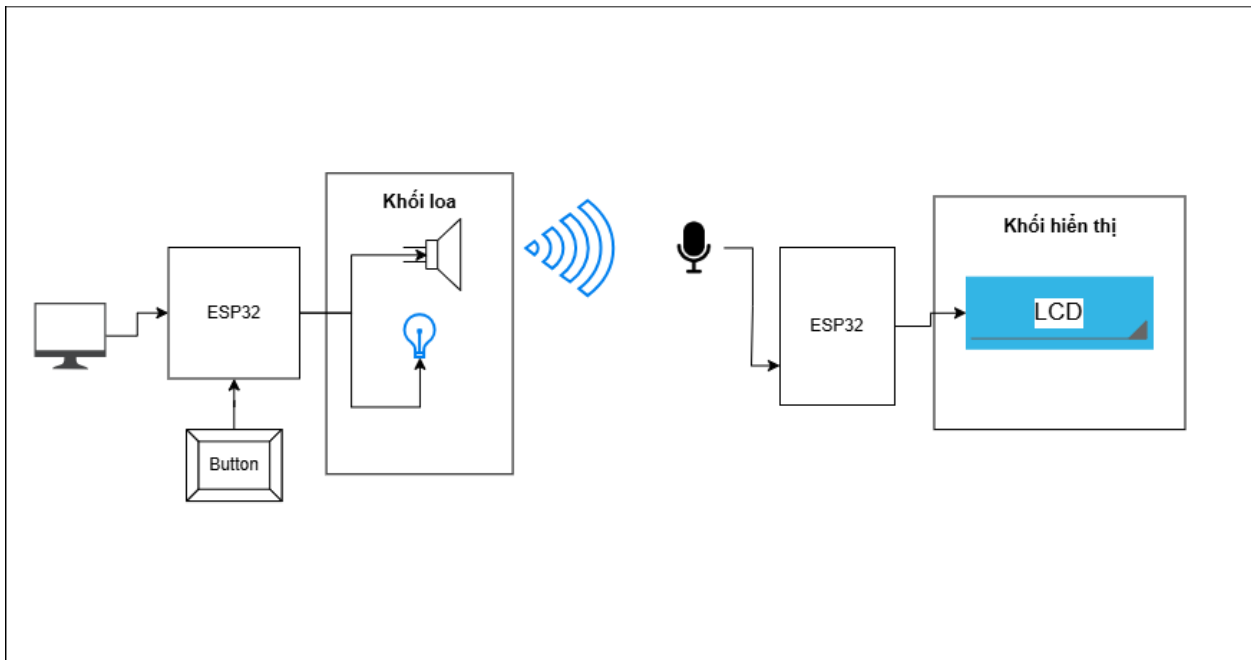
Hệ thống chỉ truyền hoặc nhận dữ liệu khi người dùng nhấn nút (button). Khi không có thao tác, mạch không phát hoặc xử lý tín hiệu âm thanh, giúp giảm tiêu thụ năng lượng và tránh hoạt động không cần thiết.

CHƯƠNG 3. THIẾT KẾ HỆ THỐNG

3.1. Tổng quan hệ thống:

Với những yêu cầu đặt ra ở phần trước, chúng em đề xuất một hệ thống truyền – thu dữ liệu bằng âm thanh sử dụng ESP32, bao gồm bốn khối chính: **khối phát tín hiệu**, **khối thu tín hiệu**, **khối xử lý tín hiệu** và **khối hiển thị**.

- **Khối phát tín hiệu** sử dụng một vi điều khiển ESP32 làm trung tâm điều khiển, kết nối với máy tính để nhận dữ liệu đầu vào dưới dạng văn bản. Khi nút nhấn được kích hoạt, dữ liệu sẽ được mã hóa bằng thư viện GGWave thành các tín hiệu âm thanh có tần số đặc trưng (FSK), sau đó được phát ra môi trường thông qua loa. Khối này cũng bao gồm một đèn LED để báo hiệu trạng thái đang phát.
- **Khối thu tín hiệu** bao gồm một microphone có nhiệm vụ thu lại các tín hiệu âm thanh được phát ra từ khối phát. Tín hiệu âm thanh sau đó được chuyển đến một vi điều khiển ESP32 thứ hai.
- **Khối xử lý tín hiệu** trên ESP32 sẽ sử dụng thư viện GGWave để giải mã dữ liệu âm thanh nhận được và khôi phục lại chuỗi văn bản ban đầu.
- **Khối hiển thị** đảm nhiệm việc hiển thị dữ liệu sau khi được giải mã lên màn hình LCD, giúp người dùng kiểm tra nội dung truyền nhận.

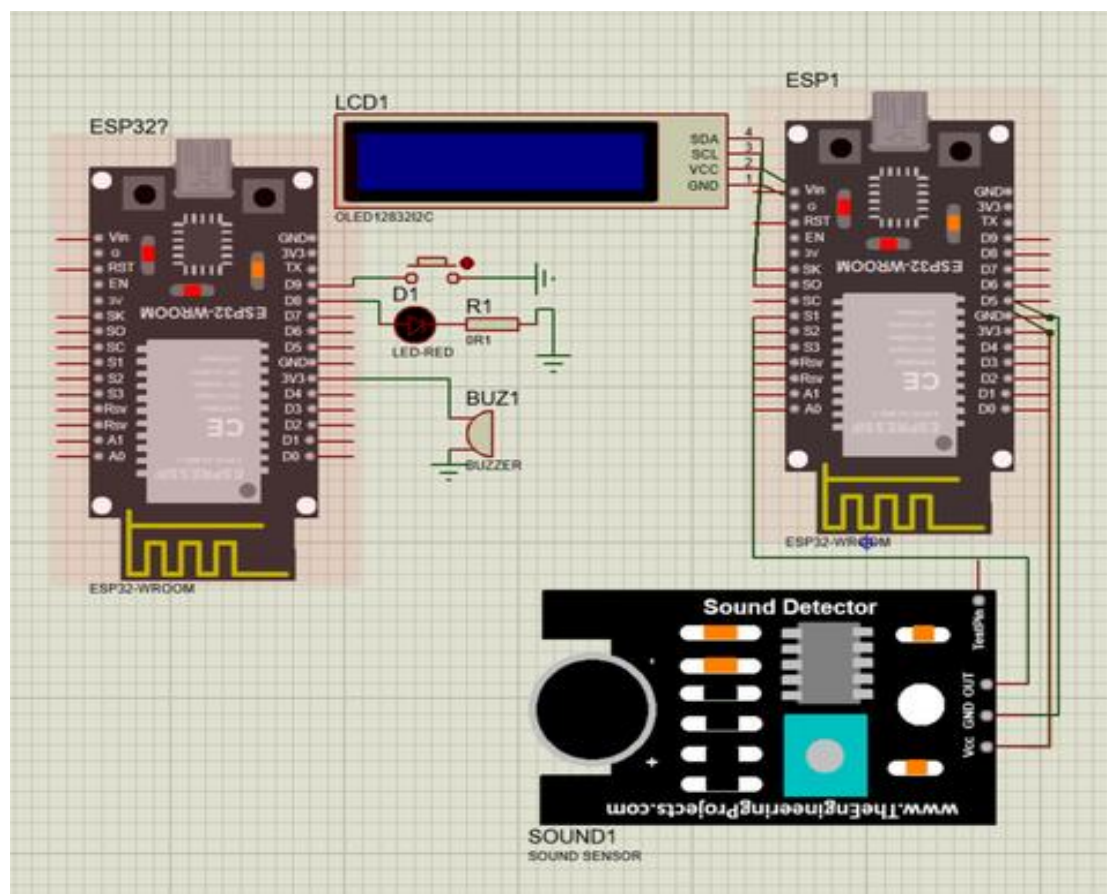


Hình 3.1 Sơ đồ tổng quan hệ thống

- **Khối phát tín hiệu** sử dụng một vi điều khiển ESP32 làm trung tâm điều khiển, kết nối với máy tính để nhận dữ liệu đầu vào dưới dạng văn bản. Khi nút nhấn được kích hoạt, dữ liệu sẽ được mã hóa bằng thư viện GGWave thành các tín hiệu âm thanh có tần số đặc trưng (FSK), sau đó được phát ra môi trường thông qua loa. Khối này cũng bao gồm một đèn LED để báo hiệu trạng thái đang phát.
- **Khối thu tín hiệu** bao gồm một microphone có nhiệm vụ thu lại các tín hiệu âm thanh được phát ra từ khối phát. Tín hiệu âm thanh sau đó được chuyển đến một vi điều khiển ESP32 thứ hai.
- **Khối xử lý tín hiệu** trên ESP32 sẽ sử dụng thư viện GGWave để giải mã dữ liệu âm thanh nhận được và khôi phục lại chuỗi văn bản ban đầu.
- **Khối hiển thị** đảm nhiệm việc hiển thị dữ liệu sau khi được giải mã lên màn hình LCD, giúp người dùng kiểm tra nội dung truyền nhận

3.2. Phương án thiết kế:

3.2.1. Thiết kế phần cứng:



Hình 3.2 Sơ đồ tổng quan thiết kế mạch

Phần phát:

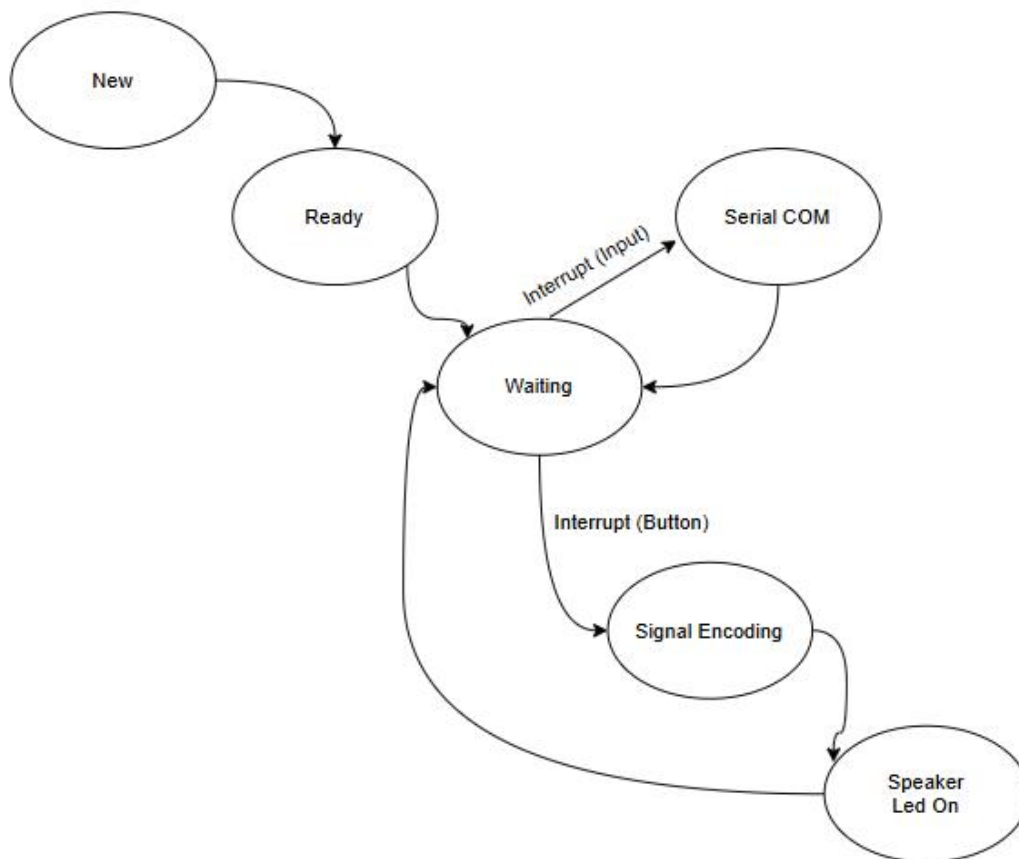
- **ESP32** giữ vai trò là thiết bị phát.
- **BUZ1 (buzzer)**: được kết nối với một chân GPIO của ESP32, phát âm thanh mang dữ liệu đã mã hóa bằng phương pháp điều chế tần số (FSK) do thư viện GGWave sinh ra.
- **LED báo trạng thái (D1 + R1)**: sáng lên khi hệ thống đang truyền dữ liệu. LED được điều khiển qua một transistor hoặc trực tiếp qua điện trở hạn dòng.
- **Nguồn cấp**: ESP32 được cấp nguồn từ cổng USB (Vin), đồng thời cung cấp nguồn 3.3V cho các linh kiện.

Phần thu:

- **ESP32** (nhãn ESP1 trong sơ đồ) đảm nhận vai trò thu tín hiệu âm thanh.

- **Sound Detector (SOUND1):** là module microphone dùng để thu tín hiệu âm thanh phát ra từ buzzer. Thiết bị này có đầu ra dạng analog hoặc digital, kết nối với chân ADC của ESP32.
- Tín hiệu âm thanh thu được sẽ được giải mã bởi ESP32 thông qua thư viện GGWave để khôi phục lại chuỗi dữ liệu ban đầu.
- **Màn hình LCD OLED (LCD1):** được kết nối với ESP32 thu qua giao tiếp I2C (chân SDA và SCL).
- Sau khi dữ liệu âm thanh được giải mã thành công, nó sẽ được hiển thị lên màn hình LCD để người dùng kiểm tra nội dung truyền nhận

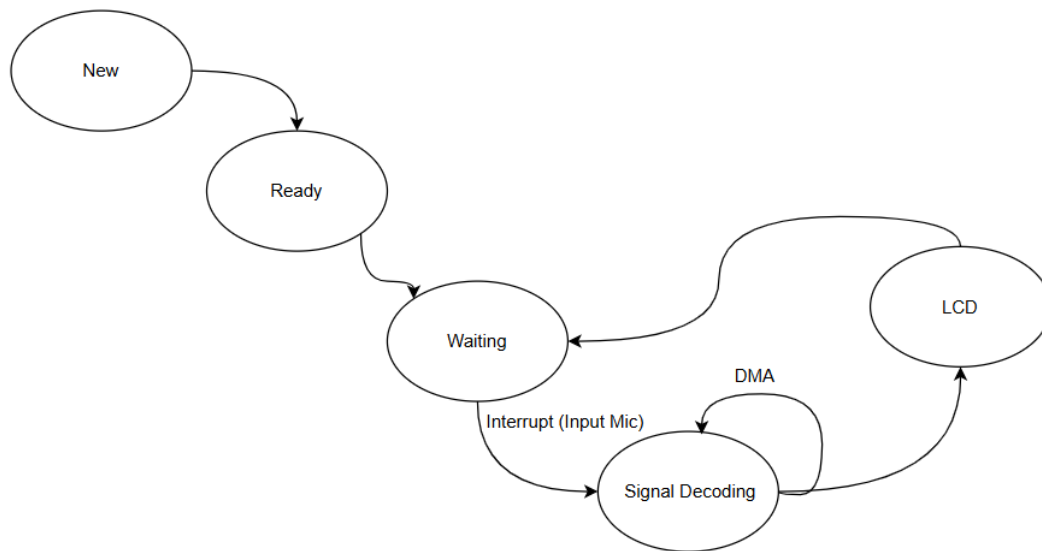
3.2.2. Thiết kế phần mềm:



Hình 3.3 Sơ đồ trạng thái mạch phát tín hiệu

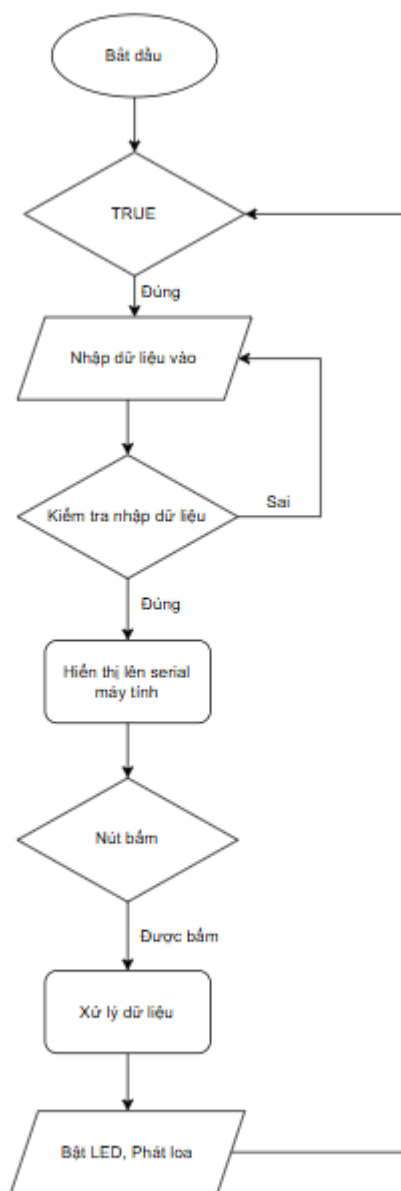
Sơ đồ trạng thái cho thấy hệ thống được thiết kế theo dạng sự kiện điều khiển (event-driven), có khả năng phản hồi nhanh với các hành động nhập dữ liệu hoặc nút nhấn từ người dùng. Việc chia tách rõ ràng các trạng thái

giúp giảm thiểu lỗi, dễ theo dõi và mở rộng về sau. Sơ đồ cũng phản ánh mối liên hệ chặt chẽ giữa phần mềm điều khiển và các phần tử ngoại vi như nút nhấn, LED, và loa.



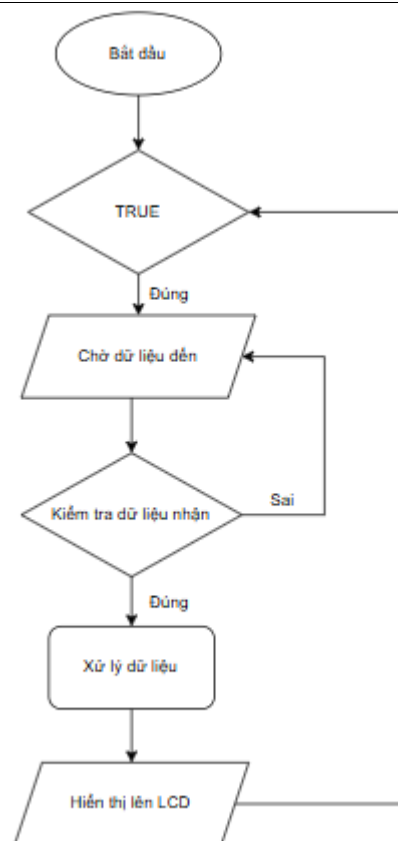
Hình 3.4 Sơ đồ trạng thái mạch thu tín hiệu

Sơ đồ trạng thái thể hiện cách tổ chức hợp lý giữa thu tín hiệu, xử lý và hiển thị, đảm bảo hệ thống có thể hoạt động gần thời gian thực (quasi real-time). Việc tích hợp DMA giúp giảm tải CPU trong quá trình lấy mẫu ADC từ microphone, trong khi các ngắt đảm bảo hệ thống phản ứng nhanh với tín hiệu đến. Sự tuần tự và khép kín của các trạng thái giúp đảm bảo tính ổn định và khả năng mở rộng cho hệ thống thu dữ liệu âm thanh.



Hình 3.5 Lưu đồ thuật toán phần phát

Sơ đồ thuật toán hoạt động của khối phát được trình bày ở Hình 3.3. Sau khi được cấp nguồn, vi điều khiển ESP32 tiến hành khởi tạo các cổng vào/ra như Serial, chân LED, chân nút nhấn và cấu hình đầu ra loa. Đồng thời, thư viện GGWave cũng được khởi tạo



Hình 3.6 Lưu đồ thuật toán phần thu

Thuật toán hoạt động của khối thu dữ liệu âm thanh được thể hiện ở Hình 3.4. Vi điều khiển ESP32 thứ hai có nhiệm vụ lắng nghe tín hiệu âm thanh được phát ra từ khối phát, sau đó giải mã và hiển thị dữ liệu nhận được lên màn hình LCD.

Sau khi được cấp nguồn, vi điều khiển thực hiện các thao tác khởi tạo như: thiết lập chân giao tiếp với microphone/sound sensor, khởi tạo LCD, và cấu hình thư viện GGWave ở chế độ giải mã tín hiệu (RX mode).

Hệ thống bước vào vòng lặp chính, liên tục kiểm tra xem có dữ liệu âm thanh đến hay không. Khi phát hiện tín hiệu âm thanh FSK, hệ thống bắt đầu quá trình lấy mẫu

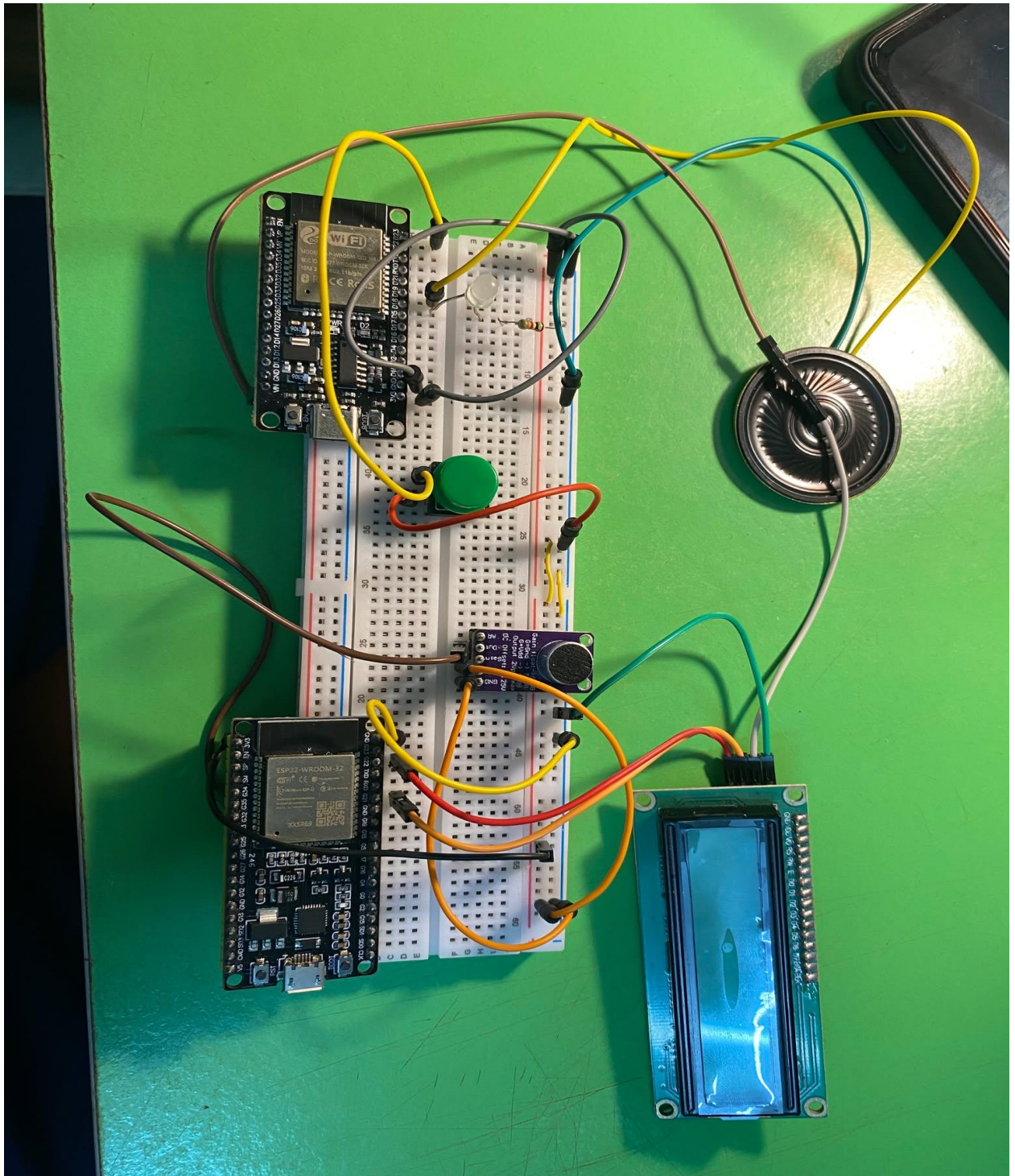
<p>với các thông số như tần số lấy mẫu, định dạng mẫu và kiểu điều chế âm thanh.</p> <p>Hệ thống sau đó chờ dữ liệu văn bản được nhập vào từ máy tính thông qua Serial. Khi có dữ liệu nhập vào, vi điều khiển tiến hành kiểm tra độ hợp lệ của chuỗi (không rỗng, đúng ký tự,...). Nếu dữ liệu hợp lệ, chuỗi sẽ được hiển thị lại lên Serial như một bước xác nhận và lưu vào bộ nhớ tạm của ESP32.</p> <p>Tiếp theo, hệ thống chuyển sang chế độ chờ nút nhấn. Khi nút được nhấn, vi điều khiển tiến hành bật LED báo hiệu và sử dụng thư viện GGWave để mã hóa chuỗi thành dãy tần số âm thanh tương ứng. Dãy âm thanh này được phát qua loa để truyền tới thiết bị thu.</p> <p>Sau khi phát xong, hệ thống quay lại trạng thái ban đầu và sẵn sàng nhận chuỗi mới từ người dùng. Quá trình lặp lại liên tục cho đến khi kết thúc phiên làm việc hoặc ngắt nguồn.</p>	<p>tín hiệu đầu vào và giải mã chuỗi văn bản từ chuỗi tần số thu được.</p> <p>Dữ liệu sau khi được giải mã sẽ được kiểm tra độ hợp lệ (ví dụ: có đúng định dạng không, không bị lỗi nhiễu sóng, không thiếu ký tự,...). Nếu dữ liệu không hợp lệ, hệ thống tiếp tục chờ tín hiệu mới. Nếu dữ liệu hợp lệ, nó sẽ được hiển thị lên màn hình LCD để người dùng quan sát kết quả truyền.</p> <p>Sau khi hiển thị xong, hệ thống quay lại trạng thái chờ và sẵn sàng nhận tín hiệu tiếp theo.</p>
--	---

➔ Việc kết hợp DSS với Reed–Solomon:

- Reed–Solomon chịu trách nhiệm bảo đảm sửa lỗi dữ liệu trong trường hợp mất hoặc nhầm nibble/tone.
- DSS trải phổ tín hiệu trước khi điều chế FSK, giúp chống nhiễu môi trường (âm thanh, phản xạ) và tăng mức độ bảo mật của tín hiệu.
- Sau khi mã hóa Reed–Solomon, gói dữ liệu được DSS-scramble bằng XOR với chuỗi code. Khi truyền, việc despread sẽ phục hồi dữ liệu gốc trước khi giải điều chế FSK.

CHƯƠNG 4. TRIỂN KHAI SẢN PHẨM

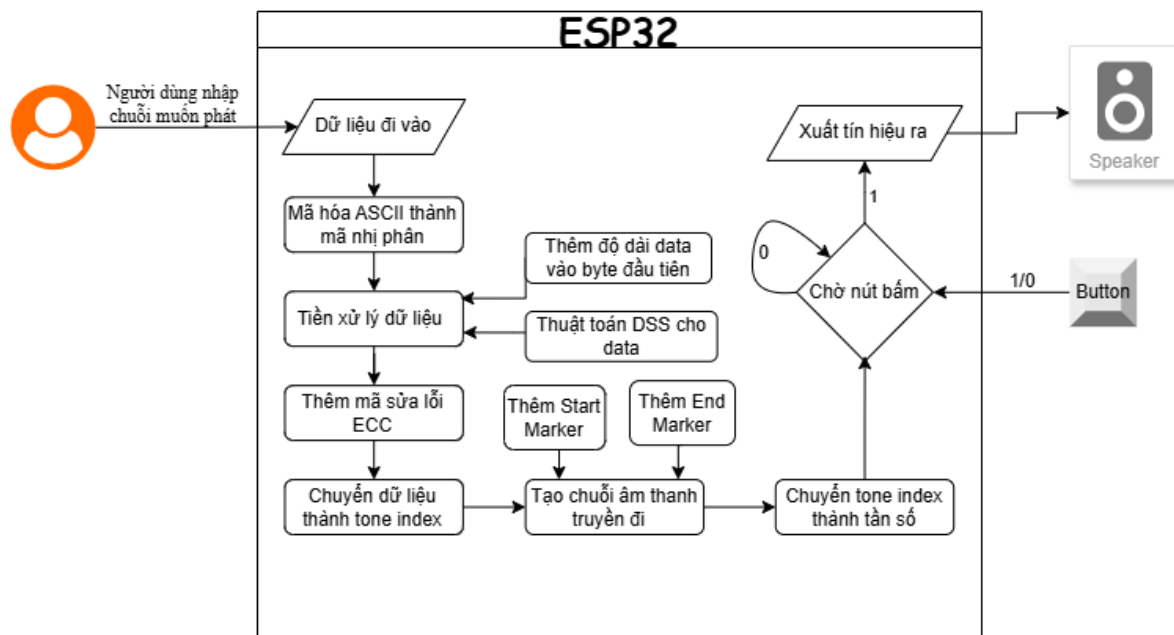
4.1. Triển khai phần cứng:



Hình 4.1 Mạch triển khai thực tế

4.2. Triển khai phần mềm:

4.2.1. Đầu phát:



Hình 4.2 Sơ đồ hệ thống đầu phát

Quy trình hoạt động của phần phát:

1. **Tiền xử lý dữ liệu:** thêm thông tin độ dài payload vào byte đầu tiên, áp dụng DSS (Direct Sequence Spread).
2. **Thêm mã sửa lỗi (ECC)** bằng Reed-Solomon.
3. **Chuyển dữ liệu thành chỉ số âm thanh (tone indices).**
4. **Chuyển tone index thành tần số thật (Hz).**
5. **Phát chuỗi âm thanh:** marker bắt đầu, data, marker kết thúc.

Chi tiết:

4.2.1.1. Tiền xử lý dữ liệu:

Mục tiêu: Chuyển chuỗi ký tự ASCII thành chuỗi có tính ngẫu nhiên hóa, giúp hệ thống âm thanh nhận diện chính xác

Các bước:

a. Bước 1: Thêm độ dài dữ liệu vào byte đầu tiên

Mục đích:

- Giúp phía thu biết cần bao nhiêu byte dữ liệu truyền đi
- Hỗ trợ bộ giải mã xác định được điểm đầu và kết thúc thông điệp

Mô phỏng với chuỗi “Hello”:

Giả sử gửi chuỗi “hello” gồm 5 ký tự (5byte)

“Hello” = [‘H’, ‘e’, ‘l’, ‘l’, ‘o’] = [0x48, 0x65, 0x6C, 0x6C, 0x6F]

Chèn thêm byte đầu tiên là 0x05 (giá trị thập phân = 5):

data = [0x05, 0x48, 0x65, 0x6C, 0x6C, 0x6F]

[00000101, 01001000, 01100101, 01101100, 01101100, 01101111] (Dạng nhị phân)

b. Bước 2: DSS:

Mục đích:

- Làm ngẫu nhiên hóa dữ liệu (Tránh các mẫu lặp dễ bị đồng bộ sai)
- Là một dạng trải phổ => Giảm các bit 0 hay 1 lặp đi lặp lại quá nhiều cùng lúc => Tăng khả năng chống nhiễu

Chuỗi cố định đặt để XOR với giá trị dữ liệu:

```
1 constexpr int kDSSMagicSize = 64;
2 const uint8_t kDSSMagic[kDSSMagicSize] PROGMEM = {
3     0x96, 0x9f, 0xb4, 0xaf, 0x1b, 0x91, 0xde, 0xc5, 0x45, 0x75, 0xe8, 0x2e, 0x0f, 0x32, 0x4a, 0x5f,
4     0xb4, 0x56, 0x95, 0xcb, 0x7f, 0x6a, 0x54, 0x6a, 0x48, 0xf2, 0x0b, 0x7b, 0xcd, 0xfb, 0x93, 0x6d,
5     0x3c, 0x77, 0x5e, 0xc3, 0x33, 0x47, 0xc0, 0xf1, 0x71, 0x32, 0x33, 0x27, 0x35, 0x68, 0x47, 0x1f,
6     0x4e, 0xac, 0x23, 0x42, 0x5f, 0x00, 0x37, 0xa4, 0x50, 0x6d, 0x48, 0x24, 0x91, 0x7c, 0xa1, 0x4e,
7 };
```

Mô phỏng với chuỗi “Hello”:

data = [0x05, 0x48, 0x65, 0x6C, 0x6C, 0x6F]

[00000101, 01001000, 01100101, 01101100, 01101100, 01101111] (Dạng nhị phân)

Nhóm 16 - 154166

Ta thực hiện XOR với các giá trị trong mảng kDSSMagic: (ko XOR byte đầu vì là byte dữ liệu độ dài khung, cần dữ liệu ổn định, không bị mã hóa)

$data[1] = 0x48 \oplus kDSSMagic[0] = 0xDE$

$data[2] = 0x65 \oplus kDSSMagic[1] = 0xFA$

$data[3] = 0x6C \oplus kDSSMagic[2] = 0xD8$

$data[4] = 0x6C \oplus kDSSMagic[3] = 0xC3$

$data[5] = 0x6F \oplus kDSSMagic[4] = 0x74$

Sau DDS: [0x05, 0xDE, 0xFA, 0xD8, 0xC3, 0x74]

[00000101 11011110 11111010 11011000 11000011
01110100]

```
Sending text: Hello
Binary: 01001000 01100101 01101100 01101100 01101111
TX Data (17 bytes):
Byte 0: 0x10
Byte 1: 0xDE
Byte 2: 0xFA
Byte 3: 0xD8
Byte 4: 0xC3
Byte 5: 0x74
```

Hình 4.3 Dữ liệu thu được thực tế sau DSS

4.2.1.2. Thêm mã sửa lỗi (Reed-Solomon)

Mục đích: Thêm khả năng sửa lỗi vào dữ liệu, để nếu tín hiệu âm thanh bị méo, mất tone, hoặc nhiễu thì vẫn khôi phục được dữ liệu ban đầu.

Cơ bản các bước trong REED-SOLOMON:

- Là mã sửa lỗi hoạt động trên byte
- Có thể phát hiện và sửa nhiều lỗi byte (tùy số lượng ECC bytes).

Mô phỏng với chuỗi “Hello”:

Dữ liệu sau DSS:

Index	Hex	Binary	Ghi chú
0	0x05	00000101	Độ dài dữ liệu 5 byte
1	0xDE	11011110	H

2	0xfa	11111010	e
3	0xD8	11011000	l
4	0xC3	11000011	l
5	0x74	01110100	o

Tổng cộng 6 byte trong đó có data[0] = 5 nghĩa là có 5 byte dữ liệu từ data[1] đến data[5]

- **ECC (Error Correction Code)** là **mã sửa lỗi** – một kỹ thuật **thêm vào dữ liệu truyền** để giúp **phát hiện và sửa lỗi** khi dữ liệu bị sai do nhiễu, mất mát, hoặc méo tín hiệu.
- ECC không phải là dữ liệu gốc mà được tính từ dữ liệu gốc nên khi có lỗi thì nó có thể suy ngược lại dữ liệu ban đầu

Xác định số byte ECC theo độ dài dữ liệu:

- Nếu ≤ 10 byte ECC = 4 byte
- Nếu ≤ 20 byte ECC = 8 byte
- Còn lại ECC = 16 byte

Vì sao ko tính byte data[0]:

- Là byte độ dài data: Nó chỉ định có bao nhiêu byte dữ liệu thực tế, nên không được sửa hoặc mã hóa
- Bộ giải mã cần đọc byte này trước => Để xác định số byte payload, từ đó mới biết cần lấy bao nhiêu byte tiếp theo để giải mã

Thành phần	Có dùng trong ECC?	Ghi chú
data[0] (length)	KHÔNG	Không được sửa/mã hóa
data[1..N] (payload)	CÓ	Được đưa vào tạo ECC (Reed–Solomon)
ECC byte	Có mặt ở cuối	Dùng để sửa lỗi cho payload

→ Mỗi byte là 1 hệ số trong đa thức (trên GF(256)):

$$P(x) = 0xDE \cdot x^4 + 0xFA \cdot x^3 + 0xD8 \cdot x^2 + 0xC3 \cdot x^1 + 0x74$$

Ta nhân P(x) với x^4 để dời bậc lên:

$$P(x) \cdot x^4 = DE \cdot x^8 + FA \cdot x^7 + D8 \cdot x^6 + C3 \cdot x^5 + 74 \cdot x^4$$

Rồi chia cho **đa thức sinh G(x)**. Một ví dụ G(x) thường dùng trong RS(255, 251) là:

$$G(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2)(x - \alpha^3)$$

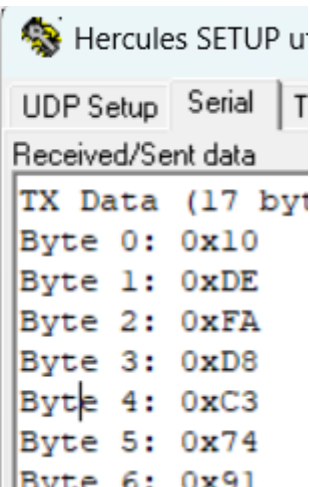
Trong đó α là phần tử sinh trong GF(256) (thường = 0x02)

Kết quả: ECC = [0x58, 0x3E, 0x94, 0xC5]

→ Đây là phần dư của phép chia: $R(x) = 58 \cdot x^3 + 3E \cdot x^2 + 94 \cdot x + C5$

Byte dữ liệu: [0x05, 0xDE, 0xFA, 0xD8, 0xC3, 0x74, 0x58, 0x3E, 0x94, 0xC5]
bao gồm data[0]: Chứa độ dài dữ liệu cần truyền đi, data[1]->data[5]: chứa byte dữ liệu đã qua DSS, data[6]->data[9]: Chứa các byte ECC.

Byte	Giá trị	Nhị phân	High nibble
0	0x05	00000101	0000 = 0
1	0xDE	11011110	1101 = 13
2	0xFA	11111010	1111 = 15
3	0xD8	11011000	1101 = 13
4	0xC3	11000011	1100 = 12
5	0x74	01110100	0111 = 7
6	0x58	01011000	0101 = 5



Hình 4.4 Dữ liệu thu được thực tế sau thêm ECC

4.2.1.3. Chuyển dữ liệu thành chỉ số âm thanh (tone indices).

Mục tiêu: Chuyển các byte dữ liệu thành dãy số tone index, mỗi số tương ứng với một âm tần (Frequency) để truyền đi.

Ý tưởng:

Trong протоко MT_FASTEST:

- Mỗi byte chia thành 2 phần:
 - High nibble = 4 bit cao (4-7)
 - Low nibble = 4 bit thấp (0-3)
- Mỗi nibble có giá trị từ 0 đến 15 → tương ứng với **1 tone index**
- Kết quả: **Mỗi byte → 2 tone index**
- Tổng cộng: 10 byte → **20 tone index**

Mô phỏng với chuỗi “Hello”:

[0x05, 0xDE, 0xFA, 0xD8, 0xC3, 0x74, 0x58, 0x3E, 0x94, 0xC5]

Cách chia byte thành tone index:

0xDE:

Binary: 11011110

High nibble: 1101 = 13 (tone index = 13)

Low nibble: 1110 = 14 (tone index = 14)

→ 4 bit biểu diễn nên có max 16 tone index từ 0->15

Cụ thể từng byte:

Byte	Giá trị	Nhị phân	High nibble	Low nibble	Tone index
0	0x05	00000101	0000 = 0	0101 = 5	→ [5, 0]
1	0xDE	11011110	1101 = 13	1110 = 14	→ [14, 13]
2	0xFA	11111010	1111 = 15	1010 = 10	→ [10, 15]
3	0xD8	11011000	1101 = 13	1000 = 8	→ [8, 13]
4	0xC3	11000011	1100 = 12	0011 = 3	→ [3, 12]
5	0x74	01110100	0111 = 7	0100 = 4	→ [4, 7]
6	0x58	01011000	0101 = 5	1000 = 8	→ [8, 5]
7	0x3E	00111110	0011 = 3	1110 = 14	→ [14, 3]
8	0x94	10010100	1001 = 9	0100 = 4	→ [4, 9]
9	0xC5	11000101	1100 = 12	0101 = 5	→ [5, 12]

→ Kết quả tone index theo thứ tự gửi: [5, 0, 14, 13, 10, 15, 8, 13, 3, 12, 4, 7, 8, 5, 14, 3, 4, 9, 5, 12]

Lý do chia nibble thay vì theo byte:

Vì 1 byte có 256 giá trị -> quá nhiều tone gây khó phân biệt, dễ sai tần số

1 nibble có 16 giá trị -> ít tone hơn

Thứ tự tone là: low nibble trước, high nibble sau

4.2.1.4. Chuyển tone index sang tần số Hz:

Mục tiêu: Dùng danh sách tone index từ bước 4 để tính ra tần số âm thanh thực tế mà ESP32 đã phát.

Công thức tính : $\text{freq} = (\text{freqStart} + \text{tone_index}) \times \text{hzPerSample}$

Tham số	Giá trị
freqStart	24

hzPerSample	$6000 / 128 = \mathbf{46.875 \text{ Hz}}$
Tone index	Là số trong khoảng [0..255]

Bảng 4.1 Các giá trị tính toán cụ thể cho protocol MT_FASTEST

Mô phỏng với chuỗi “Hello”:

Chuỗi “Hello” cần dùng 31 tone vì còn các tone start marker và end marker

Áp dụng công thức tính: $\text{freq} = (24 + \text{tone_index}) \times 46.875$

- tone = 5 $\rightarrow \text{freq} = (24 + 5) \times 46.875 = 29 \times 46.875 = \mathbf{1359.375 \text{ Hz}}$
- tone = 0 $\rightarrow \text{freq} = (24 + 0) \times 46.875 = 24 \times 46.875 = \mathbf{1125 \text{ Hz}}$
- tone = 14 $\rightarrow \text{freq} = (24 + 14) \times 46.875 = 38 \times 46.875 = \mathbf{1781.25 \text{ Hz}}$

Tone Index	Frequency (Hz)
5	1359.375
0	1125
14	1781.25
13	1734.375
10	1593.75
15	1828.125
8	1500
13	1734.375
3	1265.625
12	1687.5
4	1312.5
7	1453.125
8	1500
5	1359.375
14	1781.25
3	1265.625
4	1312.5
9	1546.875
5	1359.375
12	1687.5

Bảng 4.2 Danh sách tần số đầy đủ với chuỗi “Hello”

Tổng kết: Dãy tần số sẽ phát đi là: [1359.375, 1125, 1781.25, 1734.375, 1593.75, 1828.125, 1500, 1734.375, 1265.625, 1687.5, 1312.5, 1453.125, 1500, 1359.375, 1781.25, 1265.625, 1312.5, 1546.875, 1359.375, 1687.5] Hz

4.2.1.5. Tạo chuỗi âm thanh

Mục đích:

- Giúp bộ thu nhận biết bắt đầu truyền số liệu
- Có dạng mẫu đặc biệt, dễ phát hiện

Mô phỏng với chuỗi “Hello”:

dataTones = [5, 0, 14, 13, 10, 15, 8, 13, 3, 12, 4, 7, 8, 5, 14, 3, 4, 9, 5, 12];

Sử dụng 16 tone để làm start marker : Dấu hiệu cho phân thu biết tín hiệu sẽ được đưa đến:

tone_index của start marker = $2*i + (i \% 2)$:

→ Danh sách tone_index: [0, 3, 4, 7, 8, 11, 12, 15, 16, 19, 20, 23, 24, 27, 28, 31]

Chuyển thành tần số: $f = (\text{freqStart} + \text{tone_index}) \times \text{hzPerSample}$

Danh sách tần số start marker:

[1125.000, 1265.625, 1312.500, 1453.125,
1500.000, 1640.625, 1687.500, 1828.125,
1875.000, 2015.625, 2062.500, 2203.125,
2250.000, 2390.625, 2437.500, 2578.125]

Tương tự ta cũng sẽ sử dụng 16 tone end marker còn lại để đánh dấu kết thúc dữ liệu

Tone_index của end marker: $2*i + (1 - i\%2)$

Danh sách tone_index: endMarker = [1, 2, 5, 6, 9, 10, 13, 14, 17, 18, 21, 22, 25, 26, 29, 30];

→ Danh sách tần số end marker :

[1171.875, 1218.750, 1359.375, 1406.250, 1546.875, 1593.750, 1734.375, 1781.250, 1921.875, 1968.750, 2109.375, 2156.250, 2296.875, 2343.750, 2484.375, 2531.250]

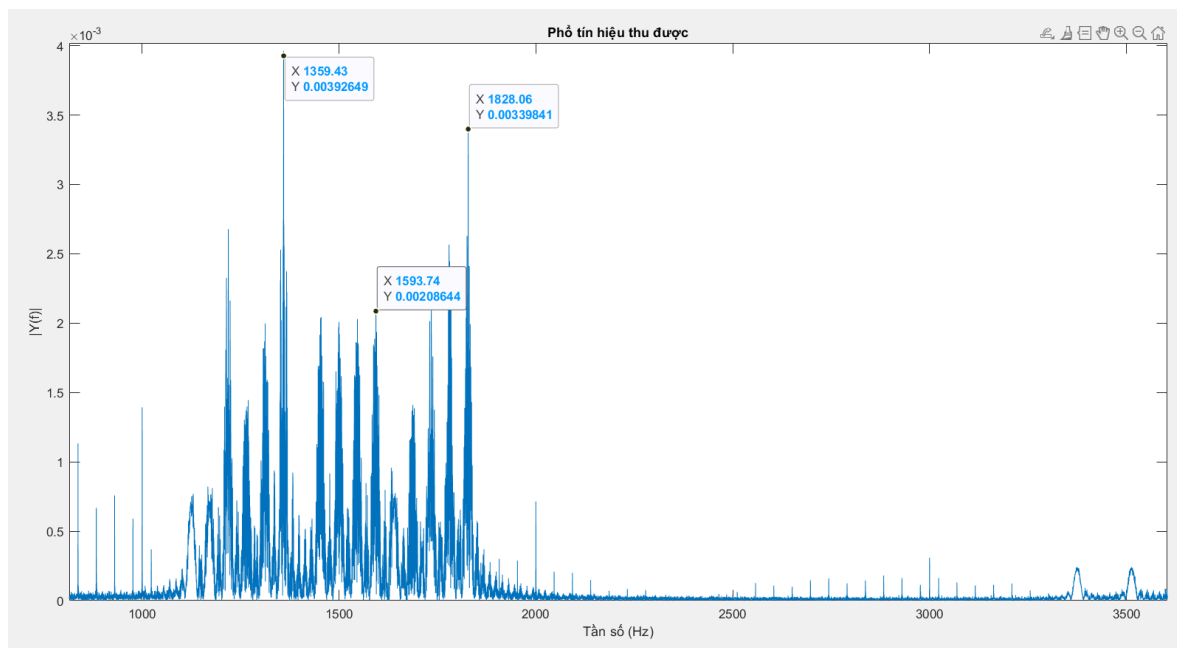
Thời điểm (ms)	Tone Index	Tần số (Hz)	Vai trò
0	0	1125.000	Start marker

64	3	1265.625	Start marker
128	4	1312.500	Start marker
...
1024	5	1359.375	Data (0x05 low)
1088	0	1125.000	Data (0x05 high)
1152	14	1781.250	Data (0xDE low)
...
1920	12	1687.500	Data (ECC)
...	End marker

Bảng 4.3 Kết quả của chuỗi âm thanh truyền “hello”

→ Phát liên tục trong ~3.5 giây

Thời điểm ms kia là phần tone_duration_ms = 64 ms được thiết lập



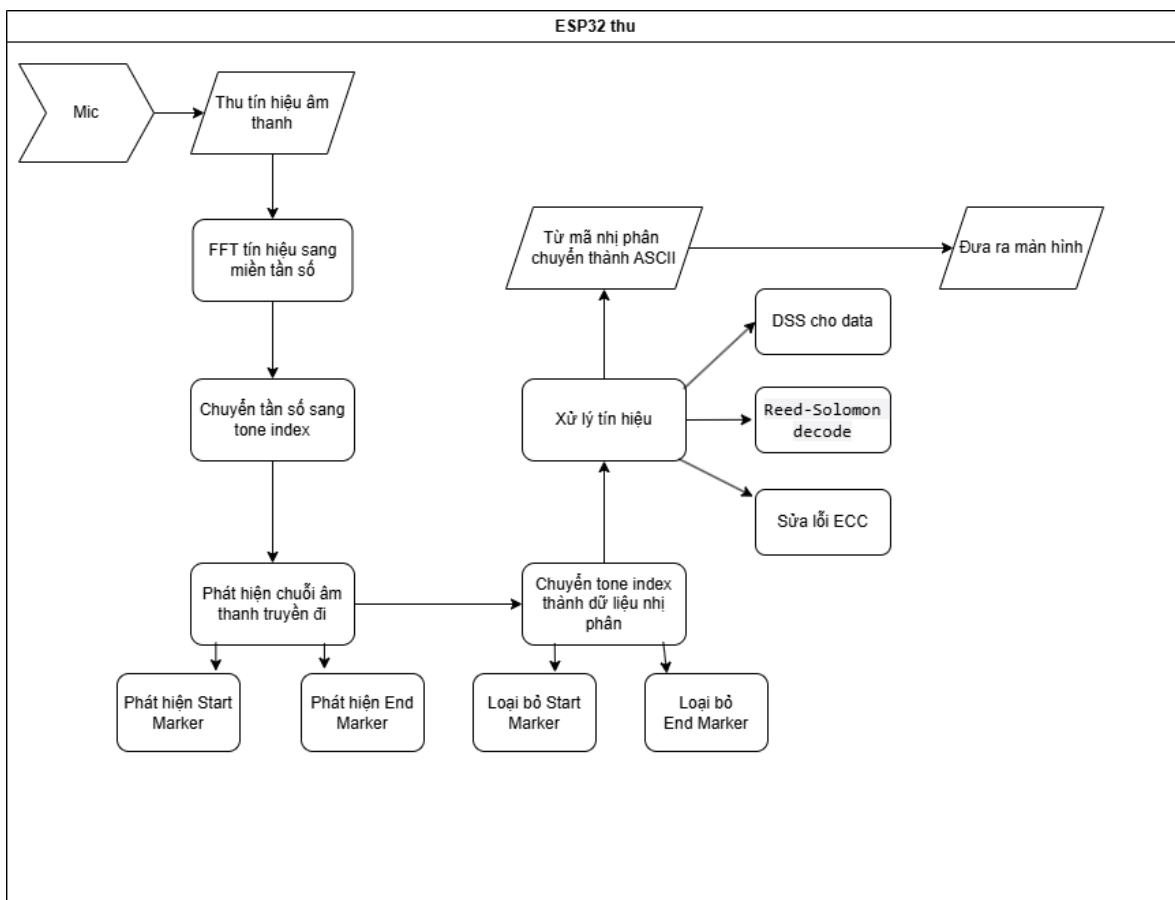
Hình 4.5 Kết quả tín hiệu ở đầu phát sau khi thu bằng MATLAB thực hiện FFT

Source Code phân phát: <https://github.com/QBL-23/fsk-transmitter.git>



Hình 4.6 QR Source code phần phát

4.2.2. Đầu thu



Quy trình hoạt động của phần thu:

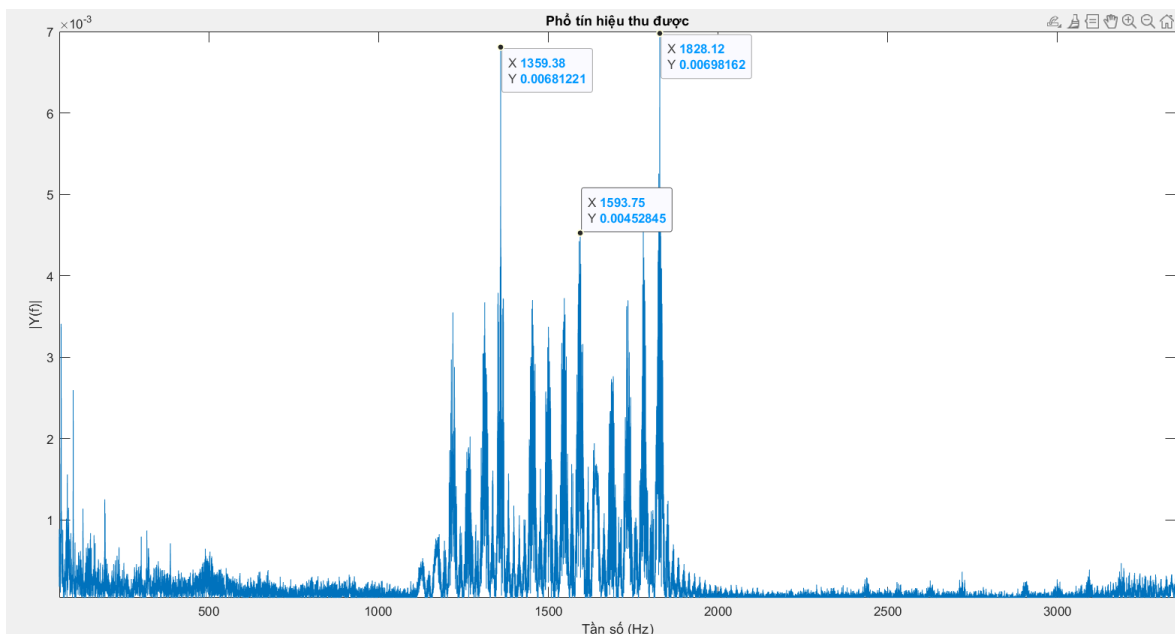
1. Thu âm thanh
2. Chuyển dữ liệu âm thanh thành 16bit
3. FFT tín hiệu sang miền tần số

Nhóm 16 - 154166

4. Chuyển tần số thành tone index
5. Tính công suất tại tần số xác định
6. Phát hiện chuỗi âm thanh truyền đi và các marker
7. Bỏ marker và chuyển dữ liệu sang nhị phân
8. Xử lý, sửa lỗi tín hiệu
9. Từ mã nhị phân chuyển sang ASCII và đưa ra màn hình

4.2.2.1. Thu tín hiệu âm thanh - FFT và phổ công suất

Mục tiêu: Chuyển tín hiệu âm thanh từ miền thời gian sang miền tần số để phát hiện các tông âm (tone) đã mã hóa dữ liệu.



Hình 4.7 Kết quả tín hiệu ở đầu thu sau khi thu bằng MATLAB thực hiện FFT

Các bước:

a. Nhận tín hiệu từ micro - Miền thời gian

Tín hiệu được thu dưới dạng mảng mẫu âm thanh chuẩn hóa trong khoảng $-1.0, 1.0-1.0, 1.0-1.0, 1.0$.

Số lượng mẫu: `samplesPerFrame = 512`, tốc độ lấy mẫu: `sampleRate = 24000 Hz`.

Mô phỏng cho chuỗi “Hello”

Dữ liệu nhận được (`m_rx.amplitude`) khi truyền chuỗi “Hello”:

`m_rx.amplitude = [0.234, 0.456, 0.123, ..., -0.089] // 512 phần tử`

b. Biến đổi Fourier nhanh (FFT)

Mục đích: Phân tích tín hiệu thành các thành phần tần số.

Thực hiện FFT để chuyển amplitude[512] thành fftOut[1024] dạng số phức.

Dữ liệu: $\text{fftOut}[2*i] = \text{phần thực}$, $\text{fftOut}[2*i+1] = \text{phần ảo}$.

Ví dụ:

$\text{fftOut}[80] = 45.67 \text{ // real (bin 40)}$

$\text{fftOut}[81] = 23.45 \text{ // imag}$

$\text{fftOut}[82] = 67.89 \text{ // real (bin 41)}$

$\text{fftOut}[83] = 34.56 \text{ // imag}$

c. Tính phổ công suất (Power Spectrum)

Công thức:

$\text{power}[i] = \text{real}^2 + \text{imag}^2$

Ví dụ:

$\text{spectrum}[40] = 45.67^2 + 23.45^2 = 2635.45$

$\text{spectrum}[41] = 67.89^2 + 34.56^2 = 5803.44$

d. Gập phổ (Spectrum Folding)

Mục đích: Kết hợp phổ đối xứng trái/phải để tăng SNR.

Công thức:

$\text{spectrum}[i] += \text{spectrum}[512 - i]$

Ví dụ:

$\text{spectrum}[40] = 2635.45 + 1234.56 = 3870.01$

$\text{spectrum}[41] = 5803.44 + 2345.67 = 8149.11$

4.2.2.2. Chuẩn hóa phổ (Quantization)

Mục tiêu: Nén phổ công suất về dải 8-bit (0–255) để xử lý hiệu quả hơn.

a. Tính hệ số chuẩn hóa:

$\text{amax} = \max(\text{spectrum}[i]) \text{ từ } i=40 \rightarrow 255$

$\text{scale} = 255.0 / \text{amax}$

Nhóm 16 - 154166

Ví dụ:

$\text{amax} = 8149.11 \rightarrow \text{scale} = 0.0313$

b. Áp dụng chuẩn hóa:

$\text{spectrumFixed}[i] = \text{round}(\text{spectrum}[i] * \text{scale})$

Kết quả:

$\text{spectrumFixed}[40] = \text{round}(3870.01 * 0.0313) = 121$

$\text{spectrumFixed}[41] = \text{round}(8149.11 * 0.0313) = 255$

4.2.2.3. Phát hiện tông âm (Tone Detection)

Mục tiêu: Phát hiện các tông tương ứng với từng nửa byte (nibble) của dữ liệu.

a. Phân tích phổ theo từng khung truyền (frame):

Mỗi byte mã hóa thành 2 nibble \rightarrow mỗi nibble được gán 1 tần số (tone).

Với $\text{binStart} = 40$, $\text{binDelta} = 16$, ta quét từ bin 40 trở đi để tìm đỉnh phổ lớn nhất trong mỗi khung.

Ví dụ - Byte 'H' (0x48):

Nibble thấp = 0x8 \rightarrow peak tại bin 48

Nibble cao = 0x4 \rightarrow peak tại bin 60

b. Ghi nhận số lần phát hiện:

$\text{detectedTones}[(2*j + k)*16 + \text{value}]++;$

$\text{detectedTones}[0*2*16 + 8]++;$ // lower nibble

$\text{detectedTones}[1*2*16 + 4]++;$ // upper nibble

4.2.2.4. Kết hợp nibble \rightarrow byte

Mục tiêu: Ghép các giá trị nibble thành byte dữ liệu.

Công thức:

$\text{byte} = (\text{upper_nibble} \ll 4) | \text{lower_nibble}$
 $\text{upper} = 4, \text{lower} = 8 \rightarrow \text{byte}$
 $= (4 \ll 4) + 8 = 0x48 = 'H'$

4.2.2.5. Giải mã Reed-Solomon

Mục tiêu: Sửa lỗi nhờ ECC (Error Correction Code).

Input: 22 byte gồm 16 dữ liệu + 6 ECC

Nếu Decode() thành công → nhận lại dữ liệu sạch

4.2.2.6. Khử trải phổ DSS

Mục tiêu: Khôi phục dữ liệu gốc sau khi đã XOR DSS lúc phát.

Công thức:

```
data[i] ^= DSSMagic[i];
```

4.2.2.7. Xuất kết quả cuối cùng

```
result = ['H', 'e', 'l', 'l', 'o', 0x00, ..., 0x00] // 16 byte
```

Hiển thị trên Serial và màn hình LCD:

LCD: “Receive: Hello”

Source phần thu: <https://github.com/QBL-23/fsk-receiver.git>



Hình 4.8 QR Source code phần thu

CHƯƠNG 5. KIỂM THỬ

Test case 1: Kiểm tra độ ổn định truyền dữ liệu ở các khoảng cách khác nhau:

User case: Độ ổn định khi truyền âm qua loa, mic. Tiêu chuẩn đánh giá: Tỷ lệ nhận đúng thông tin ở từng khoảng cách. Thủ tục kiểm thử: 1. Bấm nút gửi chuỗi “HELLO” từ phần phát sang phần thu trong môi trường yên tĩnh. 2. Đặt khoảng cách giữa phần phát và thu là các khoảng cách khác nhau. 3. Gửi liên tiếp 10 lần mỗi khoảng cách, ghi lại số lần nhận đúng.		
Khoảng cách (Cm)	Số lần nhận (Lần)	Tỷ lệ đúng (%)
3	10	100
5	10	100
10	10	100
15	9	90
20	9	90
25	9	90
30	8	80
40	7	70
50	6	60
70	5	50
100	1	10

Nhận xét: Kết quả kiểm thử cho thấy hệ thống truyền dữ liệu bằng âm thanh hoạt động ổn định ở khoảng cách gần và bắt đầu suy giảm hiệu suất rõ rệt khi khoảng cách tăng.

- Tốc độ bằng 0 khi ở mức 1/255 và bắt đầu chạy từ 2/255 tức là motor cần một mức PWM đủ lớn để bắt đầu chạy (ở đây là 2/255)
- Khi tăng duty cycle lên mức trung bình (2/255 đến 80/255), tốc độ cải thiện đáng kể (thời gian giảm rõ rệt từ 6s xuống 3,52s).
- Từ 100/255 trở lên, tốc độ dường như không thay đổi đáng kể, thời gian duy trì xấp xỉ 2,97s - 2,98s, điều này có thể do giới hạn của hệ thống (motor đạt mức bão hòa).

Kết luận: Như vậy ở duty cycle thấp, tốc độ thay đổi tương đối tuyến tính với duty cycle (giảm đều thời gian). Tuy nhiên, từ duty cycle 100/255 trở lên, tốc độ không còn tuyến tính (thời gian gần như cố định). Điều này cho thấy motor không thể đáp ứng thêm tốc độ dù duty cycle tăng.

Test case 2: Kiểm tra ảnh hưởng của môi trường lên nhiễu âm:

User case: Khả năng nhận đúng dữ liệu trong môi trường nhiều tiếng ồn.
 Tiêu chuẩn đánh giá: Tỷ lệ lỗi khi truyền trong các môi trường khác nhau.
 Thủ tục kiểm thử:

1. **Bật phát gửi chuỗi “HELLO” trong các môi trường khác nhau.**
2. Đặt khoảng cách cố định 10cm.
3. Gửi liên tiếp 10 lần, ghi lại số lần nhận đúng.

Môi trường	Số lần nhận (Lần)	Tỷ lệ đúng (%)
Phòng yên tĩnh	10	10
Có tiếng quạt nhẹ	9	9
Có nhạc nền nhỏ	9	9
Nhạc lớn	2	2

Nhận xét: Kết quả thử nghiệm cho thấy khả năng thu nhận tín hiệu của hệ thống bị ảnh hưởng rõ rệt bởi điều kiện âm thanh của môi trường xung quanh:

- Ở khoảng cách từ **3 cm đến 10 cm**, hệ thống thu nhận tín hiệu với **độ chính xác tuyệt đối (100%)**, cho thấy khả năng truyền và giải mã dữ liệu hoạt động hiệu quả trong điều kiện lý tưởng.
- Từ **15 cm đến 30 cm**, độ chính xác giảm nhẹ xuống còn **90–80%**, nhưng vẫn ở mức chấp nhận được, phù hợp với ứng dụng truyền dữ liệu cự ly ngắn.
- Khi tăng khoảng cách lên **40–70 cm**, độ chính xác tiếp tục giảm còn **70% đến 50%**, cho thấy tín hiệu âm thanh bắt đầu bị suy hao đáng kể do ảnh hưởng của khoảng cách và môi trường truyền.
- Ở khoảng cách **100 cm**, hệ thống chỉ nhận đúng **1/10 lần (10%)**, cho thấy mức độ nhiễu và suy giảm tín hiệu âm thanh vượt ngưỡng mà hệ thống có thể giải mã chính xác.

Kết luận:

- Hệ thống truyền dữ liệu bằng âm thanh sử dụng loa và microphone đạt độ ổn định cao ở cự ly ngắn (≤ 10 cm).
- Hiệu suất giảm dần theo khoảng cách, đặc biệt khi vượt quá 30 cm.
- Hệ thống có thể sử dụng hiệu quả trong các ứng dụng truyền dữ liệu tầm gần trong môi trường yên tĩnh.

Test case 3: Kiểm tra tốc độ và độ ổn định khi truyền chuỗi dài:

User case: Khả năng truyền chính xác chuỗi dữ liệu dài
 Tiêu chuẩn đánh giá: Tỷ lệ nhận đủ nội dung (không sai ký tự hoặc thiếu)
 Thủ tục kiểm thử:

1. **Bấm nút Gửi các chuỗi ký tự: “A”, “HUST”, “HELLO”, “DIENTUVIENTHONG”, “TESTTESTTESTTEST TESTTEST” trong môi trường yên tĩnh.**
2. Đặt khoảng cách giữa phần phát và thu là 10 cm.

3. Gửi liên tiếp 10 lần mỗi khoảng cách, ghi lại dữ liệu nhận được.		
Dữ liệu	Dữ liệu nhận được	Tỷ lệ đúng (%)
A	10	100
HUST	10	100
HELLO	10	100
DIENTUVIENTHONG	10	100
TESTTESTTESTTEST TESTTEST	9	90

Nhận xét: Kết quả kiểm thử cho thấy hệ thống có thể truyền chính xác dữ liệu văn bản với độ dài khác nhau, trong điều kiện môi trường yên tĩnh và khoảng cách cố định 10 cm:

- Với các chuỗi ngắn như "A", "HUST" và "HELLO", hệ thống đạt độ chính xác tuyệt đối (100%), cho thấy việc mã hóa, truyền và giải mã dữ liệu hoạt động ổn định với độ trễ và sai số gần như bằng 0.
- Đối với chuỗi dài hơn như "DIENTUVIENTHONG" (16 ký tự), hệ thống vẫn giữ tỷ lệ chính xác 100%, khẳng định hiệu suất truyền tải không bị ảnh hưởng đáng kể bởi độ dài chuỗi trong trường hợp độ dài vừa phải.
- Tuy nhiên, khi thử nghiệm với chuỗi rất dài "TESTTESTTESTTEST TESTTEST" (26 ký tự và có dấu cách), tỷ lệ nhận đúng giảm còn 90%. Điều này cho thấy với chuỗi dữ liệu dài:
- Nguy cơ mất dữ liệu hoặc lỗi giải mã tăng lên do thời gian truyền lâu hơn.
- Hệ thống có thể bị nhiễu ngắn hạn trong quá trình phát hoặc thu làm mất 1 phần thông tin giữa chuỗi.

Kết luận:

- Hệ thống truyền dữ liệu âm thanh hoạt động ổn định và chính xác với chuỗi ký tự ngắn và trung bình (từ 1–20 ký tự).
- Với chuỗi dài >25 ký tự, hiệu suất truyền giảm nhẹ, nguyên nhân có thể do:
 - o Giới hạn của thời gian truyền dẫn.
 - o Tăng xác suất nhiễu và lỗi đồng bộ.
 - o Hiện tượng "trôi tone" hoặc mất đồng bộ trong GGWave khi chuỗi kéo dài quá mức.
- Hệ thống vẫn có thể sử dụng để truyền dữ liệu dài nếu:
 - o Chia nhỏ gói tin.
 - o Tăng cường tính năng phát lại hoặc kiểm tra CRC.
 - o Giảm tốc độ truyền (tăng khoảng cách giữa các tone).

Test case 4: Kiểm tra khả năng chống nhiễu tần số:

User case: Mức độ nhầm lẫn khi có âm thanh FSK khác gần giống (chồng tần số).

Tiêu chuẩn đánh giá: Nhận sai thông tin khi có tín hiệu khác gần giống cùng lúc.
Thủ tục kiểm thử:

1. Bấm nút gửi các ký tự “A”, “HELLO”, “HUST”, “TEST” trong hai trường hợp”: Không có thiết bị gây nhiễu, Có 1 thiết bị gây nhiễu gần tần số phát, Có 1 thiết bị gây nhiễu đúng tần số phát, Có 2 thiết bị gây nhiễu gần tần số phát.

2. Đặt khoảng cách giữa phần phát và thu là 10 cm.

3. Gửi 10 lần mỗi khoảng cách, ghi lại số lần nhận đúng

Dữ liệu	Số lần nhận đúng			
	Không thiết bị nhiễu	1 thiết bị nhiễu gần tần số	1 thiết bị nhiễu đúng tần số	2 thiết bị nhiễu gần tần số
A	10	5	2	3
HELLO	10	4	2	5
HUST	10	4	3	4
TEST	10	4	3	4

Nhận xét: Kết quả kiểm thử thể hiện rõ ràng rằng hệ thống truyền dữ liệu bằng âm thanh rất nhạy cảm với nhiễu tín hiệu có tần số gần hoặc trùng tần số truyền (FSK).

- Trong trường hợp không có thiết bị gây nhiễu, hệ thống đạt hiệu suất tuyệt đối (100%) cho tất cả các chuỗi thử nghiệm (“A”, “HELLO”, “HUST”, “TEST”), cho thấy khả năng truyền – thu trong môi trường lý tưởng là rất ổn định.
- Khi xuất hiện 1 thiết bị gây nhiễu có tần số gần giống, số lần nhận đúng giảm mạnh còn 4–5 lần/10, tức là chỉ còn 40–50%, cho thấy nhiễu gần phổ đã bắt đầu ảnh hưởng tới quá trình giải mã của GGWave.
- Trong trường hợp thiết bị gây nhiễu đúng tần số phát, hệ thống gần như mất hoàn toàn khả năng nhận diện chính xác, với tỷ lệ đúng chỉ còn 20–30% (2–3 lần nhận đúng). Điều này cho thấy sự chồng lấn hoàn toàn gây mất đồng bộ hóa quá trình nhận, dẫn tới giải mã sai.
- Khi có 2 thiết bị gây nhiễu gần tần số, tỷ lệ đúng có sự cải thiện nhẹ so với trường hợp nhiễu trùng tần số (do không trùng hoàn toàn), nhưng vẫn rất thấp (30–50%) → cho thấy mức độ nhiễu phức hợp làm hệ thống khó phân biệt các tín hiệu FSK mục tiêu.

Kết luận:

- FSK dựa trên âm thanh là kỹ thuật nhạy cảm với nhiễu phổ, đặc biệt khi có tín hiệu khác có tần số nằm gần hoặc trùng với các tone đang sử dụng để truyền dữ liệu.
- Hệ thống hoạt động ổn định khi không có nhiễu hoặc chỉ có nhiễu xa phổ.

- Trong môi trường thực tế có nhiều nguồn âm thanh có tần số gần nhau (ví dụ: tiếng máy móc, nhạc điện tử, thiết bị âm thanh khác), hệ thống có thể gặp sai sót lớn hoặc không hoạt động đúng.

Test case 5: Kiểm tra ảnh hưởng của công suất loa đến khả năng truyền dữ liệu:

User case: Đánh giá hiệu quả truyền dữ liệu âm thanh theo từng mức công suất loa

Tiêu chuẩn đánh giá: Tỷ lệ nhận đúng dữ liệu khi phát bằng loa có công suất khác nhau

Thủ tục kiểm thử: Thay đổi điện áp qua loa:

1. Bấm nút gửi chuỗi “HELLO” từ phần phát sang phần thu trong môi trường yên tĩnh.
2. Đặt khoảng cách giữa phần phát và thu là 10 cm.
3. Thay đổi các giá trị điện áp qua loa dùng biến trở.
4. Gửi 10 lần, ghi lại số lần đúng.

Công suất loa	Số lần nhận (Lần)	Tỷ lệ đúng (%)
0.1 W	1	10
0.2 W	4	40
0.3 W	7	70
0.4 W	10	100
0.5 W	10	100

Nhận xét: Kết quả kiểm thử cho thấy công suất phát của loa là yếu tố then chốt ảnh hưởng đến khả năng truyền dữ liệu bằng âm thanh ở khoảng cách xa:

- Với công suất cực thấp (0.1 W), hệ thống chỉ nhận đúng 1/10 lần (10%), cho thấy âm lượng phát ra quá nhỏ để được thu đủ rõ bởi microphone ở khoảng cách 10 cm.
- Khi tăng công suất lên 0.2 W, số lần nhận đúng tăng lên 4/10 (40%), tức vẫn còn bị suy hao hoặc lẫn nhiễu nền.
- Tại 0.3 W, hệ thống bắt đầu thu tín hiệu ổn định hơn với 70% độ chính xác, nghĩa là mức công suất này đã tiệm cận ngưỡng đủ để truyền đáng tin cậy ở khoảng cách 10 cm.
- Với 0.4 W và 0.5 W, hệ thống đạt 100% tỷ lệ nhận đúng, cho thấy mức âm lượng đã đủ mạnh để đảm bảo độ rõ và tách biệt tần số cho thuật toán giải mã FSK hoạt động hiệu quả.

Kết luận:

- Có mối tương quan thuận giữa công suất loa và độ chính xác của hệ thống truyền dữ liệu: công suất càng lớn → tỷ lệ nhận đúng càng cao.

- Hệ thống chỉ hoạt động ổn định ở khoảng cách ≥ 10 cm khi công suất loa từ 0.4 W trở lên, cho thấy mức âm lượng cần đủ lớn để vượt qua nhiễu môi trường và tổn hao không khí.
- Việc điều chỉnh công suất phát phù hợp với khoảng cách sử dụng là rất quan trọng. Dư thừa công suất có thể gây nhiễu chéo, trong khi thiếu công suất dẫn đến mất tín hiệu.