

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**



BÁO CÁO TỔNG KẾT

**ĐIỀU KHIỂN XE BẰNG MPU6050 THÔNG QUA
LORA**

Sinh viên thực hiện Ngô Huỳnh Quốc Huy Mssv: 6251020057

Võ Văn Tuấn Mssv: 6251020094

Hà Nhật Chương Mssv: 6251020037

Lớp: Điện tử tin học công nghiệp K62

Người hướng dẫn: TS. Lê Tiến Lộc

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**



BÁO CÁO TỔNG KẾT

**ĐIỀU KHIỂN XE BẰNG MPU6050 THÔNG QUA
LORA**

Sinh viên thực hiện	Ngô Huỳnh Quốc Huy	Mssv: 6251020057
	Võ Văn Tuấn	Mssv: 6251020094
	Hà Nhật Churong	Mssv: 6251020037

Lớp: Điện tử tin học công nghiệp K62

Người hướng dẫn: TS. Lê Tiến Lộc

MỤC LỤC

MỤC LỤC.....	3
DANH MỤC HÌNH ẢNH.....	5
DANH MỤC BẢNG.....	6
KÍ TỰ VIẾT TẮT	7
CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI	8
1.1. Lý do chọn đề tài.....	8
1.2. Mục tiêu đề tài.....	9
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT	10
2.1. Khái niệm về hệ thống nhúng	10
2.2. Tín hiệu cảm biến và xử lý tín hiệu	11
2.3. Tổng quan về hệ thống nhúng trong đo lường	12
CHƯƠNG 3: GIỚI THIỆU LINH KIỆN SỬ DỤNG	14
3.1. Vi điều khiển STM32F103C8T6	14
3.1.1. Giới thiệu chung.....	14
3.1.2. Thông số kỹ thuật chính	14
3.1.3. Vai trò trong hệ thống.....	15
3.2. Cảm biến MPU6050	15
3.2.1. Giới thiệu chung.....	15
3.2.2. Thông số kỹ thuật chính	16
3.2.3. Vai trò trong hệ thống.....	16
3.3. Module truyền thông LoRa.....	17
3.3.1. Giới thiệu chung.....	17
3.3.2. Thông số kỹ thuật chính	18
3.3.3. Vai trò trong hệ thống.....	18
CHƯƠNG 4: THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN.....	19
4.1, kiến trúc tổng thể của hệ thống	19
4.2, Phần cứng hệ thống phát.....	20

4.2.1. Sơ đồ mạch điện	20
4.3. Phần cứng hệ thống thu.....	22
4.3.1. Sơ đồ mạch điện	22
4.4. Phần mềm hệ thống phát.....	24
4.4.1. Driver hoạt động MPU6050	24
4.4.2. Driver SX1278.....	29
4.4.3. Chương trình hệ thống đọc dữ liệu MPU6050 và truyền dữ liệu.....	37
4.5. Phần mềm hệ thống thu.....	45
CHƯƠNG 5: KẾT QUẢ THỰC NGHIỆM.....	51
5.1: Mô hình phần cứng và thực tế.....	51
5.1.1: Phần cứng hệ thống phát	51
5.1.1: Phần cứng hệ thống thu	52
CHƯƠNG 6: KẾT LUẬN VÀ BÀI HỌC	54
6.1. kết luận.....	54
6.2. Bài học.....	54
TÀI LIỆU THAM KHẢO	56
PHỤ LỤC.....	57

DANH MỤC HÌNH ẢNH

<i>Hình 2.1: Hệ thống nhúng</i>	11
<i>Hình 2.2: Mô hình cảm biến</i>	12
<i>Hình 3.1: tổng quan stm32f103c8</i>	14
<i>Hình 3.2: tổng quan mpu6050</i>	16
<i>Hình 3.3: tổng quan lora</i>	18
<i>Hình 4.1: sơ đồ mạch phát</i>	20
<i>Hình 4.2: sơ đồ mạch thu</i>	22
<i>Hình 4.3: lưu đồ thuật toán</i>	32
<i>Hình 4.4: lưu đồ thuật toán</i>	37
<i>Hình 4.5: thiết lập cấu hình</i>	38
<i>Hình 4.6: thiết lập cấu hình</i>	39
<i>Hình 4.7: thiết lập cấu hình</i>	39
<i>Hình 4.8: thiết lập cấu hình</i>	40
<i>Hình 4.9: thiết lập cấu hình</i>	41
<i>Hình 4.10: thiết lập cấu hình</i>	43
<i>Hình 4.11: thiết lập cấu hình</i>	44
<i>Hình 4.12: thiết lập cấu hình</i>	45
<i>Hình 4.13: thiết lập cấu hình</i>	46
<i>Hình 4.14: thiết lập cấu hình</i>	46
<i>Hình 4.15: thiết lập cấu hình</i>	47
<i>Hình 5.1: phần cứng phát</i>	51
<i>Hình 5.2: phần cứng thu</i>	52

DANH MỤC BẢNG

<i>Bảng 4.1. lưu đồ thuật toán</i>	19
<i>Bảng 4.2. thông số lora</i>	33
<i>Bảng 4.3. thông số lora</i>	43
<i>Bảng 4.4. thiết lập mpu6050.....</i>	44
<i>Bảng 4.5. thiết lập Led.....</i>	50
<i>Bảng 4.6. thiết lập tốc độ L298</i>	50
<i>Bảng 4.7. thiết lập động cơ L298</i>	50

KÍ TỰ VIẾT TẮT

Viết tắt	Điễn giải
ADC	Analog to Digital Converter (Bộ chuyển đổi tương tự – số)
DOF	Degree of Freedom (Bậc tự do)
DMP	Digital Motion Processor (Bộ xử lý chuyển động số)
HAL	Hardware Abstraction Layer (Lớp trừu tượng phần cứng)
I2C	Inter-Integrated Circuit (Chuẩn giao tiếp nối tiếp)
LoRa	Long Range (Công nghệ truyền thông tầm xa)
MPU6050	Tên cảm biến tích hợp gia tốc kế & con quay hồi chuyển
NSS	Network Select Slave (Chân chọn chip trong giao tiếp SPI)
PWM	Pulse Width Modulation (Điều chế độ rộng xung)
SPI	Serial Peripheral Interface (Giao tiếp ngoại vi nối tiếp)
STM32	Dòng vi điều khiển 32-bit của STMicroelectronics
UART	Universal Asynchronous Receiver Transmitter (Truyền nhận không đồng bộ phô thông)

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1. Lý do chọn đề tài.

Nhu cầu thị trường:

Hiện nay, các hệ thống điều khiển phương tiện, robot hoặc thiết bị từ xa phần lớn vẫn sử dụng tay điều khiển (remote) hoặc điện thoại thông minh kết nối qua Bluetooth hoặc WiFi. Tuy nhiên, các công nghệ này có phạm vi hoạt động hạn chế và dễ bị ảnh hưởng bởi vật cản, tín hiệu nhiễu hoặc mất kết nối trong môi trường rộng lớn, đặc biệt ở các khu vực không có mạng internet ổn định. Điều này gây ra nhiều bất tiện cho người sử dụng cũng như hạn chế khả năng mở rộng ứng dụng trong thực tế.

Tiềm năng ứng dụng rộng rãi:

Việc điều khiển thiết bị từ xa bằng cách sử dụng cảm biến chuyển động (MPU6050) kết hợp với công nghệ truyền thông tầm xa LoRa mở ra hướng ứng dụng rộng rãi cho nhiều lĩnh vực như: xe điều khiển trong nông nghiệp, robot cứu hộ, xe tự hành, hoặc các thiết bị vận chuyển trong nhà kho, nhà máy. Giải pháp này cho phép người dùng điều khiển thiết bị một cách trực quan bằng các cử chỉ tay mà không cần dây dẫn hay mạng internet, đồng thời duy trì kết nối ổn định ở khoảng cách xa.

Ứng dụng công nghệ tiên tiến:

Đề tài ứng dụng cảm biến MPU6050 – một module tích hợp con quay hồi chuyển và gia tốc kế – để nhận biết chuyển động của tay người điều khiển. Dữ liệu từ cảm biến sẽ được xử lý và truyền không dây bằng công nghệ LoRa, vốn nổi bật với khả năng truyền xa (lên đến hàng km) và tiêu thụ năng lượng thấp. Hệ thống có thể hoạt động hiệu quả trong môi trường mở, độc lập với hạ tầng mạng hiện có, đồng thời dễ dàng triển khai thực tế.

Chi phí triển khai hợp lý:

Các module MPU6050 và LoRa hiện nay có giá thành thấp, phổ biến trên thị trường và dễ tiếp cận đối với sinh viên, kỹ sư hoặc cá nhân nghiên cứu độc lập. Điều này cho phép triển khai hệ thống điều khiển thông minh với chi phí thấp hơn nhiều so với các giải pháp thương mại, đồng thời dễ dàng mở rộng và tùy chỉnh theo nhu cầu sử dụng. Đề tài mang lại một lựa chọn tiết kiệm, linh hoạt và phù hợp với xu hướng phát triển thiết bị điều khiển không dây hiện đại.

1.2. Mục tiêu đề tài

Xây dựng hệ thống điều khiển xe từ xa bằng cử chỉ tay thông qua cảm biến MPU6050, truyền tín hiệu không dây bằng công nghệ LoRa với chi phí thấp, dễ triển khai và không phụ thuộc vào mạng internet.

Nhận diện chính xác chuyển động tay (nghiêng, xoay) để điều khiển các hướng di chuyển của xe (tiến, lùi, trái, phải), đảm bảo độ nhạy và độ trễ thấp trong quá trình điều khiển thực tế.

Tăng tính linh hoạt và tự động hóa, giúp người dùng điều khiển xe từ khoảng cách xa (lên đến vài trăm mét đến vài km trong điều kiện lý tưởng), phù hợp cho các ứng dụng trong nông nghiệp, môi trường đặc biệt hoặc robot địa hình.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

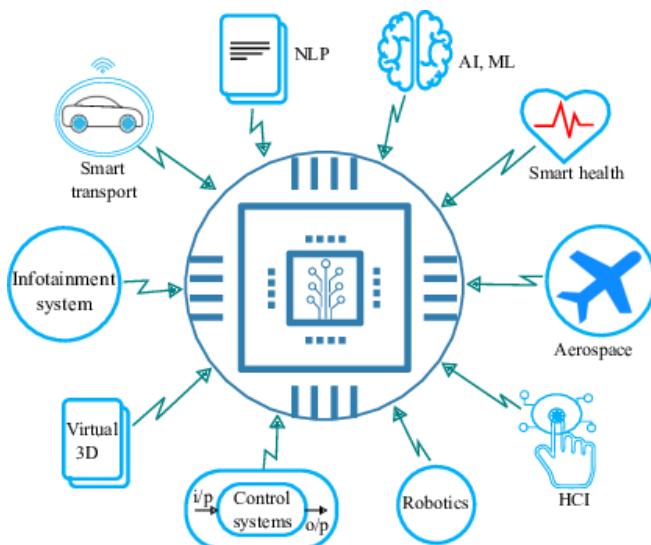
2.1. Khái niệm về hệ thống nhúng

Hệ thống nhúng (embedded system) là một thuật ngữ để chỉ một hệ thống có khả năng tự trị được nhúng vào trong một môi trường hay hệ thống mẹ. Đó là các hệ thống tích hợp cả phần cứng và phần mềm phục vụ các bài toán chuyên dụng trong nhiều lĩnh vực công nghiệp, tự động hóa điều khiển, quan trắc và truyền tin. Đặc điểm của các hệ thống nhúng là hoạt động ổn định và có tính năng tự động hóa cao.

Hệ thống nhúng thường được thiết kế để thực hiện một chức năng chuyên biệt nào đó. Khác với các máy tính đa chức năng, chẳng hạn như máy tính cá nhân, một hệ thống nhúng chỉ thực hiện một hoặc một vài chức năng nhất định, thường đi kèm với những yêu cầu cụ thể và bao gồm một số thiết bị máy móc và phần cứng chuyên dụng mà ta không tìm thấy trong một máy tính đa năng nói chung. Vì hệ thống chỉ được xây dựng cho một số nhiệm vụ nhất định nên các nhà thiết kế có thể tối ưu hóa nó nhằm giảm thiểu kích thước và chi phí sản xuất. Các hệ thống nhúng thường được sản xuất hàng loạt với số lượng lớn. Hệ thống nhúng rất đa dạng, phong phú về chủng loại. Đó có thể là những thiết bị cầm tay nhỏ gọn như đồng hồ kỹ thuật số và máy chơi nhạc MP3, hoặc những sản phẩm lớn như đèn giao thông, bộ kiểm soát trong nhà máy hoặc hệ thống kiểm soát các máy nồng lưỡng hạt nhân. Xét về độ phức tạp, hệ thống nhúng có thể rất đơn giản với một vi điều khiển hoặc rất phức tạp với nhiều đơn vị, các thiết bị ngoại vi và mạng lưới được nén gọn trong một lớp vỏ máy lớn.

Các thiết bị PDA hoặc máy tính cầm tay cũng có một số đặc điểm tương tự với hệ thống nhúng như các hệ điều hành hoặc vi xử lý điều khiển chúng nhưng các thiết bị này không phải là hệ thống nhúng thật sự bởi chúng là các thiết bị đa năng, cho phép sử dụng nhiều ứng dụng và kết nối đến nhiều thiết bị ngoại vi.

Bên cạnh đó, xu hướng hiện nay đang hướng đến việc phát triển các hệ thống nhúng thông minh, có khả năng kết nối Internet (IoT – Internet of Things) và xử lý dữ liệu ngay tại thiết bị mà không cần phụ thuộc vào máy chủ trung tâm. Việc tích hợp trí tuệ nhân tạo (AI) vào hệ thống nhúng cũng đang được nghiên cứu và ứng dụng mạnh mẽ, mở ra nhiều tiềm năng mới trong lĩnh vực tự động hóa và điều khiển thông minh.



Hình 2.1: Hệ thống nhúng

2.2. Tín hiệu cảm biến và xử lý tín hiệu

Trong một hệ thống đo lường, cảm biến là thành phần đầu vào có nhiệm vụ chuyển đổi các đại lượng vật lý cần đo (chẳng hạn như chuyển động, áp suất, nhiệt độ, ánh sáng, từ trường...) thành tín hiệu điện. Đây là bước quan trọng đầu tiên để biến thông tin thực tế thành dữ liệu có thể phân tích và xử lý trong môi trường số.

Tín hiệu do cảm biến tạo ra thường thuộc một trong hai dạng chính:

Tín hiệu tương tự (analog): Là tín hiệu biến thiên liên tục theo thời gian, biểu hiện qua điện áp hoặc dòng điện thay đổi theo giá trị đo được. Loại tín hiệu này yêu cầu sử dụng bộ chuyển đổi tương tự – số (ADC) để đưa vào xử lý trên hệ thống vi điều khiển.

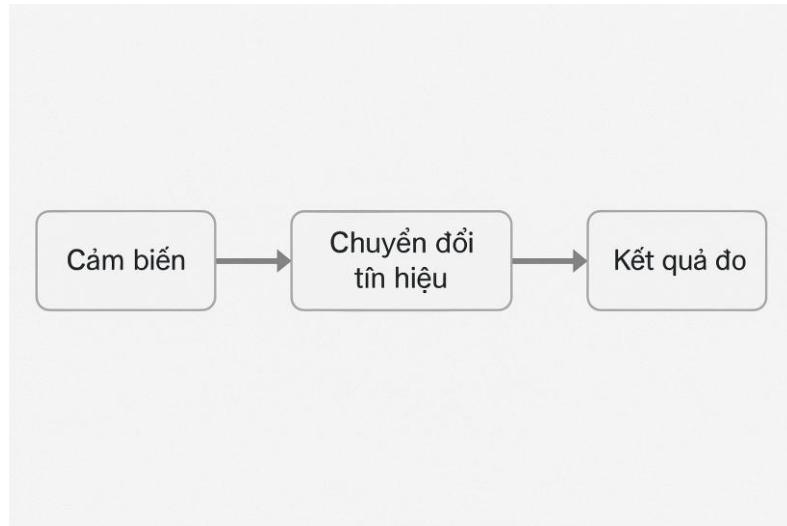
Tín hiệu số (digital): Là tín hiệu có dạng rời rạc, thường biểu diễn bằng hai mức logic (cao/thấp, 1/0). Loại tín hiệu này có thể được đọc trực tiếp bởi các chân vào số của vi điều khiển, và được xử lý dễ dàng hơn so với tín hiệu tương tự.

Việc xử lý tín hiệu cảm biến trong hệ thống đo lường thường bao gồm các bước:

Xử lý: Bao gồm khuếch đại tín hiệu yếu, lọc nhiễu điện, ổn định mức tín hiệu đầu vào.

Chuyển đổi tín hiệu: Nếu là tín hiệu analog thì cần chuyển đổi sang dạng số; nếu là tín hiệu số thì có thể đọc và xử lý trực tiếp.

Trích xuất thông tin: Là bước phân tích dữ liệu nhận được để rút ra giá trị đo có ý nghĩa, như tốc độ, vị trí...



Hình 2.2: Mô hình cảm biến

2.3. Tổng quan về hệ thống nhúng trong đo lường

Một hệ thống đo lường sử dụng nền tảng hệ nhúng thường hoạt động theo một chu trình khép kín và liên tục, đảm bảo thu thập và xử lý dữ liệu một cách hiệu quả và đáng tin cậy. Quy trình cơ bản có thể mô tả qua các bước sau:

Giai đoạn thu nhận dữ liệu đầu vào: Các cảm biến chuyên dụng sẽ ghi nhận các đại lượng vật lý như chuyển động, áp suất, ánh sáng, nhiệt độ, độ ẩm, tốc độ dòng chảy,... tùy theo yêu cầu của ứng dụng. Mỗi loại cảm biến thường cho ra tín hiệu điện áp, dòng điện hoặc xung số tương ứng với giá trị của đại lượng đo được.

Giai đoạn chuyển đổi và đưa vào hệ thống: Tín hiệu đầu ra của cảm biến sẽ được đưa vào hệ thống nhúng thông qua các chân analog hoặc digital trên vi điều khiển. Trường hợp tín hiệu là analog, hệ thống sẽ sử dụng bộ chuyển đổi ADC (Analog to Digital Converter) để biến đổi thành tín hiệu số phục vụ cho xử lý.

Xử lý tín hiệu và tính toán: Vi điều khiển trong hệ thống nhúng sẽ nhận dữ liệu và thực hiện các thuật toán xử lý: lọc nhiễu, tính trung bình, phân tích ngưỡng, hoặc các thuật toán điều khiển như PID. Đây là khâu quan trọng giúp đảm bảo kết quả đo chính xác, ổn định và phù hợp với yêu cầu thực tế.

Giai đoạn xuất dữ liệu: Sau khi xử lý, kết quả có thể được hiển thị trực tiếp trên màn hình LCD/OLED, hoặc được gửi đi thông qua giao tiếp UART, I2C, SPI hoặc Wi-Fi/Bluetooth. Ngoài ra, hệ thống cũng có thể ghi log vào bộ nhớ trong, thẻ SD hoặc gửi lên server/cloud/Firebase để lưu trữ và phân tích từ xa.

Tổng quát, mô hình hoạt động có thể biểu diễn như sau:

Cảm biến → Vị điều khiển → Xử lý tín hiệu → Hiển thị / Giao tiếp / Lưu trữ

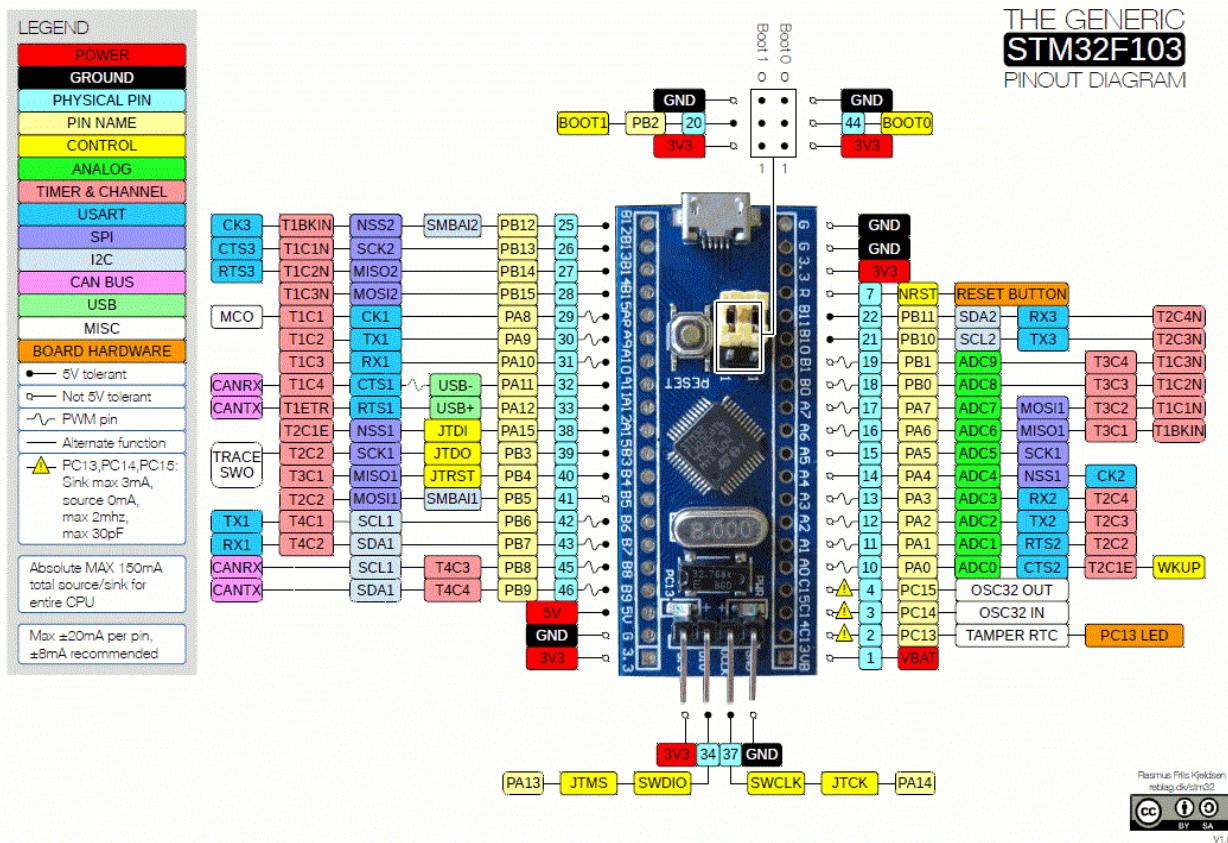
Trong các ứng dụng hiện đại, hệ thống nhúng còn có thể mở rộng thêm các chức năng như gửi cảnh báo khi vượt ngưỡng, kích hoạt relay điều khiển thiết bị, hoặc đồng bộ dữ liệu thời gian thực với hệ thống giám sát trung tâm. Điều này giúp hệ nhúng không chỉ là công cụ đo lường đơn thuần mà còn đóng vai trò như một phần tử quyết định trong hệ thống điều khiển tự động và thông minh hóa quy trình.

CHƯƠNG 3: GIỚI THIỆU LINH KIỆN SỬ DỤNG

3.1. Vi điều khiển STM32F103C8T6

3.1.1. Giới thiệu chung

STM32F103C8T6 là vi điều khiển thuộc dòng STM32 của STMicroelectronics, sử dụng lõi ARM Cortex-M3 với tốc độ xử lý lên đến 72 MHz. Đây là một trong những dòng vi điều khiển phổ biến nhờ khả năng xử lý mạnh, tiêu thụ năng lượng thấp, giá thành rẻ và được hỗ trợ bởi cộng đồng phát triển rộng lớn.



Hình 3.1: Tổng quan stm32f103c8

3.1.2. Thông số kỹ thuật chính

Lõi xử lý: ARM Cortex-M3, 32-bit, 72 MHz

Bộ nhớ: 64 KB Flash, 20 KB SRAM

Số chân I/O: 37 (đa chức năng)

Giao tiếp: UART, I2C, SPI, PWM, ADC,...

Điện áp hoạt động: 2.0V – 3.6V

Tích hợp đồng hồ RTC, bộ định thời (Timer), Watchdog,...

3.1.3. Vai trò trong hệ thống

STM32F103C8T6 là bộ vi điều khiển lõi ARM Cortex-M3 được thiết kế để đáp ứng các yêu cầu xử lý đa nhiệm trong các ứng dụng nhúng. Với tốc độ xử lý lên đến 72 MHz cùng bộ ngoại vi phong phú như UART, SPI, I2C, ADC, PWM,... STM32 cho phép kết nối với nhiều loại cảm biến và module truyền thông. Ngoài ra, nó hỗ trợ chế độ tiết kiệm năng lượng và khả năng xử lý ngắt linh hoạt, rất phù hợp cho các hệ thống yêu cầu phản hồi nhanh và ổn định.

Trong hệ thống điều khiển xe bằng cử chỉ, STM32F103C8 đóng vai trò là trung tâm xử lý chính:

Giao tiếp với cảm biến MPU6050 thông qua giao thức I2C để thu thập dữ liệu chuyển động (góc nghiêng, vận tốc quay).

Xử lý tín hiệu cảm biến, áp dụng các thuật toán lọc (như low-pass filter, Kalman filter) để loại bỏ nhiễu và ổn định dữ liệu đầu vào.

Phân tích dữ liệu chuyển động để xác định hành động điều khiển tương ứng (ví dụ: nghiêng tay về phía trước tương ứng với lệnh "tiến").

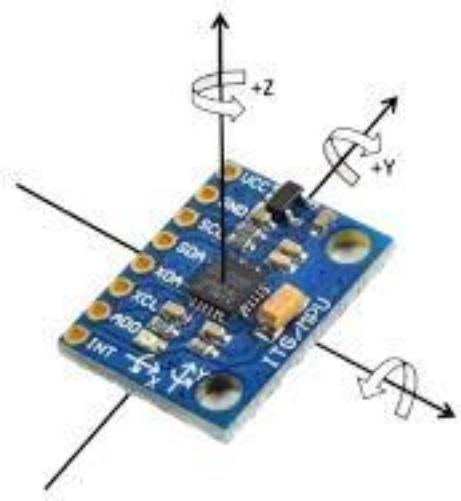
Truyền lệnh điều khiển sang module LoRa, sử dụng giao thức SPI, để gửi dữ liệu không dây đến phần nhận tín hiệu trên xe.

Nhờ khả năng xử lý nhanh, hoạt động ổn định và tài nguyên phần cứng đa dạng, STM32 là lựa chọn lý tưởng cho các hệ thống điều khiển thực thời có yêu cầu về độ tin cậy và độ trễ thấp.

3.2. Cảm biến MPU6050

3.2.1. Giới thiệu chung

MPU6050 là cảm biến tích hợp 3 trục gia tốc kế và 3 trục con quay hồi chuyển, cho phép đo được gia tốc tuyến tính và tốc độ quay quanh các trục XYZ. Module này rất phổ biến trong các ứng dụng nhận dạng chuyển động, cân bằng robot, điều khiển bằng cử chỉ...



Hình 3.2: tổng quan mpu6050

3.2.2. Thông số kỹ thuật chính

Gia tốc kế: $\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$

Con quay hồi chuyển: ± 250 , ± 500 , ± 1000 , $\pm 2000^{\circ}/s$

Giao tiếp: I2C (địa chỉ mặc định 0x68)

Tích hợp DMP (Digital Motion Processor)

Điện áp hoạt động: 3.3V – 5V

3.2.3. Vai trò trong hệ thống

MPU6050 là cảm biến tích hợp 6 bậc tự do (6 DOF), bao gồm:

Gia tốc kế 3 trục (Accelerometer): đo gia tốc tuyến tính dọc theo các trục X, Y, Z (đơn vị: m/s^2 hoặc g).

Con quay hồi chuyển 3 trục (Gyroscope): đo tốc độ quay quanh các trục X, Y, Z (đơn vị: độ/giây).

MPU6050 có thể kết hợp dữ liệu từ cả hai loại cảm biến để tính toán góc nghiêng (pitch, roll, yaw) của thiết bị. Với khả năng giao tiếp qua I2C, cảm biến dễ dàng kết nối với các vi điều khiển để truyền dữ liệu thời gian thực.

Trong đề tài này, MPU6050 được gắn vào tay người điều khiển. Dựa trên dữ liệu từ cảm biến:

Khi người dùng nghiêng tay theo hướng nhất định, hệ thống sẽ ghi nhận được sự thay đổi góc nghiêng hoặc vận tốc quay.

Dữ liệu này được phân tích để xác định hành vi điều khiển: nghiêng trái → rẽ trái, nghiêng phải → rẽ phải, nghiêng về trước → đi tới, nghiêng lùi → đi lui.

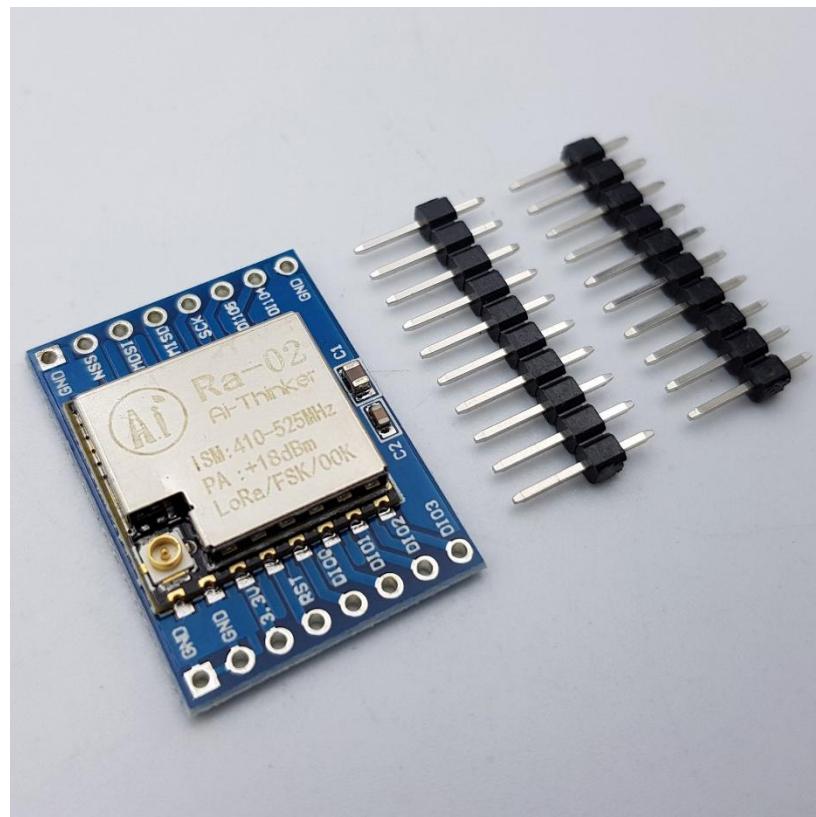
MPU6050 đóng vai trò như một bộ điều khiển chuyển động tự nhiên, thay thế cho các thiết bị điều khiển truyền thống như joystick hay nút bấm.

Tuy nhiên, do đặc điểm nhạy cảm với dao động nhỏ và nhiễu từ môi trường, cần phải có bộ lọc tín hiệu và hiệu chỉnh trong phần mềm để đảm bảo tính ổn định và chính xác của tín hiệu điều khiển.

3.3. Module truyền thông LoRa

3.3.1. Giới thiệu chung

LoRa là công nghệ truyền thông không dây tầm xa, tiêu thụ năng lượng thấp, sử dụng kỹ thuật điều chế LoRa (Long Range). Module Ra-02 hoặc SX1278 là một trong những phiên bản phổ biến, hoạt động ở tần số 433 MHz hoặc 868/915 MHz tùy theo khu vực.



Hình 3.3: tống quan lora

3.3.2. Thông số kỹ thuật chính

Tần số hoạt động: 433 MHz (hoặc 868/915 MHz)

Khoảng cách truyền: lên đến 2–5 km (môi trường mở)

Giao tiếp: SPI

Công suất phát: +20 dBm

Tốc độ truyền: 0.018 – 37.5 kbps

Điện áp: 1.8V – 3.7V (thường dùng 3.3V)

3.3.3. Vai trò trong hệ thống

LoRa (Long Range) là công nghệ truyền thông không dây sử dụng kỹ thuật điều chế dải tần rộng (spread spectrum modulation), giúp truyền tín hiệu ở khoảng cách xa với công suất thấp. LoRa hoạt động tốt trong môi trường có vật cản hoặc khu vực nông thôn – nơi tín hiệu WiFi hoặc Bluetooth thường bị hạn chế.

Các module như Ra-02 hoặc SX1278 sử dụng giao thức SPI để giao tiếp với vi điều khiển, có khả năng truyền dữ liệu với khoảng cách lên đến 2–5 km trong điều kiện lý tưởng.

Trong hệ thống điều khiển xe:

LoRa được sử dụng để truyền tín hiệu điều khiển từ bộ tay cầm (người điều khiển) đến xe. Tín hiệu này chứa thông tin đã được xử lý từ MPU6050, như hướng di chuyển và trạng thái hành động.

Nhờ khả năng truyền xa và chống nhiễu tốt, LoRa cho phép người dùng điều khiển xe ở khoảng cách lớn mà không cần đến mạng internet.

Hệ thống hoạt động ở tần số 433 MHz hoặc 868 MHz, với tốc độ truyền vừa phải nhưng ổn định và tiết kiệm năng lượng, phù hợp với các ứng dụng điều khiển đơn giản nhưng yêu cầu truyền xa và đáng tin cậy.

LoRa không thích hợp cho truyền dữ liệu lớn, nhưng rất phù hợp trong ứng dụng điều khiển như đề tài này, nơi chỉ cần gửi các gói dữ liệu nhỏ nhưng thường xuyên và ổn định.

CHƯƠNG 4: THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN

4.1, kiến trúc tổng thể của hệ thống



Bảng 4.1 lưu đồ thuật toán

Khởi động hệ thống

Hệ thống bắt đầu hoạt động khi được cấp nguồn. Vi điều khiển (STM32F103C8) tiến hành khởi tạo các thành phần phần cứng: cảm biến MPU6050, module LoRa, các chân I/O.

Kết nối LoRa

Hai module LoRa (phát và thu) được cấu hình để sẵn sàng truyền và nhận dữ liệu. Đây là bước thiết lập thông số truyền (tần số, địa chỉ, tốc độ baud...) giữa hai bên.

Xử lý tín hiệu MPU6050

Cảm biến MPU6050 gửi dữ liệu thô (góc nghiêng, gia tốc, tốc độ quay) đến vi điều khiển thông qua giao tiếp I2C. STM32 xử lý dữ liệu này (lọc nhiễu, phân tích) để xác định hướng điều khiển tương ứng (tiến, lùi, rẽ trái, rẽ phải...).

Gửi tín hiệu qua LoRa

STM32 mã hóa thông tin điều khiển (dưới dạng gói lệnh) và gửi qua module LoRa để truyền không dây đến mạch thu gắn trên xe.

Nhận dữ liệu LoRa

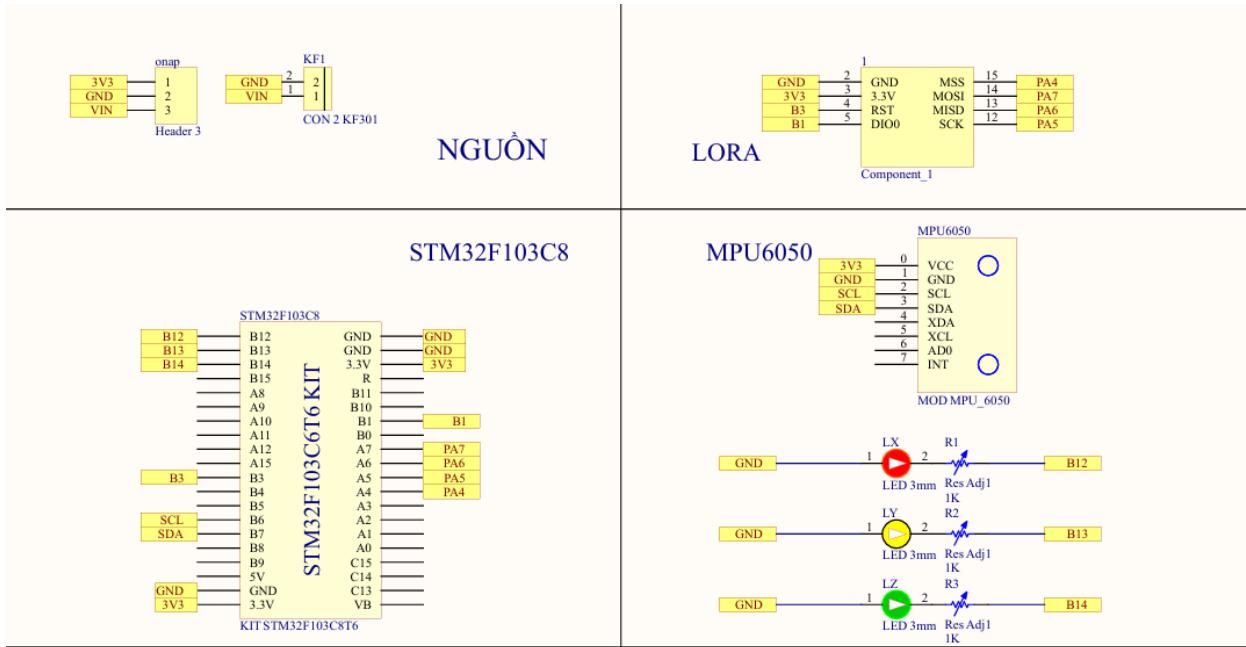
Mạch thu nhận gói dữ liệu từ mạch phát thông qua module LoRa. Dữ liệu sau đó được gửi đến STM32 để xử lý tiếp.

Chạy động cơ theo tín hiệu MPU6050

STM32 ở mạch thu sẽ dịch gói lệnh điều khiển thành tín hiệu logic điều khiển module L298 để quay động cơ theo yêu cầu. Xe sẽ di chuyển đúng với hướng tay người điều khiển.

4.2, Phản ứng hệ thống phát

4.2.1.Sơ đồ mạch điện



Hình 4.1: sơ đồ mạch phát

Mạch phát là phần trung tâm trong hệ thống điều khiển xe từ xa bằng cử chỉ tay. Mạch này được gắn lên tay người dùng và có nhiệm vụ thu nhận chuyển động từ cảm biến MPU6050, xử lý dữ liệu qua vi điều khiển STM32F103C8, sau đó truyền tín hiệu điều khiển đến mạch thu trên xe thông qua module LoRa. Toàn bộ hệ thống được cấp nguồn bằng điện áp 3.3V từ bộ nguồn ngoài.

4.2.1.1. Khởi nguồn

Khởi nguồn có nhiệm vụ cung cấp điện áp ổn định 3.3V cho toàn bộ hệ thống. Nguồn được cấp qua jack CON2 KF301 với hai đầu vào: VIN và GND. Điện áp được điều chỉnh phù hợp cho các linh kiện như STM32, MPU6050 và module LoRa. Khởi nguồn đảm bảo hệ thống hoạt động ổn định và an toàn về điện.

4.2.1.2. Vi điều khiển STM32F103C8

Vi điều khiển STM32F103C8 đóng vai trò là bộ xử lý trung tâm của hệ thống mạch phát. Nó đảm nhiệm các chức năng sau:

Giao tiếp với cảm biến MPU6050 thông qua chuẩn I2C (chân PB6 – SCL và PB7 – SDA).

Giao tiếp với module LoRa thông qua chuẩn SPI (chân PA4 đến PA7).

Phân tích dữ liệu chuyển động, nhận biết cử chỉ tay điều khiển và chuyển đổi thành tín hiệu điều khiển cụ thể cho xe.

Xuất tín hiệu điều khiển LED thông báo trạng thái chuyển động qua các chân PB12, PB13, PB14 (ứng với trục X, Y, Z).

Vi điều khiển được cấp nguồn 3.3V và có các chân GND được nối về mass chung, đảm bảo tính ổn định cho toàn hệ thống.

4.2.1.3. Cảm biến MPU6050

Cảm biến MPU6050 có chức năng ghi nhận dữ liệu chuyển động tay người điều khiển. Cảm biến tích hợp gia tốc kép 3 trục và con quay hồi chuyển 3 trục, cho phép đo được góc nghiêng và tốc độ xoay của tay theo thời gian thực.

Trong mạch phát, MPU6050 được kết nối với STM32 thông qua giao tiếp I2C:

SCL nối với chân PB6.

SDA nối với chân PB7.

Nguồn 3.3V cấp vào chân VCC và GND của cảm biến. Dữ liệu chuyển động thu được từ MPU6050 sẽ được vi điều khiển xử lý để tạo ra lệnh điều khiển thích hợp cho xe.

4.2.1.4. Module truyền thông LoRa

Module LoRa (thường dùng SX1278 hoặc Ra-02) là thành phần đảm nhận việc truyền tín hiệu không dây từ bộ phát đến bộ thu gắn trên xe. Module này hoạt động với tần số 433 MHz và được kết nối với STM32 thông qua giao tiếp SPI, cụ thể:

MOSI (Master Out Slave In): PA7.

MISO (Master In Slave Out): PA6.

SCK (Serial Clock): PA5.

NSS (Chip Select): PA4.

Module nhận dữ liệu điều khiển từ STM32 và phát sóng RF đến module LoRa còn lại trong hệ thống. Nhờ khả năng truyền xa và tiêu thụ năng lượng thấp, LoRa giúp hệ thống hoạt động ổn định trong khoảng cách vài trăm mét đến vài km tùy điều kiện môi trường.

4.2.1.5. Hệ thống LED hiển thị

Ba đèn LED (màu đỏ, vàng, xanh) được sử dụng để hiển thị trạng thái hoạt động của hệ thống theo ba trục chuyển động:

LED đỏ (trục X): nối với PB12.

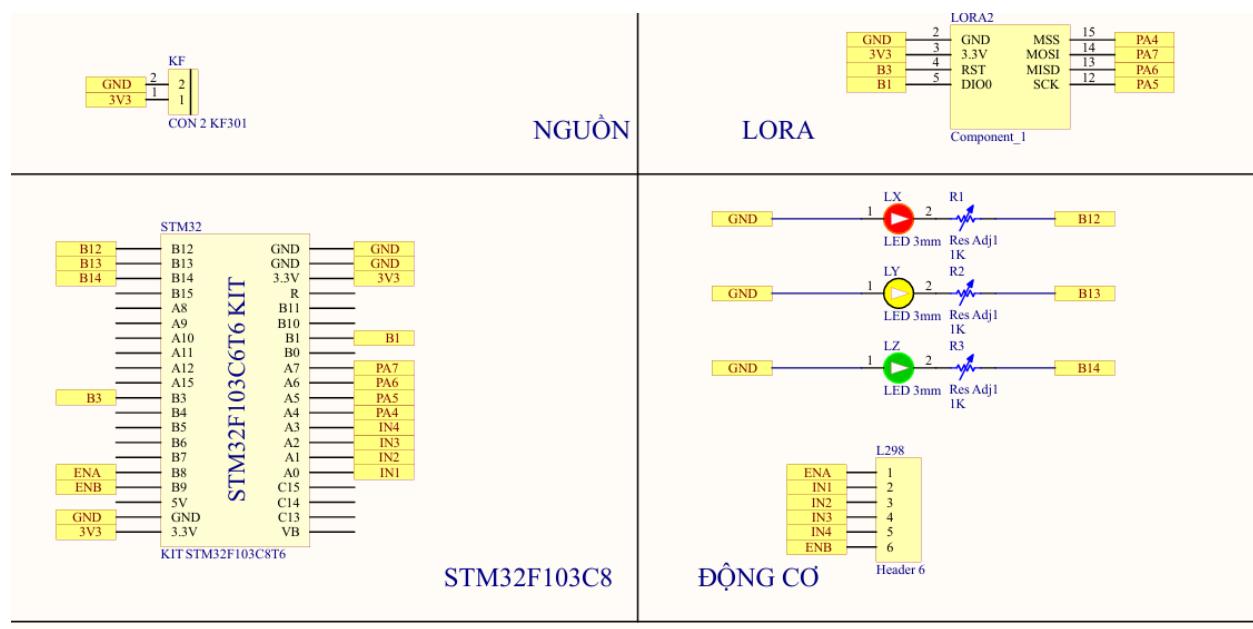
LED vàng (trục Y): nối với PB13.

LED xanh (trục Z): nối với PB14.

Mỗi LED được mắc nối tiếp với điện trở hạn dòng $1K\Omega$ để bảo vệ linh kiện khỏi hư hỏng do quá dòng. Các LED này giúp người dùng dễ dàng nhận biết hướng chuyển động đang được ghi nhận, đồng thời hỗ trợ kiểm tra nhanh hệ thống trong quá trình thử nghiệm.

4.3, Phần cứng hệ thống thu

4.3.1. Sơ đồ mạch điện



Hình 4.2: sơ đồ mạch thu

Mạch thu có nhiệm vụ nhận tín hiệu điều khiển được truyền từ mạch phát thông qua module LoRa, sau đó giải mã tín hiệu và điều khiển động cơ theo các lệnh nhận được.

Mạch thu được gắn trực tiếp trên xe điều khiển, bao gồm các khối chức năng: khối nguồn, vi điều khiển STM32F103C8, module LoRa, bộ điều khiển động cơ (L298), và hệ thống LED hiển thị.

4.3.1.1. Khối nguồn

Khối nguồn sử dụng điện áp 3.3V được cấp từ ngoài qua jack CON2 KF301. Nguồn này được dùng để cấp điện cho vi điều khiển STM32, module LoRa và các phần tử điều khiển khác trong hệ thống. Mass (GND) được nối chung toàn mạch để đảm bảo ổn định điện áp và tránh nhiễu.

4.3.1.2. Vi điều khiển STM32F103C8

STM32F103C8 là bộ xử lý trung tâm của mạch thu, thực hiện các chức năng chính sau:

Giao tiếp SPI với module LoRa để nhận dữ liệu điều khiển (thông qua các chân PA4–PA7).

Phân tích và giải mã tín hiệu nhận được để xác định hướng di chuyển của xe (tiến, lùi, rẽ trái, rẽ phải).

Xuất tín hiệu điều khiển động cơ thông qua các chân PA0–PA3 (kết nối với L298).

Điều khiển LED hiển thị trạng thái hoạt động qua các chân PB12, PB13, PB14.

STM32 được cấp nguồn 3.3V từ khối nguồn chung và sử dụng các chân GND để nối mass.

4.3.1.3. Module truyền thông LoRa

Module LoRa được sử dụng để nhận dữ liệu điều khiển không dây từ bộ phát. Cấu hình giao tiếp giống như trên mạch phát:

MOSI → PA7

MISO → PA6

SCK → PA5

NSS (CS) → PA4

Module sẽ tiếp nhận lệnh điều khiển được mã hóa từ mạch phát và gửi vào STM32 để xử lý.

4.3.1.4. Mạch điều khiển động cơ (L298)

Module L298 là bộ cầu H điều khiển 2 động cơ DC độc lập. Trong mạch thu, nó được điều khiển bởi STM32 thông qua các chân:

IN1 → PA0, IN2 → PA1

IN3 → PA2, IN4 → PA3

ENA và ENB có thể kết nối với nguồn hoặc điều khiển qua PWM (tùy cấu hình phần mềm)

Dựa vào tín hiệu điều khiển từ STM32, L298 sẽ kích hoạt động cơ quay theo hướng mong muốn (tiến, lùi, rẽ). Đây là thành phần chính để chuyển lệnh điều khiển từ cùi chỏ tay thành chuyển động thực tế của xe.

4.3.1.5. LED hiển thị trạng thái

Tương tự như mạch phát, mạch thu sử dụng 3 LED để phản hồi trạng thái điều khiển:

LED đỏ (PB12): trạng thái điều khiển theo trục X

LED vàng (PB13): trạng thái theo trục Y

LED xanh (PB14): trạng thái theo trục Z

Các LED này giúp xác nhận tín hiệu đã được nhận và xử lý, đồng thời hỗ trợ kiểm tra hệ thống trong quá trình vận hành và thử nghiệm. Mỗi LED được mắc nối tiếp với điện trở hạn dòng $1K\Omega$ để bảo vệ linh kiện.

4.4. Phần mềm hệ thống phát

4.4.1. Driver hoạt động MPU6050.

Driver này là bộ điều khiển để giao tiếp STM32 với cảm biến MPU6050 - một cảm biến 6 trục (3 trục gia tốc + 3 trục con quay hồi chuyển) thông qua giao thức I2C. Nó được thiết kế sử dụng thư viện HAL của ST dành cho vi điều khiển STM32

Driver MPU6050 sẽ lưu trữ dữ liệu thô với kiểu dữ liệu là short được đọc trực tiếp từ các thanh ghi của module MPU6050 và dữ liệu đã xử lý với kiểu dữ liệu là float được tính toán từ dữ liệu thô thông qua công thức chuyển đổi:

// Công thức chuyển đổi gia tốc

$acc_x = acc_x_raw / LSB_Sensitivity_ACC;$

// Công thức chuyển đổi con quay

$gyro_x = gyro_x_raw / LSB_Sensitivity_GYRO;$

```

// Công thức chuyển đổi nhiệt độ
temperature = (temperature_raw / 340.0f) + 36.53f;

```

Cấu trúc lưu trữ dữ liệu của MPU6050:

```

typedef struct _MPU6050 {
    // ----- DỮ LIỆU THÔ (RAW DATA) -----
    // Đọc trực tiếp từ cảm biến, chưa qua xử lý
    short acc_x_raw; // Giá trị ADC gia tốc trục X (16-bit có dấu)
    short acc_y_raw; // Giá trị ADC gia tốc trục Y
    short acc_z_raw; // Giá trị ADC gia tốc trục Z
    short temperature_raw; // Giá trị ADC nhiệt độ (16-bit có dấu)
    short gyro_x_raw; // Giá trị ADC con quay trục X
    short gyro_y_raw; // Giá trị ADC con quay trục Y
    short gyro_z_raw; // Giá trị ADC con quay trục Z

    // ----- DỮ LIỆU ĐÃ XỬ LÝ -----
    // Đã chuyển đổi sang đơn vị vật lý
    float acc_x; // Gia tốc trục X (đơn vị g, 1g = 9.8m/s2)
    float acc_y; // Gia tốc trục Y
    float acc_z; // Gia tốc trục Z
    float temperature; // Nhiệt độ (°C)
    float gyro_x; // Tốc độ góc trục X (°/s)
    float gyro_y; // Tốc độ góc trục Y (°/s)
    float gyro_z; // Tốc độ góc trục Z (°/s)
} Struct_MP6050;

```

Hàm khởi tạo MPU6050: void MPU6050_Initialization(void)

Driver MPU6050 bắt đầu với việc khởi tạo kết nối giữa STM32 và module MPU6050 với hàm: *void MPU6050_Initialization(void)*

Chương trình sẽ kiểm tra kết nối vật lý (WHO_AM_I), chương trình sẽ đọc dữ liệu từ thanh ghi 0x75 và lấy giá trị dữ liệu vừa đọc đó so sánh với giá trị mặc định 0x68 nếu trùng khớp thì hệ thống STM32 và MPU6050 kết nối vật lý thành công, ngược lại nếu so sánh không khớp thì báo lỗi hệ thống STM32 và MPU6050 kết nối vật lý không thành công.

```
MPU6050_Readbyte(MPU6050_WHO_AM_I, &who_am_i); WHO_AM_I  
if (who_am_i == 0x68) {  
}  
else  
{  
    while (1)  
    {  
        HAL_Delay(500);  
    }  
}
```

Khi hệ thống STM32 và MPU6050 kết nối vật lý thành công thì chương trình sẽ reset cảm biến và thiết lập cài đặt các thông số cơ bản ban đầu như

Cấu hình nguồn clock:

```
MPU6050_Writebyte(MPU6050_PWR_MGMT_1, 0x01);  
HAL_Delay(100);
```

Cài đặt tần số lấy mẫu:

```
MPU6050_Writebyte(MPU6050_SMPRT_DIV, 39);
```

Cấu hình bộ lọc số (DLPF):

```
MPU6050_Writebyte(MPU6050_CONFIG, 0x00);
```

Cài đặt thang đo:

```
MPU6050_Writebyte(MPU6050_GYRO_CONFIG, 0x00 << 3);
```

```
MPU6050_Writebyte(MPU6050_ACCEL_CONFIG, 0x00 << 3);
```

Tính toán độ nhạy LSB:

```
MPU6050_Get_LSB_Sensitivity(0x00, 0x00);
```

Cấu hình ngắt (Interrupt):

```
MPU6050_Writebyte(MPU6050_INT_PIN_CFG, (0x0 << 7) | (0x0 << 5) | (0x1 << 4));
```

```
MPU6050_Writebyte(MPU6050_INT_ENABLE, 0x01);
```

Hàm đọc dữ liệu thô: `MPU6050_Get6AxisRawData(Struct_MPU6050* mpu6050_data)`

Sau khi driver MPU6050 đã khởi tạo kết nối giữa STM32 và module MPU6050 với hàm `void MPU6050_Initialization(void)` thì chương trình sẽ bắt đầu thực hiện đọc dữ liệu bắt đầu từ việc đọc dữ liệu thô từ cảm biến với 14 byte liên tiếp (6 byte gia tốc, 2 byte nhiệt độ và 6 byte gyro).

Chương trình sẽ bắt đầu đọc khỏi dữ liệu 14 byte dữ liệu thô từ cảm biến MPU6050 bắt đầu từ địa chỉ 0x3B :

```
uint8_t data_buf[14];
```

```
MPU6050_Readbytes(MPU6050_ACCEL_XOUT_H, 14, data_buf);
```

Chương trình tiếp tục tạo các giá trị đọc cảm biến thô 16 bit từ 14 byte vừa đọc từ cảm biến MPU6050 và lưu vào các giá trị giá trị vào cá cấu trúc lưu trữ dữ liệu của hàm `typedef struct _MPU6050` :

```
mpu6050_data->acc_x_raw = (int16_t)((data_buf[0] << 8) | data_buf[1]);
```

```
mpu6050_data->acc_y_raw = (int16_t)((data_buf[2] << 8) | data_buf[3]);
```

```
mpu6050_data->acc_z_raw = (int16_t)((data_buf[4] << 8) | data_buf[5]);
```

```
mpu6050_data->temperature_raw = (int16_t)((data_buf[6] << 8) | data_buf[7]);
```

```
mpu6050_data->gyro_x_raw = (int16_t)((data_buf[8] << 8) | data_buf[9]);
```

```
mpu6050_data->gyro_y_raw = (int16_t)((data_buf[10] << 8) | data_buf[11]);
```

```
mpu6050_data->gyro_z_raw = (int16_t)((data_buf[12] << 8) | data_buf[13]);
```

Hàm chuyển đổi dữ liệu: MPU6050_DataConvert(Struct_MPU6050* mpu6050_data)

Sau khi chương trình thực hiện đọc và lưu các giá trị dữ liệu thô từ cảm biến MPU6050 thì chương trình tiếp tục tính toán từ các dữ liệu thô đó sang các đơn vị vật lý với công thức tính toán:

Gia tốc:

```
mpu6050_data->acc_x = (float)mpu6050_data->acc_x_raw /  
LSB_Sensitivity_ACC;
```

```
mpu6050_data->acc_y = (float)mpu6050_data->acc_y_raw /  
LSB_Sensitivity_ACC;
```

```
mpu6050_data->acc_z = (float)mpu6050_data->acc_z_raw /  
LSB_Sensitivity_ACC;
```

Nhiệt độ:

```
mpu6050_data->temperature = (float)mpu6050_data->temperature_raw / 340.0f  
+ 36.53f;
```

Gyro:

```
mpu6050_data->gyro_x = (float)mpu6050_data->gyro_x_raw /  
LSB_Sensitivity_GYRO;
```

```
mpu6050_data->gyro_y = (float)mpu6050_data->gyro_y_raw /  
LSB_Sensitivity_GYRO;
```

```
mpu6050_data->gyro_z = (float)mpu6050_data->gyro_z_raw /  
LSB_Sensitivity_GYRO;
```

Xử lý dữ liệu.

Sau khi đọc và lưu các dữ liệu thô và dữ liệu vật lý sau khi tính toán, chương trình sẽ kiểm tra dữ liệu vừa đọc và lưu vừa thực hiện xong đã sẵn sàng chưa, khi dữ liệu đã sẵn sàng thì chân INT được cài đặt sẽ được kích lên mức high và hàm sẽ trả về là 1, ngược lại

nếu dữ liệu chưa sẵn sàng thì chân INT được cài đặt sẽ không được kích lên mức high và hàm sẽ trả về là 0 :

```
int MPU6050_DataReady(void){  
    return    HAL_GPIO_ReadPin(MPU6050_INT_PORT,    MPU6050_INT_PIN) ==  
            GPIO_PIN_SET;  
}
```

4.4.2. Driver SX1278.

Driver SX1278 cho phép vi điều khiển STM32F103C6T6a giao tiếp và điều khiển module LoRa SX1278. Driver cung cấp cho vi điều khiển STM32 một bộ giao tiếp không dây để thực hiện các tác vụ chính như khởi tạo, cấu hình, gửi và nhận dữ liệu qua sóng LoRa, giúp điều khiển các hệ thống từ xa. Driver SX1278 được phân tách thành lớp logic điều khiển module LoRa SX1278 và lớp trừu tượng phần cứng, giúp tăng tính module hóa và khả năng tái sử dụng.

Lớp trừu tượng phần cứng (Hardware Abstraction Layer) viết tắt là hw là nền tảng cốt lõi của driver. Nó thực hiện tách biệt hoàn toàn phần logic điều khiển module SX1278 với các chi tiết triển khai phần cứng cụ thể trên vi điều khiển STM32. Nó giúp chương trình sạch sẽ và dễ bảo trì. Nếu muốn chuyển driver này sang một dòng vi điều khiển khác (ví dụ: ESP32, Arduino...), chúng ta chỉ cần viết lại nội dung của lớp này (SX1278_hw.c) để sử dụng các hàm API của nền tảng mới, trong khi toàn bộ lớp logic điều khiển module có thể được giữ nguyên mà không cần chỉnh sửa.

Lớp trừu tượng phần cứng tạo cho chương trình một cấu trúc dữ liệu chứa tất cả thông tin kết nối phần cứng :

```
typedef struct {  
    int pin;      // Số hiệu chân (ví dụ: GPIO_PIN_5)  
    void *port;   // Cổng GPIO (ví dụ: GPIOA)  
} SX1278_hw_dio_t;  
  
typedef struct {  
    SX1278_hw_dio_t reset; // Chân Reset  
    SX1278_hw_dio_t dio0; // Chân ngắt DIO0  
    SX1278_hw_dio_t nss; // Chân Chip Select (SPI NSS)
```

```

void *spi;           // Handle SPI (từ HAL)
} SX1278_hw_t;

```

nss (Network Select Slave): Đây là chân chọn chip. Driver này thực hiện việc điều khiển chân NSS bằng phần mềm. Trước mỗi giao tiếp SPI, driver sẽ kéo chân này xuống mức thấp (LOW) và sau khi kết thúc sẽ kéo lên mức cao (HIGH).

reset: Chân này được kết nối đến chân RESET của module SX1278. Driver sử dụng nó để thực hiện quá trình reset cứng cho module khi cần thiết, để giúp module SX1278 quay về trạng thái mặc định.

dio0: Chân Digital I/O 0 của module SX1278. Đây là chân tín hiệu đầu vào cho vi điều khiển. Nó được module SX1278 sử dụng truyền nhận dữ liệu.

spi: Một con trỏ kiểu void*. Nó sẽ trỏ đến địa chỉ của biến handle SPI do STM32CubeMX tạo ra (ví dụ: &hspi1).

Chương trình khi khởi động sẽ bắt đầu bằng việc khởi tạo phần cứng với hàm `_weak void SX1278_hw_init(SX1278_hw_t *hw)`, hàm sẽ thiết lập các trạng thái ban đầu đặt chân nss ở mức cao để vô hiệu hóa SPI và chân reset ở mức cao để module lora ra02-sx 1278 không bị reset:

```

_weak void SX1278_hw_init(SX1278_hw_t *hw) {
    SX1278_hw_SetNSS(hw, 1); // Tắt chọn chip
    HAL_GPIO_WritePin(hw->reset.port, hw->reset.pin, GPIO_PIN_SET);
}

```

Hàm điều khiển GPIO NSS (`SX1278_hw_SetNSS`) dùng để điều khiển chân nss của module LoRa SX1278 với việc đặt value = 0 chân nss kích mức thấp Lora SX1278 ngừng hoạt động giao tiếp và ngược lại đặt value = 1 chân nss kích mức thấp Lora SX1278 bắt đầu hoạt động giao tiếp:

```

void SX1278_hw_SetNSS(SX1278_hw_t* hw, int value) {
    HAL_GPIO_WritePin(hw->nss.port, hw->nss.pin,
                      (value == 1) ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

```

Hàm điều khiển reset module LoRa SX1278 (`SX1278_hw_Reset`) dùng để thực hiện việc reset module về thiết lập ban, chương trình đầu bắt đầu thực hiện với việc ngừng giao tiếp với SPI, kéo chân reset xuống mức thấp (0V), giữ trạng thái reset của module ít nhất 1ms (theo thông số của nhà sản xuất), sau đó kích reset lên mức cao (3.3V) và cuối cùng chương trình chờ 100ms để chờ module LoRa Ra_02 - SX1278 khởi động lại:

```

void SX1278_hw_Reset(SX1278_hw_t* hw) {
    SX1278_hw_SetNSS(hw, 1);
    HAL_GPIO_WritePin(hw->reset.port, hw->reset.pin, GPIO_PIN_RESET);
    SX1278_hw_DelayMs(1);
    HAL_GPIO_WritePin(hw->reset.port, hw->reset.pin, GPIO_PIN_SET);
    SX1278_hw_DelayMs(100);
}

```

Hàm giao tiếp với SPI để gửi dữ liệu (*SX1278_hw_SPICommand*) dùng để cho module LoRa SX1278 có thể giao tiếp với SPI của vi xử lý STM32 thực hiện vai trò truyền gửi dữ liệu, hàm sẽ thực hiện kéo chân NSS xuống mức thấp (0V), sau đó truyền 1 byte dữ liệu qua chân MOSI và cuối cùng là chờ quá trình hoàn tất:

```

void SX1278_hw_SPICommand(SX1278_hw_t* hw, uint8_t cmd) {
    SX1278_hw_SetNSS(hw, 0);
    HAL_SPI_Transmit(hw->spi, &cmd, 1, 1000);
    while (HAL_SPI_GetState(hw->spi) != HAL_SPI_STATE_READY);
}

```

Hàm giao tiếp với SPI để nhận dữ liệu (*SX1278_hw_SPIReadByte*) dùng để cho module LoRa SX1278 có thể giao tiếp với SPI của vi xử lý STM32 thực hiện vai trò nhận dữ liệu, hàm sẽ thực hiện gửi 1 byte rỗng 0x00 qua MOSI, sau đó gọi hàm *HAL_SPI_TransmitReceive* để nhận byte dữ liệu từ SX1278 qua MISO cùng lúc:

```

uint8_t SX1278_hw_SPIReadByte(SX1278_hw_t* hw) {
    uint8_t txByte = 0x00;
    uint8_t rxByte = 0x00;
    HAL_SPI_TransmitReceive(hw->spi, &txByte, &rxByte, 1, 1000);
    return rxByte;
}

```

Hàm đọc trạng thái của chân DIO0 (*SX1278_hw_GetDIO0*) dùng để kiểm tra trạng thái của DIO0 nếu như chân này đang trong quá trình đọc hoặc nhận dữ liệu thì chân này sẽ ở mức cao và ngược lại nếu chân này không hoạt động thì chân này sẽ ở mức thấp:

```

__weak int SX1278_hw_GetDIO0(SX1278_hw_t *hw) {
}

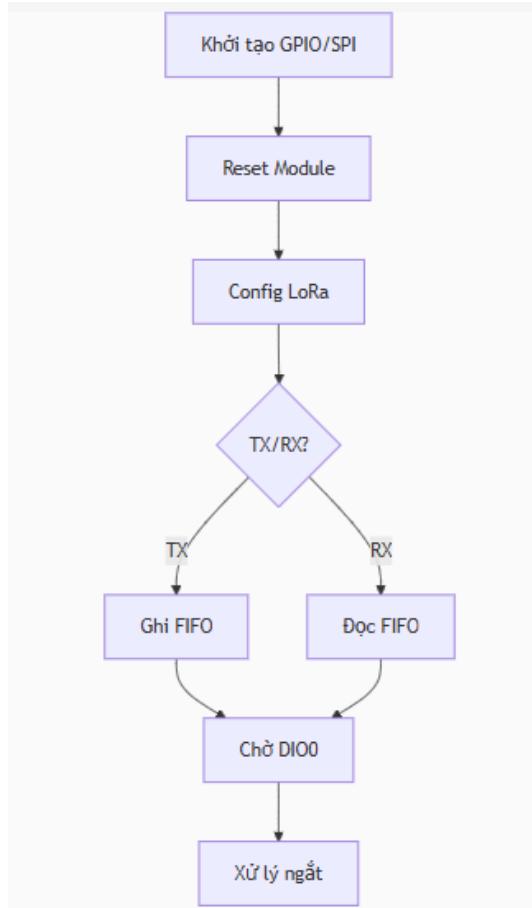
```

```

    return (HAL_GPIO_ReadPin(hw->dio0.port, hw->dio0.pin) == GPIO_PIN_SET);
}

```

Sơ đồ khái hoạt động của chương trình của lớp trùu tượng phần cứng:



Hình 4.3: lưu đồ thuật toán

Lớp logic điều khiển module LoRa SX1278

Lớp điều khiển LoRa (SX1278) là lớp quản lý giao tiếp giữa vi xử lý STM32 với module LoRa SX1278 thông qua lớp phần cứng, cung cấp các chức năng:

Cấu hình thông số hoạt động của LoRa (tần số, công suất, SF, BW, CR).

Truyền và nhận dữ liệu không dây thông qua giao thức LoRa.

Quản lý trạng thái hoạt động module LoRa (Sleep/Standy/TX/RX).

Ở lớp điều khiển LoRa (SX1278) này có cấu trúc dữ liệu như sau:

```

typedef struct {
    SX1278_hw_t *hw;
}

```

```

    uint64_t frequency;
    uint8_t power;
    uint8_t LoRa_SF;
    uint8_t LoRa_BW;
    uint8_t LoRa_CR;
    uint8_t LoRa_CRC_sum;
    uint8_t packetLength;

    SX1278_Status_t status;

    uint8_t rxBuffer[SX1278_MAX_PACKET];
    uint8_t readBytes;
} SX1278_t;

```

Hàm sử dụng con trỏ đến phần cứng *hw để giao tiếp với SPI và thiết lập điều khiển các chân NSS,RESET,DIO0. Hàm còn thực hiện công việc cài đặt các giá trị thông số hoạt động của LoRa như sau:

frequency	862–1020 MHz (tùy khu vực)	Tần số hoạt động, ví dụ 868000000 cho 868MHz
power	0 (20dBm) – 3 (11dBm)	Công suất phát, sử dụng mảng SX1278_Power[] để chuyển đổi
LoRa_SF	6–12	Spreading Factor (độ rộng symbol), ảnh hưởng đến tốc độ và độ nhạy
LoRa_BW	0 (7.8kHz) – 9 (500kHz)	Băng thông, sử dụng mảng SX1278_LoRaBandwidth[]
LoRa_CR	0 (4/5) – 3 (4/8)	Tỉ lệ mã hóa, ảnh hưởng đến khả năng chống nhiễu
LoRa_CRC_sum	0 (Enable) hoặc 1 (Disable)	Kiểm tra CRC để phát hiện lỗi

Bảng 4.2. thông số lora

Quản lý trạng thái hoạt động của LoRa:

```
typedef enum _SX1278_STATUS {
```

```
    SLEEP,
```

```
    STANDBY,
```

```
    TX,
```

```
    RX
```

```
} SX1278_Status_t;
```

và cuối cùng là thực hiện việc quản lý dữ liệu truyền nhận:

```
uint8_t rxBuffer[SX1278_MAX_PACKET];
```

```
uint8_t readBytes;
```

Các hàm trong chương trình để hoạt động quá trình truyền nhận dữ liệu bằng module LoRa:

Hàm Khởi Tạo Module (*SX1278_init*) dùng để thiết lập các tham số hoạt động của module LoRa (tần số, công suất, SF, BW, CR, CRC), khởi tạo phần cứng (SPI, GPIO) và cuối cùng là cấu hình các thanh ghi của module LoRa thông qua hàm *SX1278_config*:

```
void SX1278_init(SX1278_t *module, uint64_t frequency, uint8_t power,
```

```
    uint8_t LoRa_SF, uint8_t LoRa_BW, uint8_t LoRa_CR,
```

```
    uint8_t LoRa_CRC_sum, uint8_t packetLength) {
```

```
    module->frequency = frequency;
```

```
    module->power = power;
```

```
    module->LoRa_SF = LoRa_SF;
```

```
    module->LoRa_BW = LoRa_BW;
```

```
    module->LoRa_CR = LoRa_CR;
```

```
    module->LoRa_CRC_sum = LoRa_CRC_sum;
```

```
    module->packetLength = packetLength;
```

```
    SX1278_hw_init(module->hw);
```

```
SX1278_config(module);  
}
```

Hàm cấu hình LoRa (*SX1278_config*) chương trình thực hiện chuyển chế độ module sang sleep, sau đó chuyển sang chế độ LoRa và thiết lập tần số thông qua tính toán các giá trị thanh ghi từ tần số mong muốn , thiết lập công suất dùng bảng *SX1278_Power[]* để chọn mức phù hợp, thiết lập cấu hình Implicit Header mode và cuối cùng chuyển sang chế độ Standby:

```
void SX1278_config(SX1278_t *module) {  
    SX1278_sleep(module); // Chuyển về chế độ Sleep  
    SX1278_entryLoRa(module); // Chuyển sang chế độ LoRa  
  
    uint64_t freq_reg = ((uint64_t)module->frequency << 19) / 32000000;  
    uint8_t freq_buf[3] = {  
        (uint8_t)(freq_reg >> 16),  
        (uint8_t)(freq_reg >> 8),  
        (uint8_t)freq_reg  
    };  
    SX1278278_SPIWrite(module, LR_RegLna, 0x23); // LNA gain  
  
    if (module->LoRa_SF == 6) {  
        SX1278_SPIWrite(module, LR_RegModemConfig1, 0x72);  
    }  
    else {  
        SX1278_SPIWrite(module, LR_RegModemConfig1, 0x74);  
    }  
    SX1278_standby(module); // Chuyển sang Standby  
}
```

Hàm Truyền Dữ Liệu (*SX1278_transmit*) dùng để thực hiện quá trình truyền dữ liệu không dây thông qua module LoRa, chương trình sẽ cấu hình cho LoRa ở chế độ TX và thiết lập FIFO, sau đó thực hiện ghi dữ liệu cần truyền vào FIFO và bắt đầu truyền và cuối cùng là chờ ngắt timeout:

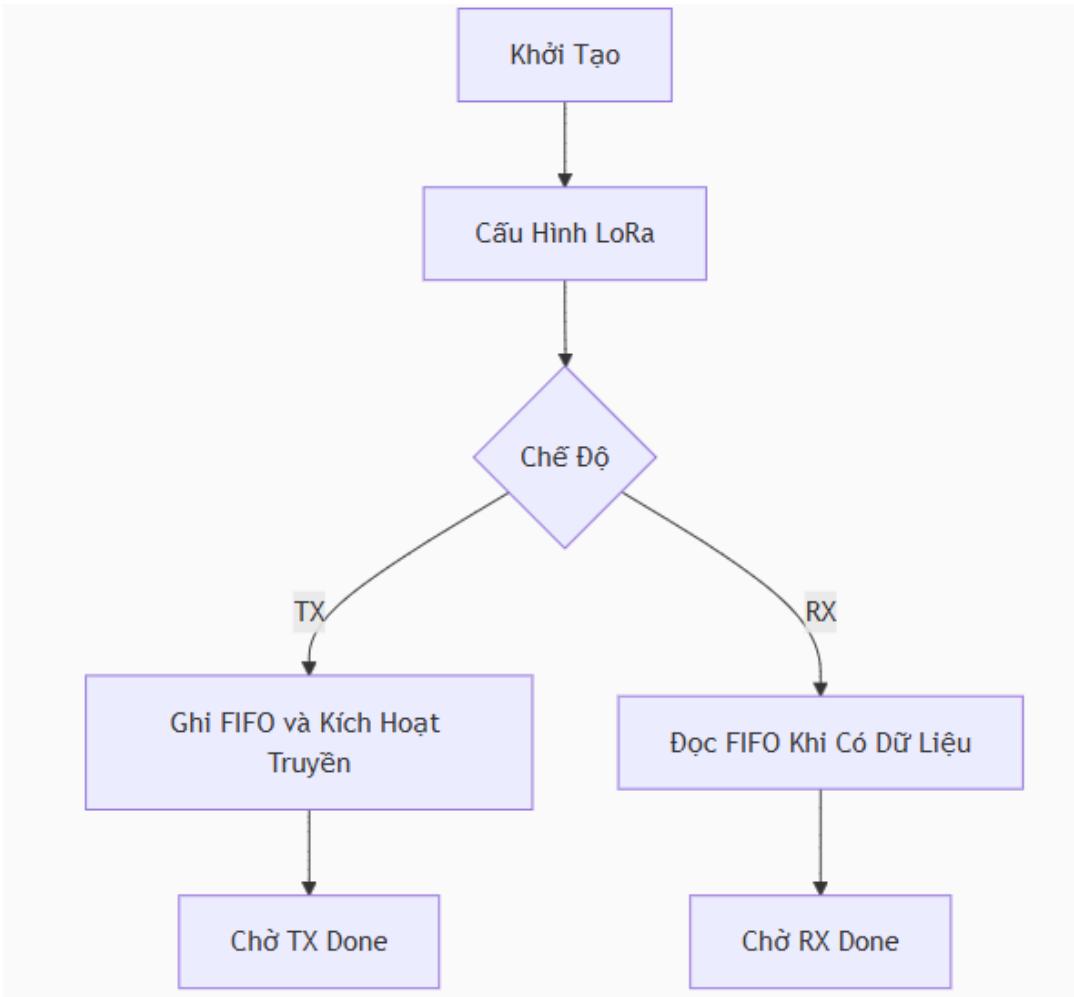
```
int SX1278_transmit(SX1278_t *module, uint8_t *txBuf, uint8_t length, uint32_t timeout)
{
    if (SX1278_LoRaEntryTx(module, length, timeout)) {
        return SX1278_LoRaTxPacket(module, txBuf, length, timeout);
    }
    return 0;
}
```

Hàm Nhận Dữ Liệu (*SX1278_receive*) dùng để thực hiện quá trình nhận dữ liệu không dây thông qua module LoRa, chương trình sẽ cấu hình cho LoRa ở chế độ nhận RX và và bật ngắt RX Done, sau đó kiểm tra DIO0 và đọc dữ liệu từ FIFO, và dữ liệu đó sẽ được lưu vào rxBuffer với độ dài là readBytes:

```
int SX1278_receive(SX1278_t *module, uint8_t length, uint32_t timeout) {
    return SX1278_LoRaEntryRx(module, length, timeout);
}

uint8_t SX1278_available(SX1278_t *module) {
    return SX1278_LoRaRxPacket(module);
}
```

Sơ đồ khái niệm hoạt động của chương trình của lớp logic điều khiển module LoRa SX1278 :

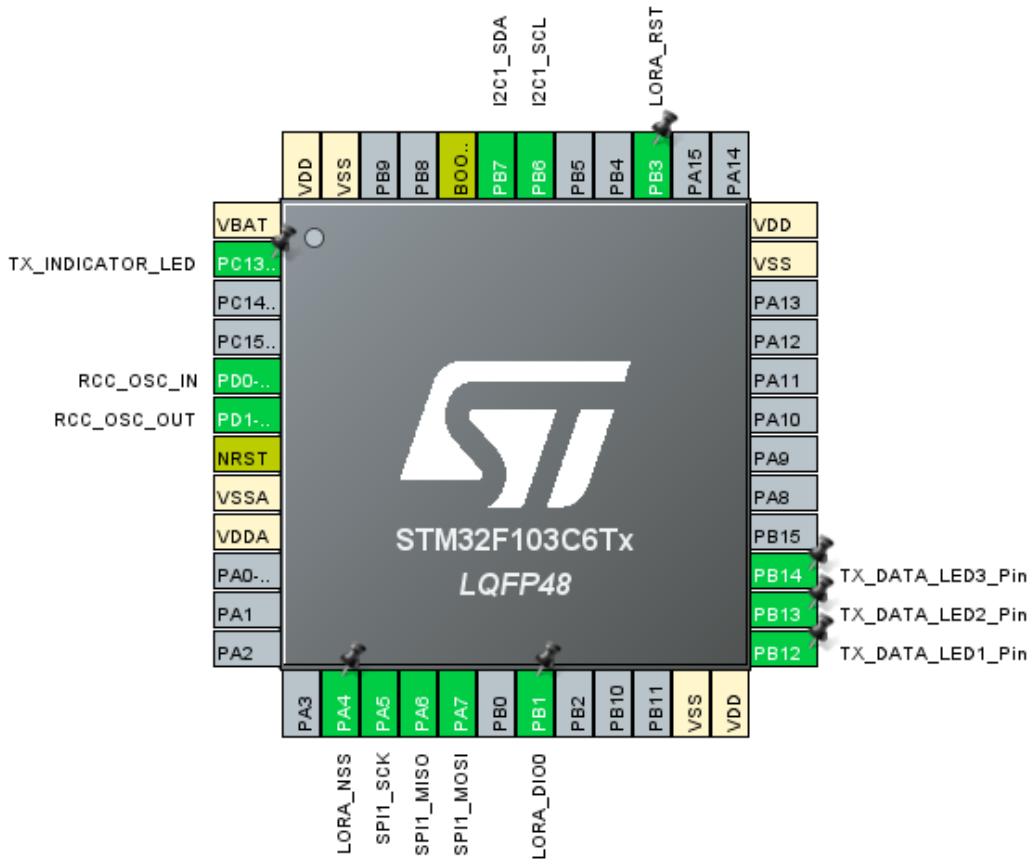


Hình 4.4: lưu đồ thuật toán

4.4.3. Chương trình hệ thống đọc dữ liệu MPU6050 và truyền dữ liệu.

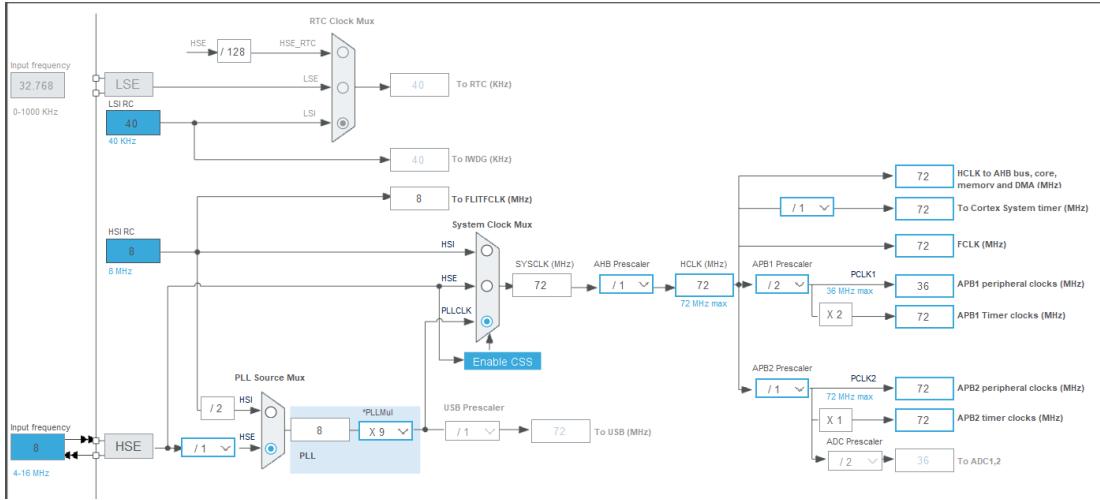
Chương trình được xử lý bởi vi xử lý STM32F103C6T6a với các chân GPIO kết nối với module MPU6050 thông qua I2C và kết nối với module LoRa RA02-SX1278 được

cấu hình như sau:



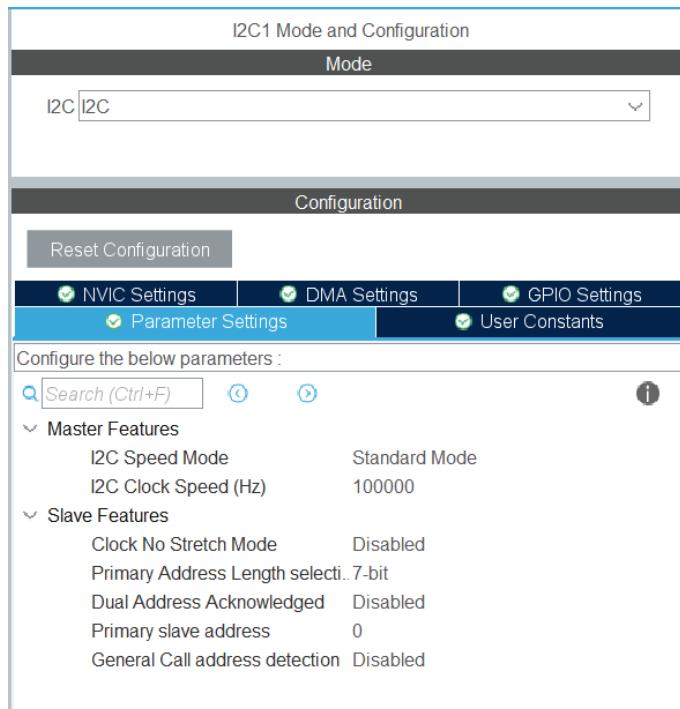
Hình 4.5: thiết lập cấu hình

Với cấu hình Clock Configuration với High Speed Clock (HSE) được cấu hình là Crytal/ceramic và thông số HCLK là 72MHz



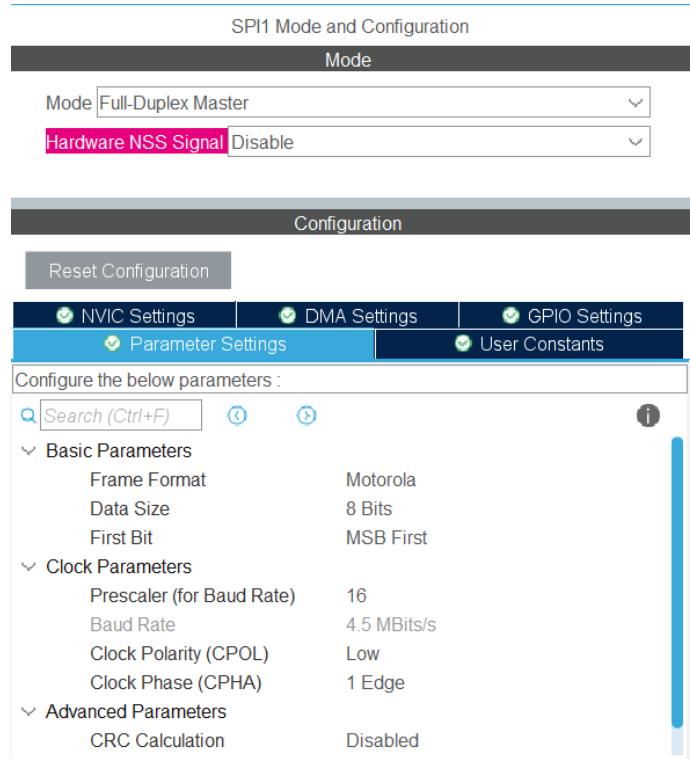
Hình 4.6: thiết lập cấu hình

Vì xử lý STM32 sử dụng giao thức I2C1 để kết nối với MPU 6050 với chân PB7-I2C_SDA và chân PB8-I2C_SCL với thông số cài đặt



Hình 4.7: thiết lập cấu hình

Vì xử lý STM32 sử dụng giao thức SPI1 để kết nối với LoRa SX1278 với chân PA5-SCK, chân PA6-MISO, CHÂN PA7-MOSI và các chân thiết lập GPIO OUTPUT như chân PB1-DIO0, chân PB3-RESET, chân PA4-NSS. Được cài đặt thông số SPI1:



Hình 4.8: thiết lập cấu hình

Chương trình sử dụng vi xử lý đọc cảm biến góc nghiên MPU6050 và gửi dữ liệu bằng LoRa. Chương trình bắt đầu bằng việc khởi tạo các thư viện

```
int main(void) {
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_SPI1_Init();
    MX_I2C1_Init();
    // ...
}
```

Sau khi khởi tạo các thư viện, chương trình bắt đầu khởi tạo MPU 6050 và thiết lập các thông số cho MPU hoạt động:

```
if (HAL_I2C_IsDeviceReady(&hi2c1, MPU6050_I2C_ADDR, 3, 100) != HAL_OK) {
```

```

system_status_flag = 1;
Indicate_Overall_Status(0);
}

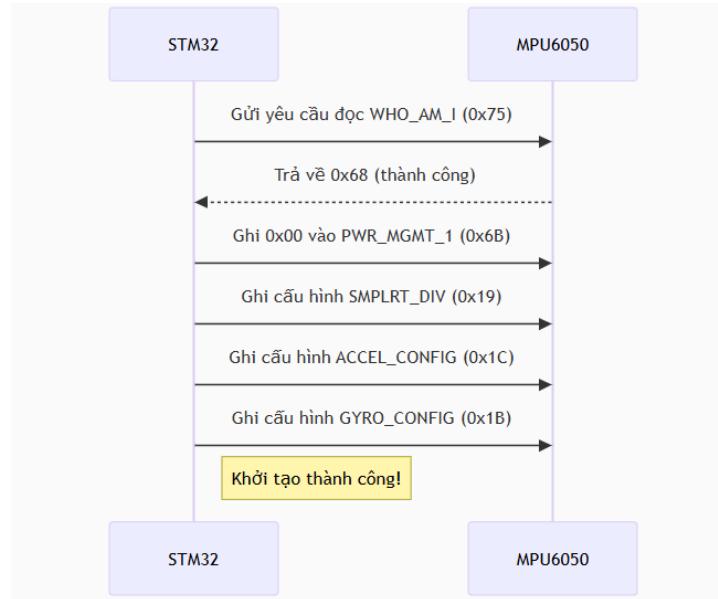
```

```

while (MPU6050_Init_Device() != HAL_OK && system_status_flag == 0) {
system_status_flag = 2;
Indicate_Overall_Status(4);
HAL_Delay(2000);
}
if (system_status_flag == 0) {
system_status_flag = 0;
Indicate_Overall_Status(1);
}

```

Sơ đồ khôi khởi tạo giữa STM32 và MPU6050 thông qua giao thức I2C:



Hình 4.9: thiết lập cấu hình

Tiếp tục chương trình sẽ khởi tạo LoRa thiết lập các thông số cho LoRa hoạt động:

```
if(system_status_flag == 0) {  
    LoRa_HW_TX.spi = &hspi1;  
  
    LoRa_HW_TX.nss.port = LORA_NSS_GPIO_Port; LoRa_HW_TX.nss.pin =  
    LORA_NSS_Pin;  
  
    LoRa_HW_TX.reset.port = LORA_RST_GPIO_Port; LoRa_HW_TX.reset.pin =  
    LORA_RST_Pin;  
  
    LoRaModule_TX.hw = &LoRa_HW_TX;  
  
  
    SX1278_hw_Reset(&LoRa_HW_TX);  
    SX1278_init(&LoRaModule_TX,  
    LORA_FREQUENCY_HZ, LORA_TX_POWER, LORA_SF,  
    LORA_BANDWIDTH, LORA_CODING_RATE, LORA_CRC_ENABLE,  
    LORA_PACKET_MAX_LEN);  
    uint8_t version = SX1278_SPIRead(&LoRaModule_TX, REG_LR_VERSION);  
  
  
    if(version == 0x12) {  
        Indicate_Overall_Status(1);  
    } else {  
        system_status_flag = 3; // LoRa init failed  
        Indicate_Overall_Status(0); // Dừng với lỗi nghiêm trọng } }  
}
```

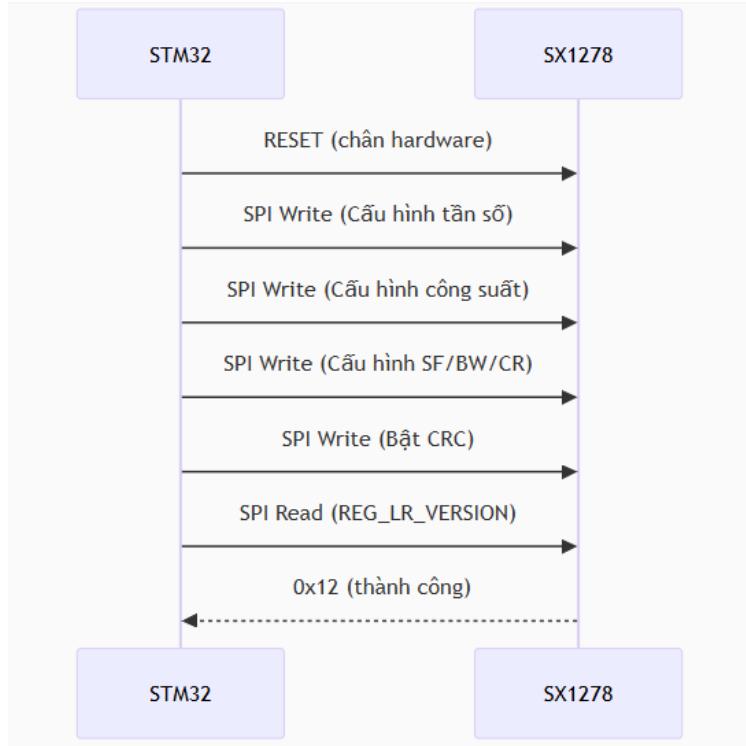
Các thông số được cài đặt để LoRa hoạt động:

SF (Spreading Factor)	7	Tốc độ cao, khoảng cách ngắn
BW (Bandwidth)	125kHz	Độ rộng kênh cân bằng
CR (Coding Rate)	4/5	Hiệu suất truyền cao
Tần số	433MHz	Phổ ISM, khoảng cách tốt
Công suất	17dBm	Cân bằng giữa phạm vi và tiêu thụ năng

	lượng
--	-------

Bảng 4.3 thông số lora

Sơ đồ khôi quá trình khởi tạo giữa STM32 và LoRa SX1278 thông qua giao thức SPI:



Hình 4.10: thiết lập cấu hình

Sau khi chương trình khởi động các thư viện cần thiết cho hệ thống, chương trình sẽ bắt đầu chạy vào vòng lặp while

Ở trong vòng lặp này chương trình sẽ thực hiện đọc 6 byte liên tiếp từ thanh ghi ACCEL_XOUT_H của MPU6050 và sau đó tính toán chuyển sang giá trị vật lý và lưu trữ với công thức

$$mpu_data.Ax = raw_value / 16384.0f;$$

Từ giá trị vật lý vừa được lưu trữ chương trình sẽ tiếp tục so sánh các giá trị đó để biết được MPU6050 đang ở trạng thái góc nghiêng nào.

Từ góc nghiêng đó vi xử lý STM32 sẽ thực hiện điều khiển Led tương ứng và lưu bằng 1 giá trị có kích thước 1 byte để truyền dữ liệu bằng LoRa với bảng giá trị

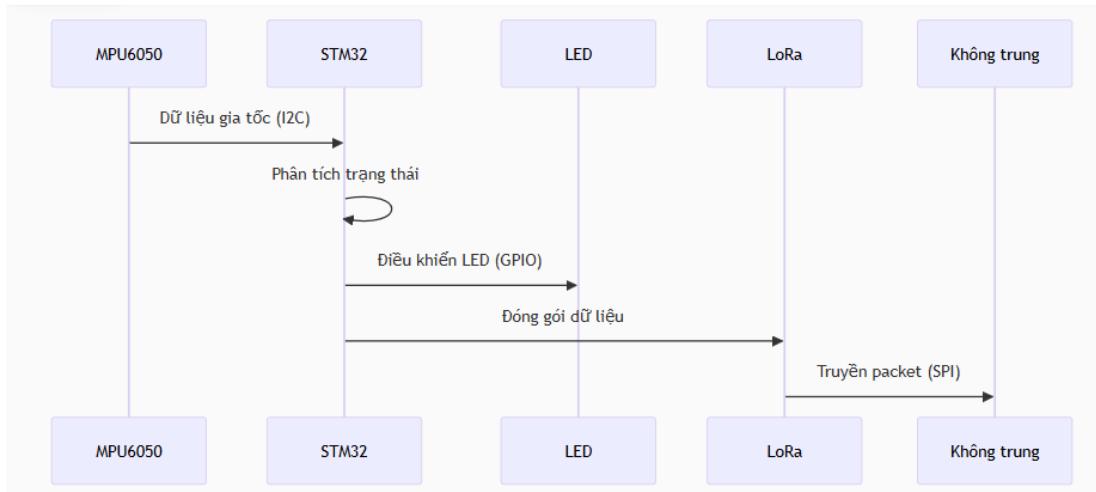
0	Phẳng	Tất cả tắt
1	Nghiêng phải	PB12 sáng
2	Nghiêng trái	PB12 sáng

3	Ngửa lên	PB14 sáng
4	Cúi xuống	PB13 sáng

Bảng 4.4. thiết lập mpu6050

Cuối cùng hệ thống sẽ truyền dữ liệu vừa được lưu trữ thông qua LoRa đã cài đặt và hệ thống sẽ delay 200 ms để hệ thống chạy ổn định hơn

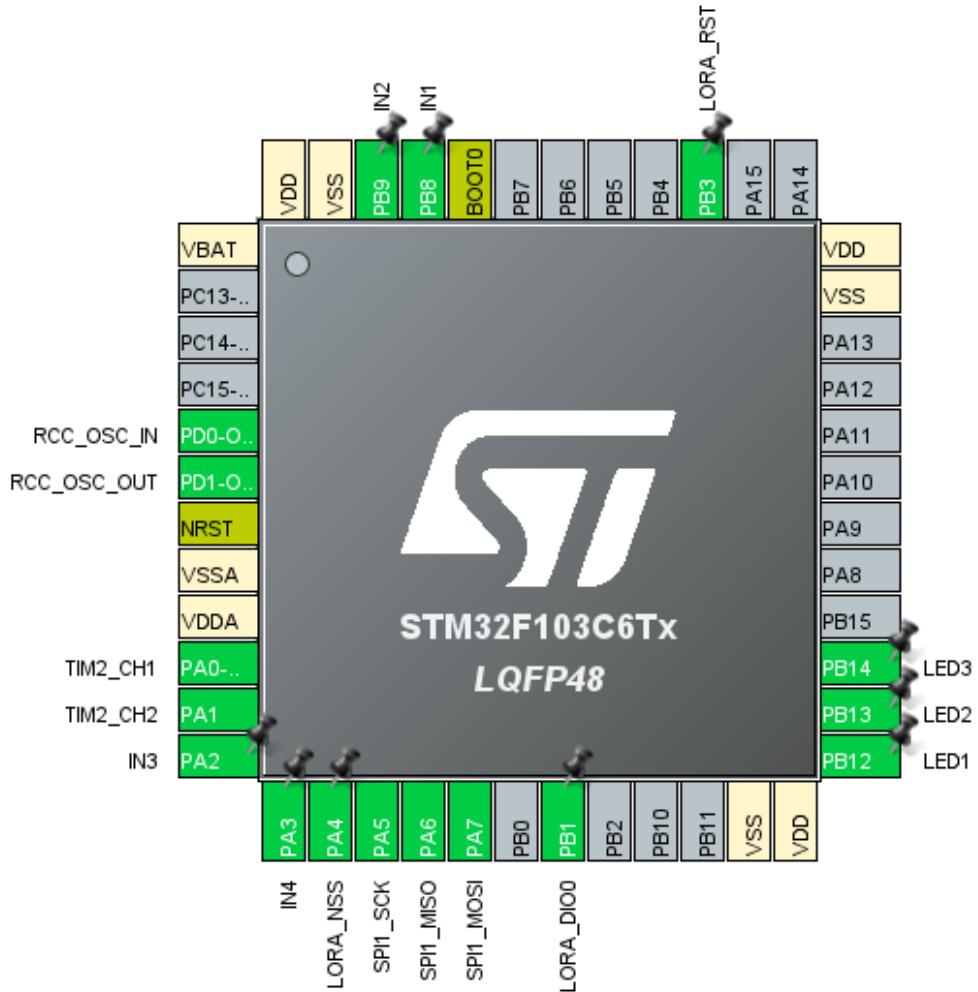
Sơ đồ khái niệm hoạt động của hệ thống phát dữ liệu:



Hình 4.11: thiết lập cấu hình

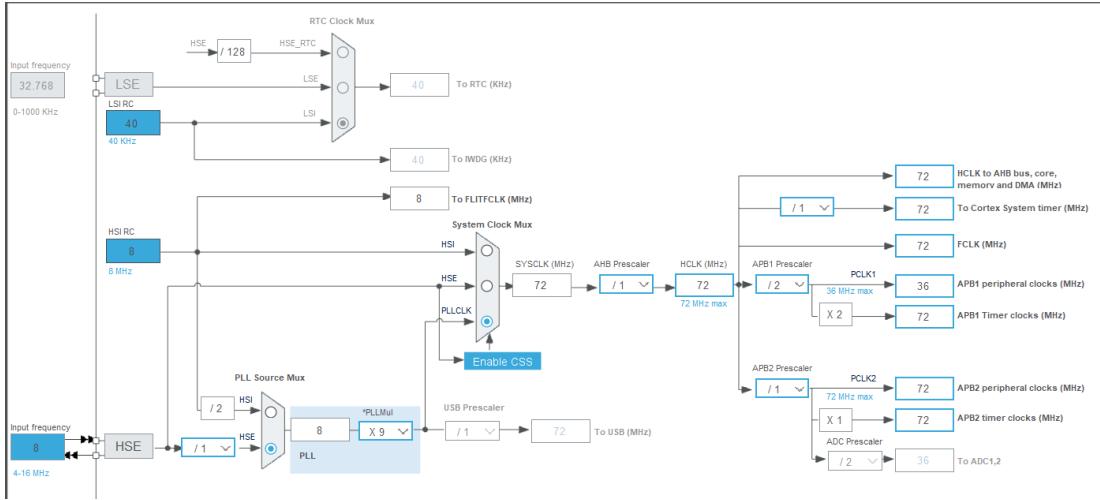
4.5. Phần mềm hệ thống thu

Chương trình được xử lý bởi vi xử lý STM32F103C6T6a với các chân GPIO kết nối với module LoRa RA02-SX1278 và bộ điều khiển động cơ L298 được kết nối như sau:



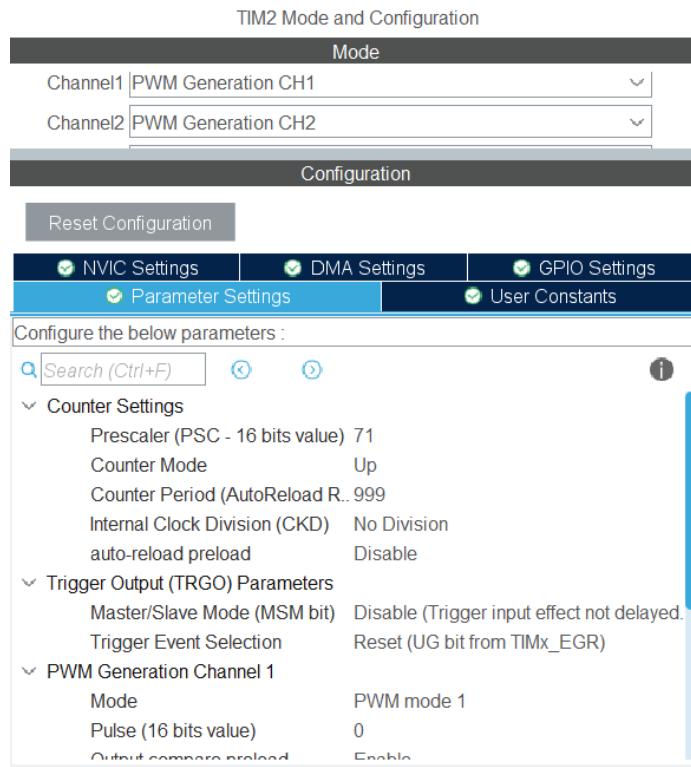
Hình 4.12: thiết lập cấu hình

Với cấu hình Clock Configuration với High Speed Clock (HSE) được cấu hình là Crytal/ceramic và thông số HCLK là 72MHz



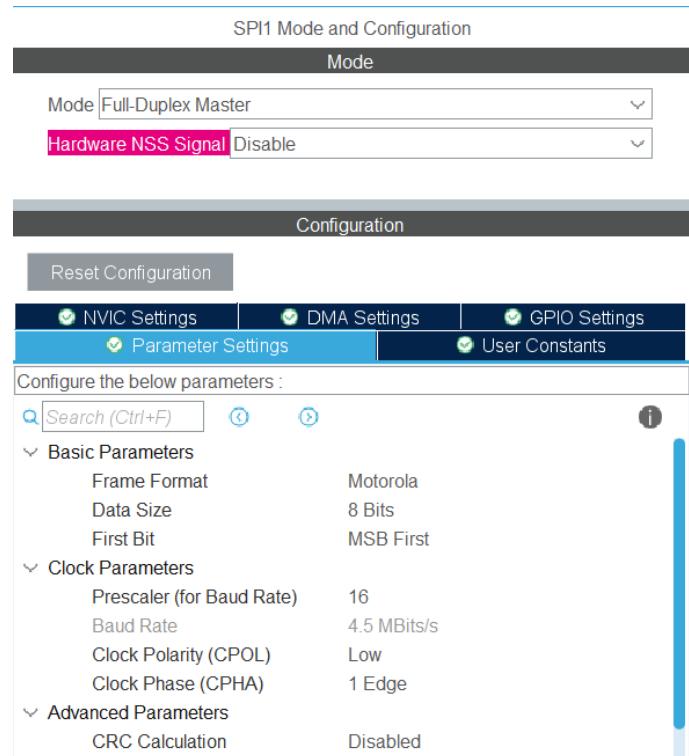
Hình 4.13: thiết lập cấu hình

Vì xử lý STM32 sử dụng giao thức TIM1 để kết nối với chân PA0-TIM2_CH1, chân TIM2_CH2 và các chân GPIO OUTPUT chân PB8-IN1, chân PB9-IN2, chân PA2-IN2, chân PA3-IN4 với thông số cài đặt TIM2



Hình 4.14: thiết lập cấu hình

Vì xử lý STM32 sử dụng giao thức SPI1 để kết nối với LoRa SX1278 với chân PA5-SPI_SCK , chân PA6-SPI_MISO, CHÂN PA7-SPI_MOSI và các chân thiết lập GPIO OUTPUT như chân PB3-RESET, chân PA4-NSS và cuối cùng chân PB1-DIO0 được cài đặt cấu hình là EXTI. Được cài đặt thông số SPI1:



Hình 4.15: thiết lập cấu hình

Chương trình sử dụng vi xử lý STM32 nhận dữ liệu được gửi bằng LoRa và từ dữ liệu đó STM32 điều khiển động cơ DC thông qua bộ điều khiển động cơ L298. Chương trình bắt đầu bằng việc khởi tạo các thư viện

```
int main(void) {
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_SPI1_Init();
    MX_TIM2_Init();
    // ...
}
```

```
}
```

Sau khi khởi tạo các thư viện, chương trình bắt đầu khởi tạo LoRa SX1278 và thiết lập các thông số cho LoRa SX1278 hoạt động, các thông số LoRa nhận dữ liệu này phải trùng khớp với LoRa ở đầu phát thì hệ thống mới truyền nhận dữ liệu thành công và đặt cho module LoRa ở chế độ nhận dữ liệu liên tục không ngừng nghỉ:

```
SX1278_hw_t LoRa_HW_RX = {  
    .spi = &hspil,           // SPI1  
    .nss.port = LORA_NSS_GPIO_Port, .nss.pin = LORA_NSS_Pin,  
    .reset.port = LORA_RST_GPIO_Port, .reset.pin = LORA_RST_Pin,  
    .dio0.port = LORA_DIO0_GPIO_Port, .dio0.pin = LORA_DIO0_Pin  
};  
  
SX1278_hw_Reset(&LoRa_HW_RX);  
  
SX1278_init(&LoRaModule_RX,  
    LORA_FREQUENCY_HZ,  
    LORA_POWER,  
    LORA_SF,  
    LORA_BANDWIDTH,  
    LORA_CODING_RATE,  
    LORA_CRC_ENABLE,  
    LORA_PACKET_LEN_CONFIG  
);  
  
uint8_t version = SX1278_SPIRead(&LoRaModule_RX, REG_LR_VERSION);  
if(version == 0x12) /* SX1278 hợp lệ */  
  
SX1278_receive(&LoRaModule_RX,LORA_PACKET_LEN_CONFIG,SX1278_DEFAULT_TIMEOUT);
```

Sau khi khởi động LoRa thành công hệ thống bắt đầu tiếp tục khởi tạo 2 kênh PWM bằng TIM2 đã cài đặt bằng STM32CubeIDE, hệ thống sử dụng HAL để khởi tạo 2 kênh PWM và cài đặt thông số tốc độ:

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
#define SPEED_FULL 999
#define SPEED_RUN 200
#define SPEED_TURN 150
```

Vì chân DIO0 của LoRa được cấu hình là ngắt nên khi nhận tín hiệu thì chân DIO0 sẽ được kích lên mức cao để báo hiệu có dữ liệu mới:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if(GPIO_Pin == LORA_DIO0_Pin) {
        loraPacketReadyFlag_RX = 1;
    }
}
```

Để hệ thống không bị bỏ sót dữ liệu mới trong vòng lặp while của chương trình có lệnh xoá bỏ cờ ngắt khi chân DIO0 ở mức cao thì ngay lập tức sẽ bị kéo xuống mức thấp để không bỏ sót dữ liệu nào

Tiếp theo chương trình sẽ kiểm tra dữ liệu vừa nhận được có độ dài bao nhiêu, bao nhiêu byte và so sánh với số byte lớn hơn 0 thì chương trình bắt đầu đọc dữ liệu các byte từ bộ đệm của LoRa.

Tùy dữ liệu vừa nhận được vi xử lý STM32 điều khiển động cơ thông qua TIM2 và đồng thời sáng đèn LED tương ứng với bảng dữ liệu sau

0	Đứng yên	Tắt cả tắt
1	Xoay sang phải	PB12 sáng
2	Xoay sang trái	PB12 sáng
3	Lui về	PB14 sáng
4	Tiến tới	PB13 sáng

Bảng 4.5. thiết lập Led

Cơ chế điều khiển PWM (Pulse Width Modulation) là kỹ thuật điều chỉnh điện áp hiệu dụng bằng cách thay đổi độ rộng xung, độ rộng sung càng nhỏ thì động cơ càng mạnh mẽ nhất là 999 ở SPEED_FULL và đứng yên ở SPEED-STOP

```
void motor1_set_speed(uint16_t speed_pulse) {
    // Giới hạn giá trị PWM trong phạm vi cho phép
    if(speed_pulse > SPEED_FULL) speed_pulse = SPEED_FULL;

    // Cập nhật giá trị PWM cho kênh TIM2 CH1
    HAL_TIM_SET_COMPARE(&htim2, MOTOR1_PWM_CHANNEL, speed_pulse);
}
```

Bảng dữ liệu

Giá trị PWM	Tốc độ	Ví dụ
0	Dừng	Xe đứng yên
150	Chậm	Xe chạy chậm
200	Vừa	Tốc độ bình thường
999	Nhanh	Tốc độ nhanh

Bảng 4.6. thiết lập tốc độ L298

Cơ chế điều khiển động cơ: việc điều khiển động cơ hoạt động với tiến bằng cách kích chân IN1 và lùi bằng cách kích chân IN2, còn PWN như bộ điều khiển tốc độ quyết định động cơ đó chạy với tốc độ bao nhiêu:

IN1	IN2	Hướng chạy	Ví dụ
1	0	Tiến	Xe chạy thẳng
0	1	Lùi	Xe chạy lùi
0	0	Tự do	Xe dừng tự từ
1	1	Phanh đứng yên	Xe đứng yên

Bảng 4.7. thiết lập động cơ L298

Vòng lặp While thực hiện liên tục không delay để cho việc nhận dữ liệu hiệu quả không bị bỏ xót dữ liệu nào từ LoRa

CHƯƠNG 5: KẾT QUẢ THỰC NGHIỆM

5.1: Mô hình phần cứng và thực tế

5.1.1: Phần cứng hệ thống phát



Hình 5.1: phần cứng phát

Bo mạch được thiết kế nhỏ gọn và bố trí hợp lý trên một bảng mạch in (PCB), giúp dễ dàng tích hợp vào các thiết bị di động hoặc hệ thống xe điều khiển từ xa. Các linh kiện chính được hàn cố định chắc chắn trên PCB, đảm bảo kết nối ổn định và an toàn trong quá trình sử dụng.

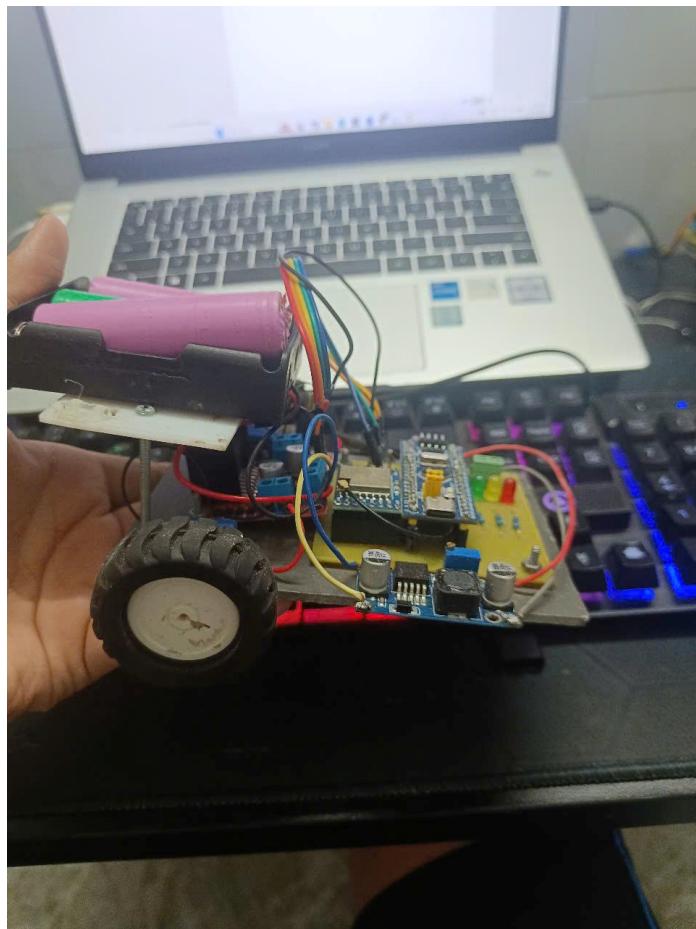
Vi điều khiển STM32F103C8T6 (blue pill) được gắn ở trung tâm bo mạch, đóng vai trò xử lý chính. Cạnh bên phải là module truyền thông LoRa Ra-02 được gắn trên module chuyển mức logic, kết nối với STM32 qua giao tiếp SPI. ăng-ten LoRa được kéo ra ngoài, giúp tăng cường khả năng thu phát tín hiệu trong môi trường rộng.

Ba đèn LED báo trạng thái (đỏ, vàng, xanh) được bố trí thẳng hàng phía trên bên trái của bo mạch, đi kèm với các điện trở hạn dòng $1K\Omega$, giúp hiển thị trạng thái tín hiệu hoặc quá trình xử lý từ MPU6050 một cách trực quan.

Phía dưới cùng bên trái là cảm biến MPU6050, được cố định trực tiếp trên PCB bằng chân header để thuận tiện trong quá trình tháo lắp, thử nghiệm. Cảm biến này sẽ ghi nhận chuyển động của tay người điều khiển và gửi dữ liệu về STM32.

Đầu cáp nguồn được đưa ra phía dưới của bo mạch, sử dụng terminal 2 chân chắc chắn để kết nối với pin hoặc adapter. Việc đi dây ngắn gọn, cố định giúp hạn chế đứt, lỏng dây trong điều kiện di chuyển hoặc rung lắc.

5.1.1: Phần cứng hệ thống thu



Hình 5.2: phần cứng thu

Bo mạch thu được lắp đặt trực tiếp lên khung xe điều khiển hai bánh, kết nối với hệ thống động cơ thông qua mạch điều khiển cầu H L298. Mô hình xe sử dụng khung nhựa nhẹ, dễ gia công, giúp giảm trọng lượng và tối ưu hiệu suất hoạt động.

Trên xe, bo mạch thu gồm các thành phần chính:

Vi điều khiển STM32F103C8T6 nằm ở trung tâm mạch, xử lý tín hiệu từ LoRa và điều khiển động cơ theo lệnh nhận được.

Module LoRa gắn bên cạnh vi điều khiển để nhận tín hiệu điều khiển từ xa thông qua sóng RF.

Bộ điều khiển động cơ L298 được kết nối với 2 động cơ DC của xe, điều khiển hướng quay và tốc độ của động cơ trái/phải.

Ba đèn LED báo trạng thái hỗ trợ hiển thị dữ liệu hoặc tín hiệu điều khiển đã nhận được.

Module nguồn hạ áp được sử dụng để chuyển đổi điện áp từ pin Li-ion (7.4V) xuống mức 5V hoặc 3.3V phù hợp cho các linh kiện nhạy dòng như STM32 và LoRa.

Nguồn cấp chính cho toàn hệ thống là hai viên pin Li-ion 18650 được đặt trong giá đỡ phía sau xe. Pin được hàn dây chắc chắn và cố định bằng khung nhôm để đảm bảo ổn định trong quá trình di chuyển.

Toàn bộ hệ thống dây tín hiệu và dây nguồn được đi gọn gàng bằng dây jump hoặc dây dupont nhiều màu, giúp dễ quan sát và khắc phục sự cố. Bánh xe được kết nối trực tiếp với trục động cơ, đảm bảo xe có thể di chuyển linh hoạt theo các lệnh điều khiển như tiến, lùi, rẽ trái, rẽ phải.

Việc tích hợp mạch thu vào mô hình xe hoàn chỉnh đã tạo nên một hệ thống điều khiển từ xa hoàn chỉnh, có khả năng phản hồi nhanh và hoạt động ổn định trong môi trường thực tế

CHƯƠNG 6: KẾT LUẬN VÀ BÀI HỌC

6.1. Kết luận

Đề tài "Điều khiển xe bằng MPU6050 thông qua LoRa" đã được nghiên cứu, thiết kế và triển khai hoàn chỉnh, đạt được các mục tiêu đặt ra ban đầu. Hệ thống cho phép người dùng điều khiển xe từ xa thông qua cử chỉ tay, sử dụng cảm biến chuyển động MPU6050 để thu thập dữ liệu, xử lý bằng vi điều khiển STM32F103C8 và truyền không dây qua module LoRa.

Qua quá trình thực nghiệm, hệ thống hoạt động ổn định ở khoảng cách truyền vài trăm mét trong điều kiện không vật cản. Xe phản hồi nhanh với các chuyển động tay cơ bản như tiến, lùi, rẽ trái, rẽ phải. Việc sử dụng LoRa giúp hệ thống tiết kiệm năng lượng, không phụ thuộc vào hạ tầng mạng WiFi hay Bluetooth, phù hợp với các ứng dụng ngoài trời hoặc trong môi trường không có kết nối internet.

6.2. Bài học

Qua quá trình nghiên cứu và triển khai đề tài "Điều khiển xe bằng MPU6050 thông qua LoRa", nhóm đã tích lũy được nhiều bài học chuyên sâu trong quá trình làm việc thực tế với hệ thống nhúng, cảm biến và truyền thông không dây. Cụ thể như sau:

Hiểu sâu về cấu trúc và cơ chế hoạt động của MPU6050:

Nhóm không chỉ dừng lại ở việc đọc giá trị cơ bản từ cảm biến, mà đã tìm hiểu chi tiết các thanh ghi cấu hình bên trong MPU6050. Việc lựa chọn dải đo, cấu hình tốc độ lấy mẫu và hiểu rõ nguyên lý hoạt động của DMP (Digital Motion Processor) giúp nhóm xử lý tín hiệu ổn định hơn. Nhóm cũng rút ra được kinh nghiệm trong việc chuyển đổi dữ liệu thô sang góc nghiêng thực tế và áp dụng thuật toán lọc để giảm sai số.

Làm chủ cấu hình truyền thông LoRa và tối ưu hóa thông số:

Nhóm đã nghiên cứu và thử nghiệm các thông số quan trọng của LoRa như: Spreading Factor (SF), Bandwidth (BW) và Coding Rate (CR), từ đó cân bằng giữa tốc độ truyền và khoảng cách truyền phù hợp với môi trường sử dụng. Ngoài ra, việc lựa chọn chế độ phát công suất và cấu hình SPI cũng được thực hiện một cách chủ động thay vì phụ thuộc hoàn toàn vào thư viện.

Ứng dụng thành thạo STM32 cả ở mức HAL và thanh ghi:

Bên cạnh việc sử dụng thư viện HAL để lập trình nhanh, nhóm cũng đã tìm hiểu và thao tác ở mức thanh ghi trong một số khối ngoại vi như SPI, GPIO và Timer để nâng cao

hiệu suất và độ kiểm soát của chương trình. Qua đó, nhóm hiểu rõ hơn về cấu trúc bộ nhớ, hệ thống clock và ngắt (interrupt) của STM32.

TÀI LIỆU THAM KHẢO

[1] TapIT. (2023). Hướng dẫn sử dụng STM32CubeMX và Keil C lập trình STM32. Truy cập từ:

<https://www.tapit.vn/huong-dan-su-dung-stm32cubemx-va-keil-c-lap-trinh-stm32/>

[2] DocHelp.net. (2022). Tổng quan vi điều khiển STM32F103C8T6 và cách sử dụng Keil. Truy cập từ:

<https://pdfcoffee.com/dochelpnet-tong-quan-stm32f103c8t6-va-cach-su-dung-keilpdf-pdf-free.html>

[3] Scribd. (2023). Lập trình vi xử lý bằng chip STM32F103C8T6 theo yêu cầu. Truy cập từ:

<https://www.scribd.com/document/669670633/L%E1%BA%ADp-Tr%C3%ACnh-Vi-X%E1%BB%AD-L%C3%BD-b%E1%BA%B1ng-Chip-STM32F103C8T6-Theo-Y%C3%AAu-C%E1%BA%A7u>

[4] Laptrinhdientu.com. (2022). Giới thiệu cảm biến chuyển động MPU6050 và hướng dẫn sử dụng. Truy cập từ:

<https://www.laptrinhdientu.com/2022/08/MPU6050.html>

[5] Studocu.vn. (2022). Cảm biến gia tốc MPU6050 – phân tích và xử lý tín hiệu.

<https://www.studocu.vn/vn/document/truong-dai-hoc-bach-khoa-ha-noi/cam-bien-gia-toc-mpu6050/114841568>

[6] 4Evn.com. (2021). Cách sử dụng module LoRa SX1278 433MHz.

<https://www.4evn.com/2021/01/cach-su-dung-module-lora-sx1278-433mhz.html>

[7] OhStem Education. (2023). Module LoRa E32 – tài liệu kỹ thuật và cấu hình truyền không dây

<https://docs.ohstem.vn/en/latest/module/cam-bien/lora.html>

[8] Dientu360.com. (2022). Module thu-phát RF LoRa SX1278 UART 3km (AS62-T20)

<https://dientu360.com/module-thu-phat-rf-lora-sx1278-433mhz-uart-3km-as62-t20>

PHỤ LỤC