

HỌC VIỆN KỸ THUẬT MẬT MÃ
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP MÔN LẬP TRÌNH ARM NÂNG CAO

**LẬP TRÌNH VI ĐIỀU KHIỂN TRIỂN
KHAI XE ĐIỀU KHIỂN TỪ XA KẾT HỢP
TRÁNH VẬT CẢN**

Ngành: Công nghệ thông tin

Chuyên ngành: Kỹ thuật phần mềm nhúng

Nhóm thực hiện: Nhóm 1

Cao Thị Thùy Linh CT020328

Nguyễn Huy Long CT020329

Nguyễn Văn Hiền CT020316

Giảng viên: **TS. Nguyễn Đào Trường**

Khoa Công nghệ thông tin – Học viện Kỹ thuật mật mã

Hà Nội, 2021

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

MỤC LỤC

LỜI NÓI ĐẦU	1
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT	2
1.1 Giới thiệu về điều khiển xe từ xa kết hợp tránh vật cản.....	2
1.2 Giới thiệu STM32F103C8T6	2
1.2.1 Giới thiệu về vi xử lý ARM	2
1.2.2 Tóm tắt lịch sử ARM.....	3
1.2.3 Giới thiệu về ARM Cortex-M3	3
1.2.4 Cấu trúc ARM	4
1.2.5 Các thành phần của STM32F103C8T6.....	9
1.3 Giới thiệu module bluetooth HC-05	12
1.4 Giới thiệu Module HC-SR04, servo GS90, L298N	12
1.4.1 Module HC-SR04.....	14
1.4.2 Servo GS90.....	15
1.4.3 Module điều khiển động cơ L298N	16
1.5 Giới thiệu GPIO.....	19
1.6 Giới thiệu UART	24
1.7 Giới thiệu Timer, PWM.....	28
1.8 Giới thiệu về IDE.....	31
1.8.1 Giới thiệu về STM32CubeMX.....	31
1.8.2 Giới thiệu về Keil uVision 5	33
1.9 Quy trình tạo 1 Project.....	34
1.9.1 Tại mục Pinout & Configuration:	36

1.9.2	Cấu hình các ngoại vi	36
1.9.3	Cấu hình xung clock.....	37
1.9.4	Lưu thông tin Project và sinh code.....	38
1.9.5	Các thao tác với Keil C	39
1.9.6	Nạp chương trình.....	40
CHƯƠNG 2: THỰC NGHIỆM TRIỂN KHAI		42
2.1.	Cài đặt, cấu hình	42
2.1.1.	Thiết kế thuật toán cho chương trình	42
2.1.2.	Cấu hình trên STM32CubeMX	43
2.2.	Thiết kế, lắp đặt phần cứng	64
2.2.1.	Cơ sở thiết yếu.....	64
2.2.2.	Sơ đồ.....	66
2.2.3.	Thi công, lắp ráp.....	68
KẾT LUẬN		69
TÀI LIỆU THAM KHẢO.....		70
PHỤ LỤC		71

DANH MỤC HÌNH ẢNH

Hình 1- 1. Các sản phẩm giải trí dùng vi điều khiển ARM	3
Hình 1- 2. Cấu trúc vi điều khiển ARM dùng lõi Cortex-M3	4
Hình 1- 3. Cấu trúc cơ bản của ARM Cortex-M3.....	5
Hình 1- 4. Các thanh ghi của ARM Cortex-M.....	6
Hình 1- 5. Bản đồ bộ nhớ của Cortex –M3.....	7
Hình 1- 6. Luồng lệnh 3 tầng của Cortex-M.....	8
Hình 1- 7. Sơ đồ khối chi tiết hệ thống vi xử lý Cortex-M3.....	8
Hình 1- 8. Kit STM32F103C8T6 SmartV2	10
Hình 1- 9. Module bluetooth HC-05	12
Hình 1- 10. Module HC-SR04	14
Hình 1- 11. Động cơ Servo GS90	16
Hình 1- 12. Module L298.....	17
Hình 1- 13. Cấu hình chân GPIO	20
Hình 1- 14. Chức năng của GPIO	21
Hình 1- 15. Cấu hình đầu ra của GPIO	22
Hình 1- 16. UART trong STM32F103.....	25
Hình 1- 17. Định dạng gói tin của UART.....	26
Hình 1- 18. Sóng PWM.....	30
Hình 1- 19. Kênh Timer để xuất xung PWM.....	31
Hình 1- 20. Giao diện phần mềm STM32CubeM.....	33
Hình 1- 21. Giao diện Keil uVision 5	34
Hình 1- 22. Tạo project mới trên STM32CubeMX	35
Hình 1- 23. Các cấu hình của STM32CubeMX.....	35
Hình 1- 24. Cấu hình nạp code.....	36
Hình 1- 25. Cấu hình các ngoại vi của STM32F103C8T6	37
Hình 1- 26. Các cấu hình ngoại vi khác.....	37
Hình 1- 27. Cấu hình xung clock của STM32F103C8T6.....	38

Hình 1- 28. Lưu thông tin project và sinh code	38
Hình 1- 29. Cài đặt Project.....	39
Hình 1- 30. Các thao tác trên Keil uVision 5.....	39
Hình 1- 31. Vị trí Option for Target trong Keil uVision 5	40
Hình 1- 32. Cửa sổ Option for Target trong Keil uVision 5.....	41
Hình 2- 1. Chọn mạch STM32F103C8T6 trên STM32CubeMX.....	43
Hình 2- 2. Cấu hình sys.....	43
Hình 2- 3. Cấu hình UART	44
Hình 2- 4. Setting GPIO.....	44
Hình 2- 5. Enable interrupt UART 1.....	45
Hình 2- 6. Cấu hình TIM1.....	45
Hình 2- 7. <i>Enable interrupt TIM1</i>	46
Hình 2- 8. Cấu hình TIM3.....	46
Hình 2- 9. <i>Enable interrupt TIM3</i>	47
Hình 2- 10. Setting xung clock cho STM32F103c8t6	47
Hình 2- 11. Cấu hình project manager	48
Hình 2- 12. Sơ đồ khối của hệ thống.....	66
Hình 2- 13. Sơ đồ phân cứng hệ thống	67
Hình 2- 14. Mô hình sau khi lắp ráp 1	68
Hình 2- 15. Mô hình sau khi lắp ráp 2	68

DANH MỤC BẢNG

Table 1- 1. Định dạng các gói tin trong UART27

Table 2- 1. Các cơ sở thiết yếu.....65

LỜI NÓI ĐẦU

Trong xu hướng công nghệ và máy móc hiện đại luôn giành được sự quan tâm đặc biệt của cả đất nước, việc sử dụng các robot ứng dụng vào cuộc sống đang là một giải pháp tân tiến để phục vụ nhiều mục đích nằm ngoài khả năng của con người. Thế giới và nước ta hiện đang đề tâm rất nhiều đến vấn đề chế tạo robot, xe điều khiển,

Đã có rất nhiều cuộc thi được tổ chức để khích lệ tinh thần ham học hỏi và tích lũy kinh nghiệm về những bước đầu chế tạo robot dành cho sinh viên. Trong đó xe dò line, robocon, robot sumo... là những cuộc thi được tổ chức dành cho sinh viên nhiều nhất. Nhận thấy, công nghệ chế tạo robot, xe điều khiển, sẽ là một trong những ngành quan trọng nhất, là một ngành mà bất cứ quốc gia nào cũng rất chú trọng trong việc phát triển đất nước.

Với sự thiết thực đó nhóm em đã nảy sinh ý tưởng về “điều khiển xe từ xa kết hợp tránh vật cản”. Bài tập lớn gồm các nội dung sau:

Chương 1: CƠ SỞ LÝ THUYẾT

Chương 2: THỰC NGHIỆM TRIỂN KHAI

Hà Nội, ngày 03 tháng 10 năm 2021

Sinh viên thực hiện

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 Giới thiệu về điều khiển xe từ xa kết hợp tránh vật cản

Xe điều khiển từ xa là xe được chính con người điều khiển bằng các thiết bị thông minh khác như smartphone thông qua Bluetooth

Xe tránh vật cản là xe có thể tự động di chuyển mà không cần đến người điều khiển- nó có thể tự động tránh các vật cản. Xe tránh vật cản bao gồm : vi điều khiển, module điều khiển động cơ và cảm biến. Xe tránh vật cản có nhiệm vụ tránh vật cản phía trước một cách nhanh chóng và chính xác

1.2 Giới thiệu STM32F103C8T6

1.2.1 Giới thiệu về vi xử lý ARM

ARM (Advanced RISC Machine) là một loại cấu trúc vi xử lý 32-bit và 64-bit kiểu RISC được sử dụng rộng rãi trong các thiết kế nhúng.

Do có đặc điểm tiết kiệm năng lượng, các bộ CPU ARM chiếm ưu thế trong các sản phẩm điện tử di động, mà với các sản phẩm này việc tiêu tán công suất thấp là một mục tiêu thiết kế quan trọng hàng đầu.

Ngày nay, hơn 75% CPU nhúng 32-bit là thuộc họ ARM, điều này khiến ARM trở thành cấu trúc 32-bit được sản xuất nhiều nhất trên thế giới.

Các nhà sản xuất IC đưa ra thị trường hơn 240 dòng vi điều khiển sử dụng lõi ARM.

CPU ARM được tìm thấy khắp nơi trong các sản phẩm thương mại điện tử, từ thiết bị cầm tay (PDA, điện thoại di động, máy đa phương tiện, máy trò chơi cầm tay, và máy tính cầm tay) cho đến các thiết bị ngoại vi máy tính (ổ đĩa cứng, bộ định tuyến để bàn)

1.2.2 Tóm tắt lịch sử ARM

ARM được thành lập vào năm 1990 bởi Advanced RISC Machines Ltd, một công ty liên doanh của Apple Computer, Acorn Computer Group, và VLSI Công nghệ. Năm 1991, ARM giới thiệu họ vi xử lý ARM6 và VLSI đã được cấp phép đầu tiên.

Sau đó, các công ty khác, bao gồm Texas Instruments, NEC, Sharp, và ST Microelectronics, được cấp phép thiết kế vi xử lý ARM, mở rộng các ứng dụng của bộ vi xử lý ARM vào điện thoại di động, đĩa cứng máy tính, hỗ trợ các thiết bị số cá nhân (PDA), hệ thống giải trí gia đình, và nhiều sản phẩm tiêu dùng khác



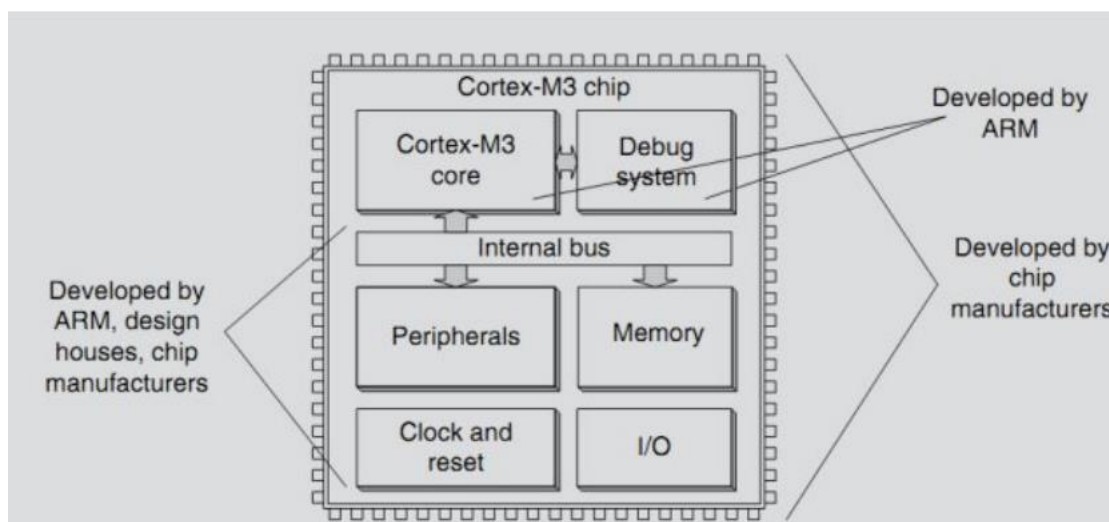
Hình 1- 1. Các sản phẩm giải trí dùng vi điều khiển ARM

1.2.3 Giới thiệu về ARM Cortex-M3

Bộ vi xử lý Cortex-M3 là đơn vị xử lý trung tâm (CPU) của một chip vi điều khiển. Ngoài ra, còn có số lượng các thành phần khác được yêu cầu cho vi điều khiển hoàn toàn dựa vào vi xử lý Cortex-M3.

Sau khi cấp phép cho nhà sản xuất chip vi xử lý Cortex-M3, họ có thể đặt các bộ vi xử lý Cortex-M3 trong các thiết kế bán dẫn của họ, thêm bộ nhớ, thiết bị ngoại vi, đầu vào/ra (I/O), và các tính năng khác.

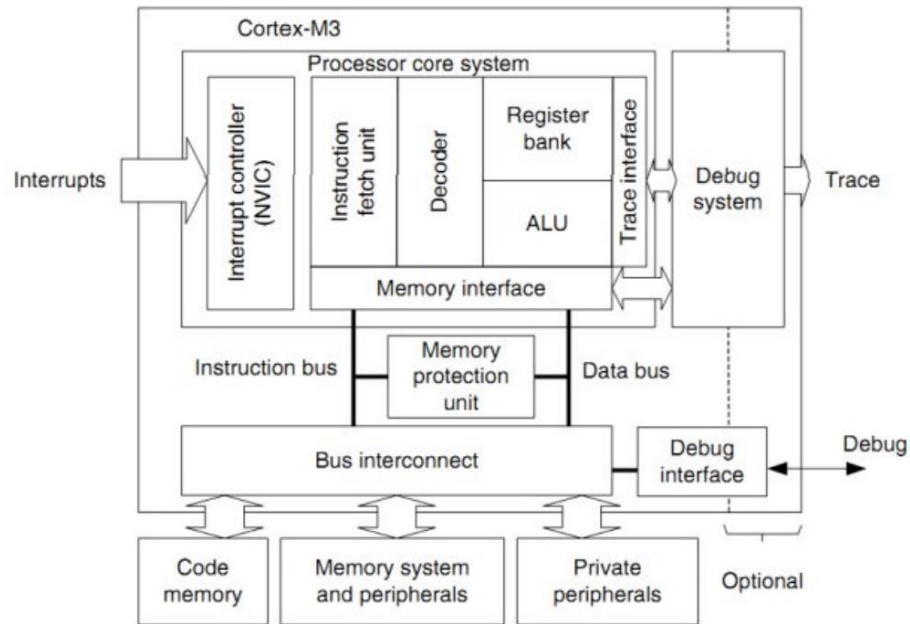
Chip dựa vào vi xử lý Cortex-M3 từ các nhà sản xuất khác nhau sẽ có kích cỡ khác nhau về bộ nhớ, khác nhau về chủng loại, khác nhau về thiết bị ngoại vi, và khác nhau về các tính năng



Hình 1- 2. Cấu trúc vi điều khiển ARM dùng lõi Cortex-M3

1.2.4 Cấu trúc ARM

Cortex-M3 là vi xử lý 32-bit, có đường dữ liệu 32-bit, bank thanh ghi 32-bit, và các giao diện bộ nhớ 32-bit - xem hình 1.3



Hình 1- 3. Cấu trúc cơ bản của ARM Cortex-M3

Bộ vi xử lý có kiến trúc Harvard: có bus để giao tiếp bộ nhớ chương trình và bộ nhớ dữ liệu độc lập. Điều này cho phép truy cập dữ liệu và lệnh diễn ra cùng một lúc nên hiệu suất của bộ vi xử lý tăng lên

Bộ vi xử lý Cortex-M3 có các thanh ghi từ R0 đến R15, xem hình 1.4.

Name	Functions (and banked registers)
R0	General-purpose register
R1	General-purpose register
R2	General-purpose register
R3	General-purpose register
R4	General-purpose register
R5	General-purpose register
R6	General-purpose register
R7	General-purpose register
R8	General-purpose register
R9	General-purpose register
R10	General-purpose register
R11	General-purpose register
R12	General-purpose register
R13 (MSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)
R13 (PSP)	
R14	Link Register (LR)
R15	Program Counter (PC)

Low registers

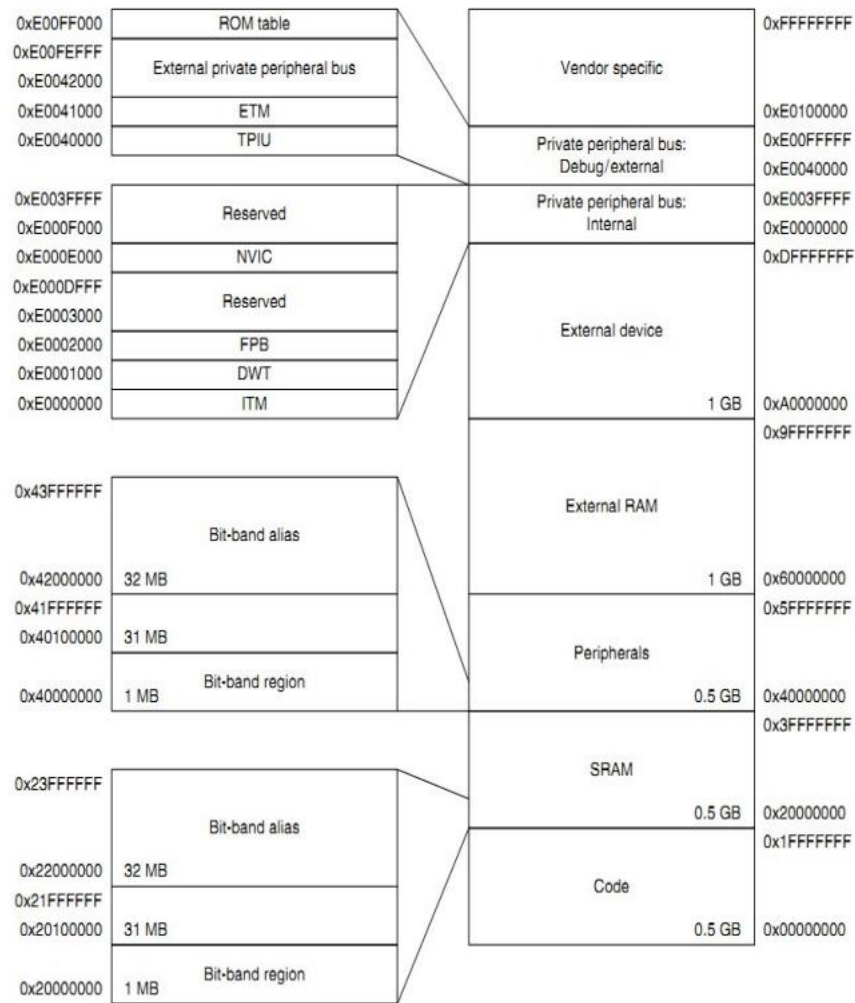
High registers

Hình 1- 4. Các thanh ghi của ARM Cortex-M

Bộ vi xử lý Cortex-M3 có bản đồ bộ nhớ cố định như hình 1.5.

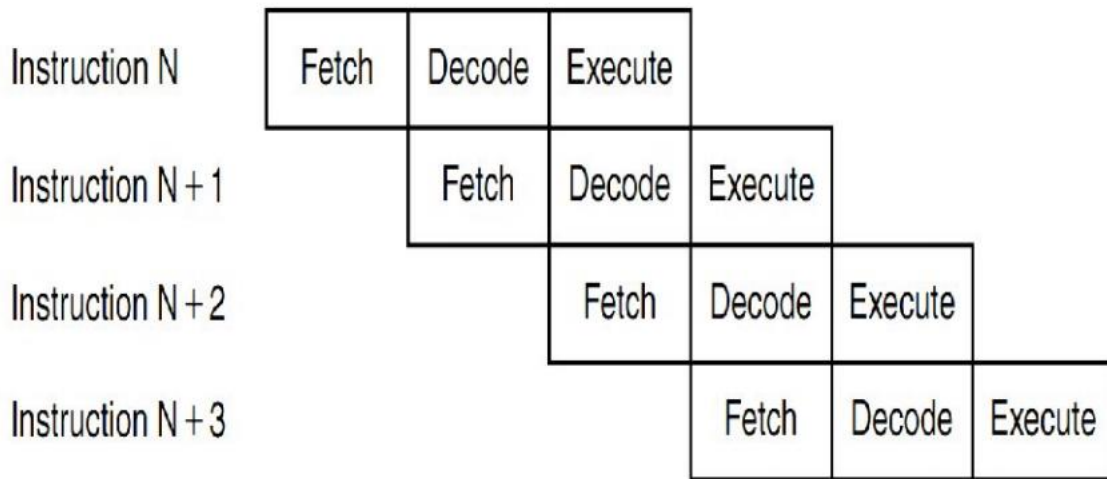
Vi xử lý Cortex-M3 có tổng cộng 4 GB không gian địa chỉ. Mã chương trình có thể được lưu trong vùng nhớ mã, lưu trong vùng nhớ SRAM, hoặc các vùng nhớ RAM bên ngoài. Tuy nhiên, tốt nhất là lưu mã chương trình trong vùng nhớ mã bởi vì với sự sắp xếp này thì đón mã lệnh và truy cập dữ liệu được thực hiện đồng thời trên ở hai giao diện bus riêng biệt

Vùng nhớ SRAM là bộ nhớ nội. Truy cập vào vùng nhớ SRAM này được thực hiện thông qua các bus giao diện hệ thống. Trong vùng nhớ này, một phạm vi 34-MB được định nghĩa là vùng nhớ bit-band. Trong vùng nhớ 34-bit-band, mỗi địa chỉ từ (32 bit) đại diện một 1 bit trong vùng 1 MB bit-band



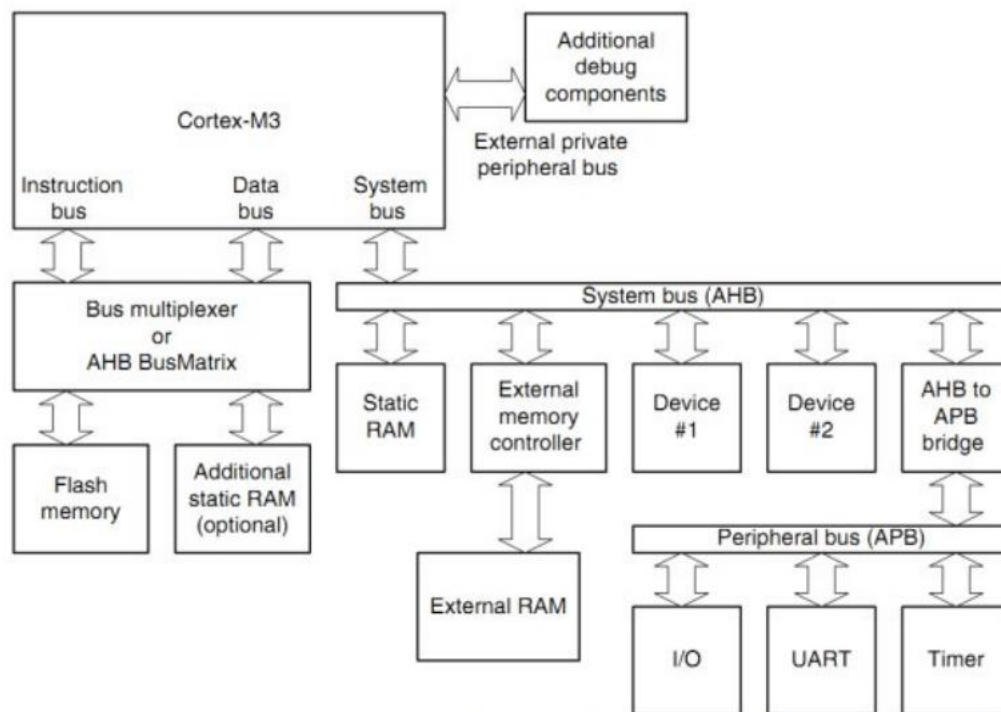
Hình 1- 5. Bản đồ bộ nhớ của Cortex –M3.

Bộ vi xử lý Cortex-M3 có cấu trúc pipeline ba tầng. Các tầng của luồng bao gồm: Đón lệnh, giải mã lệnh, và thực hiện lệnh như hình 1.6



Hình 1- 6. Luồng lệnh 3 tầng của Cortex-M

Hình 1.7 trình bày rõ các giao diện bus kết nối các thành phần với nhau:



Hình 1- 7. Sơ đồ khối chi tiết hệ thống vi xử lý Cortex-M3

Các nguồn dao động:

Có nhiều nguồn xung clock khác nhau được dùng để cấp nguồn xung clock hệ thống.

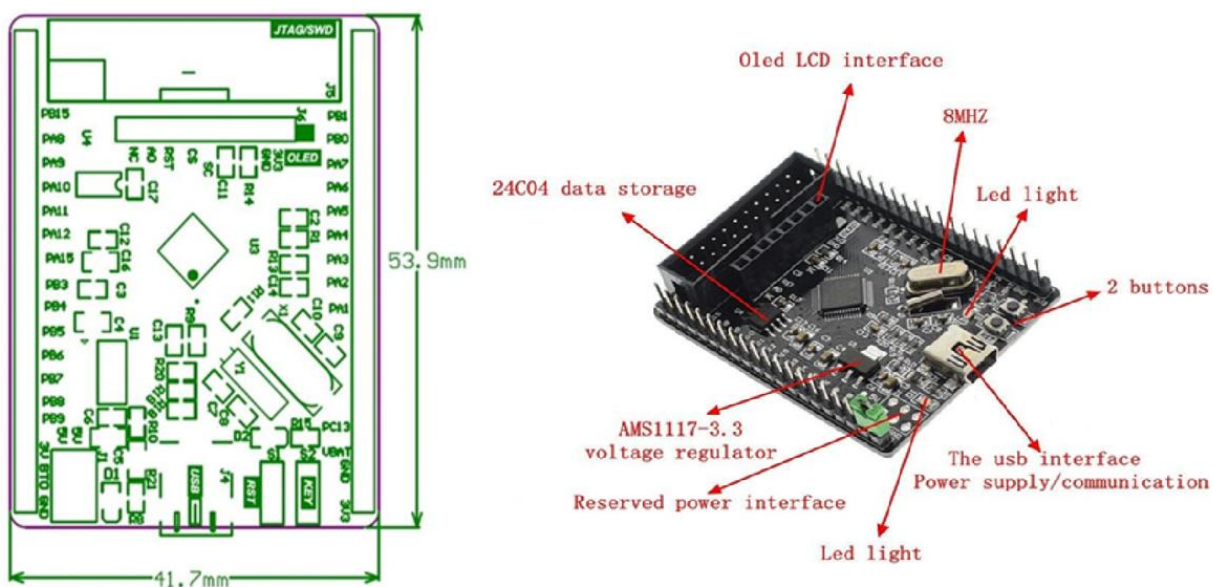
- Nguồn xung clock của bộ dao động HSI (HSI oscillator clock)
- Nguồn xung clock của bộ dao động HSE (HSE oscillator clock)
- Nguồn xung clock PLL

Các thiết bị có 2 nguồn xung clock phụ theo sau:

- Mạch dao động RC bên trong tần số thấp 40kHz (LSI RC) dùng để cấp cho bộ định thời giám sát độc lập và bộ RTC được dùng để tự động đánh thức CPU khỏi chế độ ngừng/chế độ chờ
- Mạch dao động thạch anh gắn thêm ở bên ngoài tần số thấp 32768Hz (LSE crystal) dùng để cấp cho bộ thời gian thực (Real time clock RTCCLK).

1.2.5 Các thành phần của STM32F103C8T6

KIT STM32F103C8T6 Smart V2 thuộc loại kit phát triển là Kit phát triển được thiết kế với đơn giản, kít ra đầy đủ chân của vi điều khiển, có cổng giao tiếp USB và cổng nạp SWD, sử dụng dòng vi điều khiển 32 Bit của dòng ST. Thích hợp với những người tiếp cận dòng STM 32 Bit



Hình 1- 8. Kit STM32F103C8T6 SmartV2

Thông số kỹ thuật

- Kit sử dụng vi điều khiển STM32F103C8T6
- Tốc độ: 72Mhz
- 48 chân
- Flash: 64KB
- Ram: 20KB
- chip SPI FLASH , W25X16 , dung lượng 2M byte
- Giao diện truyền không dây: 2.4G
- Hỗ trợ bộ nhớ rời: NOR/ NAND FLASH, SRAM, ...
- USB: 2.0, DMA
- Kết nối: SPI, USART, I2C, I2S, CAN...
- RTC độ chính xác cao, hỗ trợ lịch trong phần cứng.
- Chip có cảm biến nhiệt độ tích hợp.
- Nhiều I/O hỗ trợ 5V
- LM1117-3.3V điện áp điều chip, cung cấp 800mA lớn nhất hiện nay

- Hỗ trợ đa dạng các OS nhúng như: uC/OS, STemWin, LWiP.
- Kích thước: 54x42 mm
- Kích thước lỗ: 3mm
- Trọng Lượng của Kit STM32F10C8T6 Smart V2: 35g

Sử dụng mạch nạp: ST-Link V2: ST-Link V2 là mạch nạp được sử dụng để nạp chương trình cho các vi điều khiển STM8 và STM32, mạch nạp có ưu điểm giá thành rẻ, chất lượng đảm bảo, ít bị lỗi. Kích thước nhỏ gọn.

Nạp theo chuẩn SWD:

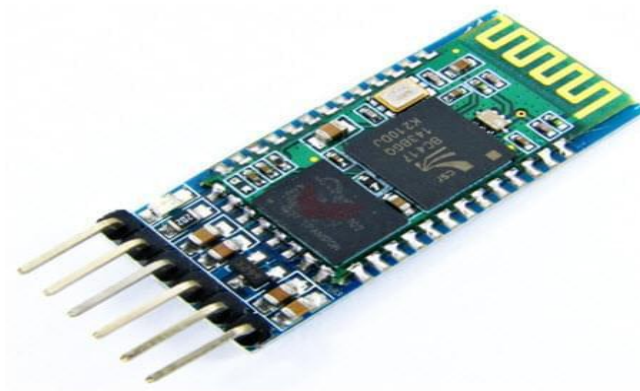
- SWIM-- CLK
- SWDIO-- DIO
- GND-- GND
- 3.3V-- 3.3V

1.3 Giới thiệu module bluetooth HC-05

Module thu phát Bluetooth HC-05 dùng để thiết lập kết nối Serial giữa 2 thiết bị bằng sóng bluetooth. Điểm đặc biệt của module bluetooth HC-05 là module có thể hoạt động được ở 2 chế độ: MASTER hoặc SLAVE. Trong khi đó, bluetooth module HC-06 chỉ hoạt động ở chế độ SLAVE.

- Ở chế độ SLAVE: bạn cần thiết lập kết nối từ smartphone, laptop, usb bluetooth để dò tìm module sau đó pair với mã PIN là 1234. Sau khi pair thành công, bạn đã có 1 cổng serial từ xa hoạt động ở baud rate 9600.
- Ở chế độ MASTER: module sẽ tự động dò tìm thiết bị bluetooth khác (1 module bluetooth HC-06, usb bluetooth, bluetooth của laptop...) và tiến hành pair chủ động mà không cần thiết lập gì từ máy tính hoặc smartphone.

Module Bluetooth thu phát HC-05 được thiết kế nhỏ gọn ra chân tín hiệu giao tiếp cơ bản và nút bấm để vào chế độ AT COMMAND, mạch được thiết kế để có thể cấp nguồn và giao tiếp qua 3.3VDC hoặc 5VDC, thích hợp cho nhiều ứng dụng khác nhau: Robot Bluetooth, điều khiển thiết bị qua Bluetooth,....



Hình 1- 9. Module bluetooth HC-05

Thông số module Bluetooth HC-05

- Điện áp hoạt động: 3.3 ~ 5VDC

- Mức điện áp chân giao tiếp: TTL tương thích 3.3VDC và 5VDC.
- Dòng điện khi hoạt động: khi Pairing 30 mA, sau khi pairing hoạt động truyền nhận bình thường 8 mA.
- Baudrate UART có thể chọn được: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
- Support profiles: Bluetooth serial port (master and slave)
- Bluetooth protocol: Bluetooth specification v2.0 + EDR
- Frequency: 2.4 GHz ISM band
- Modulation: GFSK (Gaussian frequency shift keying)
- Transmit power: =4 dBm, class 2
- Sensitivity: =-84 dBm at 0.1% BER
- Rate: Asynchronous: 2.1 Mbps (max.)/160 kbps
- Synchronous: 1 Mbps/1 Mbps
- Security features: authentication and encryption
- Kích thước: 15.2 x 35.7 x 5.6mm

Thiết lập mặc định:

- Thiết lập UART mặc định: Baudrate 9600, N, 8, 1.
- Pairing code mặc định: 1234 hoặc 0000.
- Để vào chế độ AT COMMAND, bấm và giữ nút trước khi cấp nguồn, LED sẽ nháy 2s. baud rate cho chế độ AT COMMAND là 38400. Chân Tx nối với chân Rx. Lưu ý các lệnh AT đều là chữ in hoa.
- Cấp nguồn và không nhấn nút sẽ chạy bình thường. LED sẽ nháy nhanh
- Chân EN chỉ nhận mức logic TTL 3V3. Không có chức năng chọn vào chế độ AT COMMAND

1.4 Giới thiệu Module HC-SR04, servo GS90, L298

1.4.1 Module HC-SR04

Cảm biến khoảng cách HC-SR04 là cảm biến dùng để xác định khoảng cách trong phạm vi nhỏ bằng cách phát sóng siêu âm. Cảm biến với độ chính xác khá cao (với khoảng cách nhận biết nhỏ nhất 3mm) và độ ổn định cao trong quá trình sử dụng, đồng thời dễ dàng kết nối với các MCU (Arduino, DSP, AVR, PIC, ARM...)

Các ứng dụng thường thấy với cảm biến như: xe tránh vật cản, xác định khoảng cách của vật thể đến cảm biến, hay mô phỏng làm radar...



Hình 1- 10. Module HC-SR04

Thông số kỹ thuật

- Điện áp làm việc: 5V (DC)
- Góc quét tốt nhất: 30°
- Khoảng cách hoạt động: 2cm ~ 450cm
- Khoảng cách nhận biết nhỏ nhất: 3mm
- Pinout:
 - VCC
 - Trig(T)

- Echo®
- GND

1.4.2 Servo GS90

Servo là một dạng động cơ điện đặc biệt. Không giống như động cơ thông thường cứ cắm điện vào là quay liên tục, servo chỉ quay khi được điều khiển (bằng xung PPM) với góc quay nằm trong khoảng bất kì từ 0o - 180o. Mỗi loại servo có kích thước, khối lượng và cấu tạo khác nhau. Có loại thì nặng chỉ 9g (chủ yếu dùng trên máy bay mô hình), có loại thì sở hữu một momen lực bá đạo (vài chục Newton/m), hoặc có loại thì khỏe và không sắc chắc chắn,...

Động cơ servo được thiết kế những hệ thống hồi tiếp vòng kín. Tín hiệu ra của động cơ được nối với một mạch điều khiển. Khi động cơ quay, vận tốc và vị trí sẽ được hồi tiếp về mạch điều khiển này. Nếu có bất kỳ lý do nào ngăn cản chuyển động quay của động cơ, cơ cấu hồi tiếp sẽ nhận thấy tín hiệu ra chưa đạt được vị trí mong muốn. Mạch điều khiển tiếp tục chỉnh sai lệch cho động cơ đạt được điểm chính xác. Các động cơ servo điều khiển bằng liên lạc vô tuyến được gọi là động cơ servo RC (radio-controlled). Trong thực tế, bản thân động cơ servo không phải được điều khiển bằng vô tuyến, nó chỉ nối với máy thu vô tuyến trên máy bay hay xe hơi. Động cơ servo nhận tín hiệu từ máy thu này.

Động Cơ Servo SG90 là loại động cơ được dùng phổ biến trong các mô hình điều khiển nhỏ và đơn giản như cánh tay robot. Động cơ có tốc độ phản ứng nhanh, được tích hợp sẵn Driver điều khiển động cơ, dễ dàng điều khiển góc quay bằng phương pháp điều độ rộng xung PWM.



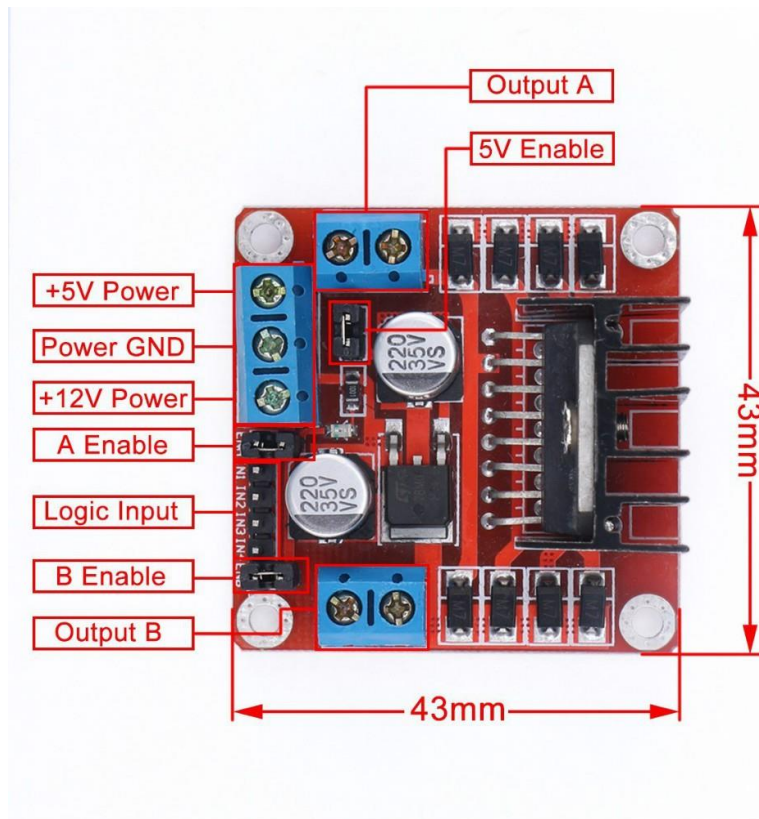
Hình 1- 11. Động cơ Servo GS90

Thông số

- Khối lượng : 9g
- Kích thước: 23mmX12.2mmX29mm
- Momen xoắn: 1.8kg/cm
- Tốc độ hoạt động: 60 độ trong 0.1 giây
- Điện áp hoạt động: 4.8V(~5V)
- Nhiệt độ hoạt động: 0 °C – 55 °C

1.4.3 Module điều khiển động cơ L298

IC L298 là một IC tích hợp nguyên khối gồm 2 mạch cầu H bên trong. Với điện áp làm tăng công suất đầu ra từ 5V – 47V , dòng lên đến 4A, L298 rất thích hợp trong những ứng dụng công suất nhỏ như động cơ DC loại vừa ...



Hình 1- 12. Module L298

Tóm tắt qua chức năng các chân của L298:

- 4 chân INPUT: **IN1, IN2, IN3, IN4** được nối lần lượt với các chân tương ứng của L298. Đây là các chân nhận tín hiệu điều khiển.
- 4 chân OUTPUT: **OUT1, OUT2, OUT3, OUT4** (tương ứng với các chân INPUT) được nối với các chân tương ứng của L298. Các chân này sẽ được nối với động cơ.
- Hai chân **ENA** và **ENB** dùng để điều khiển các mạch cầu H trong L298. Nếu ở mức logic “1” (nối với nguồn 5V) thì cho phép mạch cầu H hoạt động, nếu ở mức logic “0” thì mạch cầu H không hoạt động.

Điều khiển chiều quay với L298:

- Khi $ENA = 0$: Động cơ không quay với mọi đầu vào .
- Khi $ENA = 1$:

- $INT1 = 1; INT2 = 0$: động cơ quay thuận.
- $INT1 = 0; INT2 = 1$: động cơ quay nghịch.
- $INT1 = INT2$: động cơ dừng ngay tức thì.

(tương tự với các chân ENB, INT3, INT4).

1.5 Giới thiệu GPIO

GPIO là từ viết tắt của General purpose I/O ports tạm hiểu là nơi giao tiếp chung giữa tín hiệu ra và tín hiệu vào. GPIO là bài cơ bản, cần nắm vững khi học bất kỳ một VĐK nào đó. Cần hiểu được các thuật ngữ, chế độ, cấu hình, số lượng... của các chân GPIO. Ở STM32 thì các chân GPIO chia ra làm nhiều Port vd: PortA, PortB.... Số lượng Port phụ thuộc vào số lượng chân(pin) và cách gọi phụ thuộc vào nhà sản xuất(ví dụ VĐK X có PortA mà lại không có PortD). Mỗi Port thường có 16 chân đánh số từ 0 -> 15 tương ứng với mỗi chân là 1bit. Mỗi chân có 1 chức năng khác nhau như analog input, external interrupt. hay đơn thuần chỉ là xuất tín hiệu on/off ở mức 0,1. Chức năng của mỗi chân thì chúng ta cần tra datasheet của nhà sản xuất trước khi lập trình hoặc thiết kế mạch

Các tính mode GPIO của STM32

Input floating : cấu hình chân I/O là ngõ vào và để nổi.

Input pull-up : cấu hình chân I/O là ngõ vào, có trở kéo lên nguồn.

Input-pull-down: cấu hình chân I/O là ngõ vào, có trở kéo xuống GND.

Analog : cấu hình chân I/O là Analog, dùng cho các mode có sử dụng ADC hoặc DAC.

Output open-drain: cấu hình chân I/O là ngõ ra, khi output control = 0 thì N-MOS sẽ dẫn, chân I/O sẽ nối VSS, còn khi output control = 1 thì P-MOS và N-MOS đều không dẫn, chân I/O được để nổi.

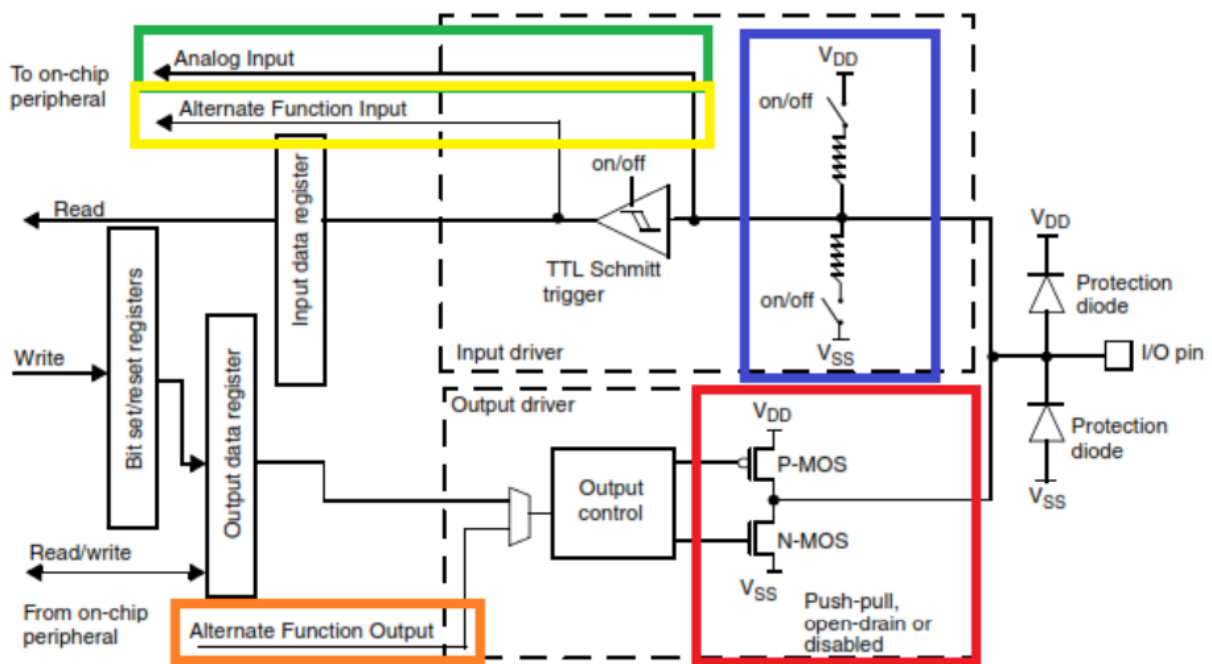
Output push-pull: cấu hình chân I/O là ngõ ra, khi output control = 0 thì N-MOS sẽ dẫn, chân I/O sẽ nối VSS, còn khi output control = 1 thì P-MOS dẫn, chân I/O được nối VDD.

Alternate function push-pull : sử dụng chân I/O vừa là ngõ ra và vừa là ngõ vào, tuy nhiên sẽ không có trở kéo lên và kéo xuống ở input, chức năng

output giống Output push-pull. Ngoài ra nó còn để sử dụng cho chức năng remap.

Alternate function push-pull : : sử dụng chân I/O vừa là ngõ ra và vừa là ngõ vào, tuy nhiên sẽ không có trở kéo lên và kéo xuống ở input, chức năng output giống Output open-drain. Ngoài ra nó còn để sử dụng cho chức năng remap.

Cấu trúc 1 chân GPIO của chip:



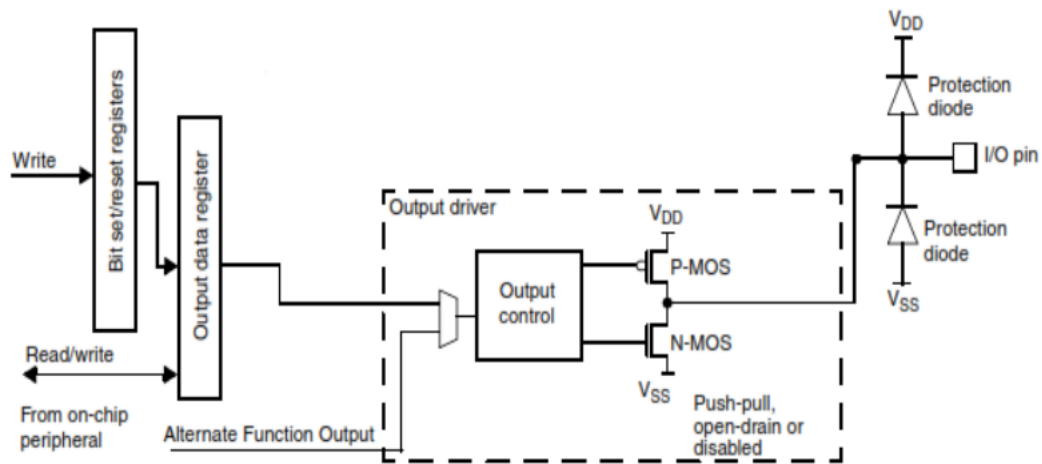
Hình 1- 13. Cấu hình chân GPIO

Không giống như những dòng vi điều khiển 8 bit, chỉ có 8 chân IO trên 1 port thì ở các vi điều khiển 32bit, có đến 16 chân IO trên 1 port.

Đối với chip STM32F103C8Tx gồm có 3 Port chính đó là GPIOA, GPIOB, GPIOC.

Không phải tất cả các port đều có 16 chân, chỉ riêng GPIOA, GPIOB trên kit thì có đủ 16 chân GPIO.

Chức năng của GPIO



Hình 1- 14. Chức năng của GPIO

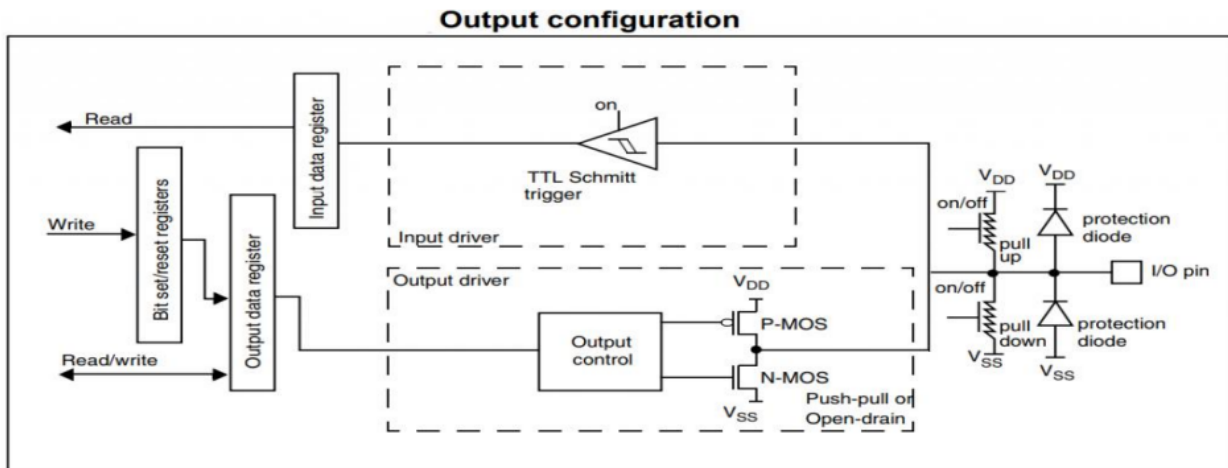
Các thanh ghi dữ liệu

GPIOx_CRx - Port configuration register: Mỗi pin có 2 cặp bit để cài đặt. CNF quyết định là mode analog, floating, pull-up... còn MODE quyết định là input hay output. Output với tốc độ bao nhiêu.

GPIOx_IDR - Port input data register: Đây là thanh ghi đọc giá trị đầu vào của từng chân. Khi đầu vào ở chân nào đó có mức logic là 1 thì bit tương ứng với chân đó sẽ có giá trị là 1. Và ngược lại là 0.

GPIOx_ODR - Port output data register: Đây là thanh ghi quyết định mức logic của đầu ra trên chân STM32 tương ứng với mode output. Giá trị của bit nào ở mức cao thì output của nó sẽ ở mức cao nếu sử dụng mode output push pull và ngược lại. Riêng mode output open drain thì có chút khác biệt

Cách xuất tín hiệu output thông qua khối CMOS



Hình 1- 15. Cấu hình đầu ra của GPIO

Khi một I/O Pin được cấu hình hoạt động với chức năng OUTPUT thì: Khối điều khiển OUTPUT được sử dụng với các chế độ: Open drain mode hoặc Push-pull mode.

Với chế độ Open drain: Một giá trị bit bằng “0” ở thanh ghi ODR sẽ làm NMOS dẫn, P-MOS ngưng dẫn, lúc này chân vi điều khiển có mức logic 0 (được nối với GND); Một giá trị bit bằng “1” ở thanh ghi ORD sẽ làm ngưng dẫn cả N-MOS và P-MOS, chân tương ứng sẽ ở trạng thái Hi-Z (trở kháng cao).

Với chế độ Push-pull: Một giá trị bit bằng 0 ở thanh ghi ODR sẽ làm N-MOS dẫn và P-MOS ngưng dẫn, lúc này chân vi điều khiển có mức logic 0 (được nối với GND); Một giá trị bit bằng “1” ở thanh ghi ODR sẽ làm N-MOS ngưng dẫn và PMOS dẫn. Lúc này chân vi điều khiển có mức logic 1 (được nối với VDD).

Như vậy, để điều khiển giá trị logic của một pin được cấu hình hoạt động với chức năng OUTPUT thì chúng ta cần ghi giá trị logic vào thanh ghi Output Data (GPIOx_ODR). Bit tương ứng của thanh ghi sẽ điều khiển pin ở vị trí

tương ứng. Ví dụ: BIT thứ 0 của thanh ghi GPIOA_ODR sẽ điều khiển pin tương ứng là PA0

1.6 Giới thiệu UART

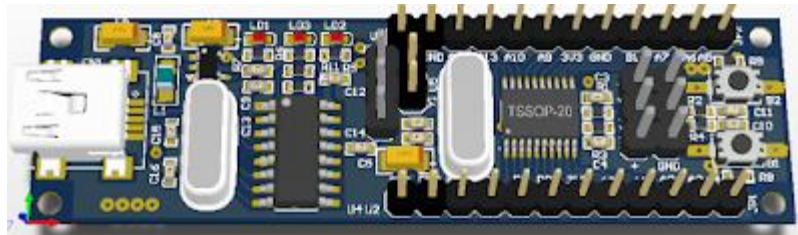
UART - Universal synchronous asynchronous receiver transmitter là một ngoại vi cơ bản và thường dùng trong các quá trình giao tiếp với các module như : Xbee, Wifi, Bluetooth.... Khi giao tiếp UART kết hợp với các IC giao tiếp như MAX232CP, SP485EEN.... thì sẽ tạo thành các chuẩn giao tiếp RS232, RS485. Đây là các chuẩn giao tiếp thông dụng và phổ biến trong công nghiệp từ trước đến nay.

Khi ta sử dụng chân UART_CLK thì giao tiếp UART sẽ trở thành giao tiếp đồng bộ và không dùng sẽ là chuẩn giao tiếp không đồng bộ. Các bạn để ý là với bất cứ 1 chuẩn truyền thông nào, khi có sử dụng 1 chân tín hiệu làm chân CLK thì chuẩn giao tiếp đó sẽ là chuẩn giao tiếp đồng bộ và ngược lại. Ở đây mình chỉ đề cập đến giao tiếp UART không đồng bộ.

Ưu điểm của giao tiếp UART không đồng bộ: tiết kiệm chân vi điều khiển(2 chân), là ngoại vi mà bất kì 1 VĐK nào cũng có, có khá nhiều module, cảm biến dùng UART để truyền nhận data với VĐK. Nhược điểm của loại ngoại vi này là tốc độ khá chậm, tốc độ tối đa tùy thuộc vào từng dòng; quá trình truyền nhận dễ xảy ra lỗi nên trong quá trình truyền nhận cần có các phương pháp để kiểm tra(thông thường là truyền thêm bit hoặc byte kiểm tra lỗi). UART không phải là 1 chuẩn truyền thông, Khi muốn nó là 1 chuẩn truyền thông hoặc truyền data đi xa, chúng ta cần phải sử dụng các IC thông dụng để tạo thành các chuẩn giao tiếp đáng tin cậy như RS485 hay RS232....

Thông thường chúng ta sẽ dùng ngắt nhận UART để nhận dữ liệu vì sử dụng ngắt sẽ tiện lợi, không tốn thời gian chờ cũng như mất dữ liệu. Các tốc độ thường dùng để giao tiếp với máy tính 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200.

Một số phần mềm giao tiếp với máy tính: hercules_3-2-5, teraterm, Serial-Oscilloscope-v1.5... Một số module dùng để giao tiếp với máy tính: CP2102 USB 2.0, USB ra UART dùng PL2303, USB to UART dùng TTL FT232RL, USB ra UART dùng CH340G...



Hình 1- 16. UART trong STM32F103

STM32F103C8 có 3 bộ UART với nhiều mode hoạt động, với nhiều bộ UART ta có thể sử dụng được nhiều ứng dụng với 1 chip điều khiển so với STM8S. Một số tính năng nổi bật như sau:

- Đầy đủ các tính năng của bộ giao tiếp không đồng bộ.
- Điều chỉnh baud rate bằng lập trình và tốc độ tối đa lên đến 4.5Mb/s.
- Độ dài được lập trình là 8 hoặc 9 bit.
- Cấu hình bit stop hỗ trợ là 1 hoặc 2.
- Có chân clock nếu muốn chuyển giao tiếp thành đồng bộ.
- Cấu hình sử dụng 1 dây hoặc 2 dây.
- Có bộ DMA nếu muốn đẩy cao thời gian truyền nhận.
- Bit cho phép truyền nhận riêng biệt.

Cách hoạt động của giao thức UART:

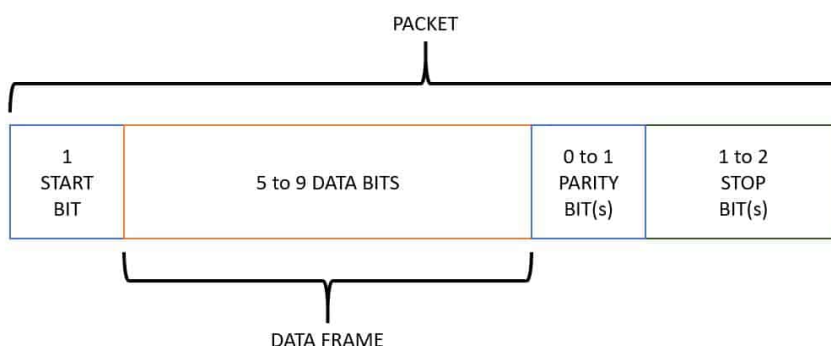
UART là giao thức truyền thông không đồng bộ, nghĩa là không có xung Clock, các thiết bị có thể hiểu được nhau nếu các Setting giống nhau

UART là truyền thông song công(Full duplex) nghĩa là tại một thời điểm có thể truyền và nhận đồng thời.

Trong đó quan trọng nhất là Baund rate (tốc độ Baund) là khoảng thời gian dành cho 1 bit được truyền. Phải được cài đặt giống nhau ở gửi và nhận.

Sau đó là định dạng gói tin.

Định dạng gói tin như sau:



Hình 1- 17. Định dạng gói tin của UART

Start Bit	Start-bit (bit đồng bộ hóa) được đặt trước dữ liệu thực tế. Nói chung, một đường truyền dữ liệu không hoạt động được điều khiển ở mức điện áp cao. Để bắt đầu truyền dữ liệu, truyền UART kéo đường dữ liệu từ mức điện áp cao (1) xuống mức điện áp thấp (0). UART thu được thông báo sự chuyển đổi này từ mức cao sang mức thấp qua đường dữ liệu cũng như bắt đầu hiểu dữ liệu thực. Nói chung, chỉ có một start-bit.
Stop Bit	Bit dừng được đặt ở phần cuối của gói dữ liệu. Thông thường, bit này dài 2 bit nhưng thường chỉ sử dụng 1 bit. Để dừng sóng, UART giữ đường dữ liệu ở mức điện áp cao.
Partity Bit	Bit chẵn lẻ cho phép người nhận đảm bảo liệu dữ liệu được thu thập có đúng hay không. Đây là một hệ thống kiểm tra lỗi cấp

	thấp & bit chẵn lẻ có sẵn trong hai phạm vi như Chẵn lẻ – chẵn lẻ cũng như Chẵn lẻ – lẻ. Trên thực tế, bit này không được sử dụng rộng rãi nên không bắt buộc.
Data frame	Các bit dữ liệu bao gồm dữ liệu thực được truyền từ người gửi đến người nhận. Độ dài khung dữ liệu có thể nằm trong khoảng 5 & 8. Nếu bit chẵn lẻ không được sử dụng thì chiều dài khung dữ liệu có thể dài 9 bit. Nói chung, LSB của dữ liệu được truyền trước tiên sau đó nó rất hữu ích cho việc truyền.

Table 1- 1. Định dạng các gói tin trong UART

1.7 Giới thiệu Timer, PWM

Timer trong STM32 có rất nhiều chức năng chẳng hạn như bộ đếm counter, PWM, inputcapture ngoài ra còn một số chức năng đặc biệt để điều khiển động cơ như encoder, hallsensors.

Timer là bộ định thời có thể sử dụng để tạo ra thời gian cơ bản dựa trên các thông số: clock, prescaler, autoreload, repetition counter. Timer của STM32 là timer 16 bits có thể tạo ra các sự kiện trong khoảng thời gian từ nano giây tới vài phút gọi là UEV(update event)

Các thành phần cơ bản của 1 timer

- TIM_CLK: clock cung cấp cho timer.
- PSC (prescaler): là thanh ghi 16bits làm bộ chia cho timer, có thể chia từ 1 tới 65535
- ARR (auto-reload register): là giá trị đếm của timer (16bits hoặc 32bits).
- RCR (repetition counter register): giá trị đếm lặp lại 16bits. Giá trị UEV được tính theo công thức sau:

$$UEV = TIM_CLK / ((PSC + 1) * (ARR + 1) * (RCR + 1))$$

Ví dụ: TIM_CLK = 72MHz, PSC = 1, ARR = 65535, RCR = 0. UEV = $72000000 / ((1 + 1) * (65535 + 1) * (1)) = 549.3 \text{ Hz}$

- Với giá trị ARR (auto-reload) được cung cấp thì bộ định thời (timer) thực hiện các chế độ đếm khác nhau: đếm lên, đếm xuống hoặc kết hợp cả 2. Khi thực hiện đếm lên thì giá trị bộ đếm bắt đầu từ 0 và đếm tới giá trị ARR-1 thì báo tràn. Khi thực hiện đếm xuống thì bộ đếm bắt đầu từ giá trị ARR đếm xuống 1 thì báo tràn.

Các chế độ hoạt động của Timer

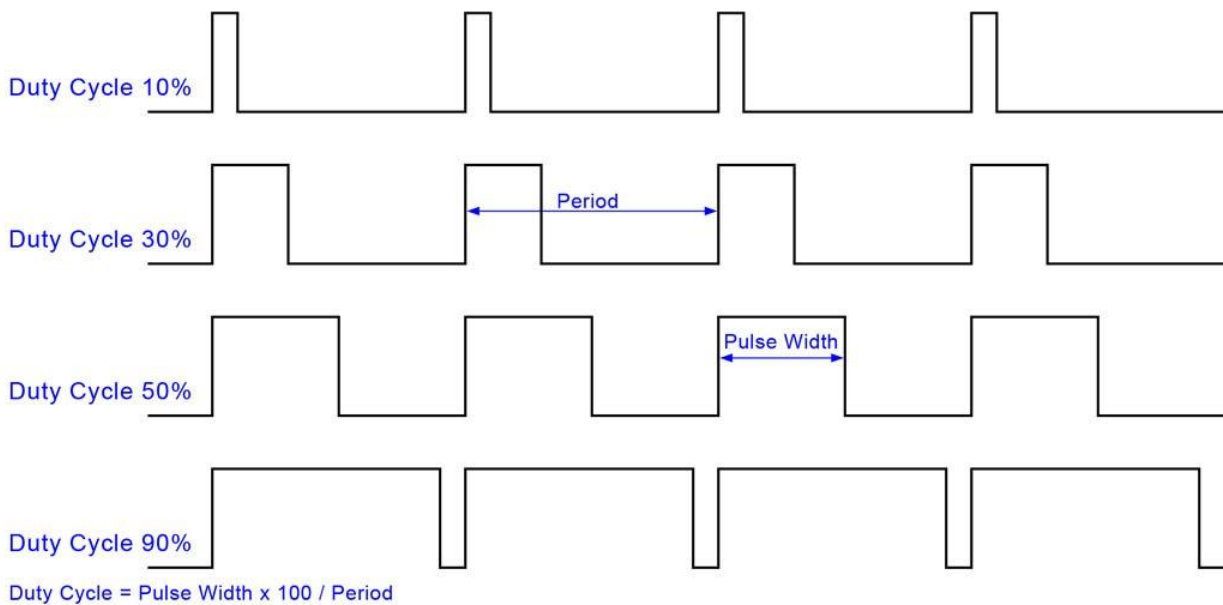
- Input capture
- Output compare
- PWM generation (Edge and Centeraligned modes)
- Onepulse mode output

Tổng quan về PWM

PWM(pulse- with modulation) hay còn gọi nôm na là “bấm xung” hay “điều khiển độ rộng xung” là ứng dụng phổ biến và thường dùng trong lĩnh vực điều khiển động cơ. Một ví dụ đơn giản để hiểu về độ rộng xung là mức độ sáng tắt của LED, ở tần số mà mắt người có thể nhìn thấy thì nó là độ chênh lệch giữa mức sáng và tắt của 1 đèn khoảng thời gian lặp đi lặp lại(vd đèn sáng 5s, tắt 3s lặp đi lặp lại thì chu kì sẽ là 8 giây); với tần số cao mắt người không thể nhìn thấy thì ta sẽ thấy LED sáng mờ hay sáng rõ đó là do tổng thời gian sáng/ tổng thời gian tắt trong khoảng thời gian lớn hay nhỏ mà mắt người nhìn thấy(tính bằng đơn vị nhỏ như ms)

Các khái niệm cần nắm

- Duty cycle : tỷ lệ phần trăm xung ở mức cao.
- Period : là chu kì xung(bao gồm tổng thời gian mức cao + mức thấp).
- Pulse width là giá trị của mức cao so với period.
- PTO là xung vuông có 50% thời gian cao, 50% thời gian thấp.
- Biên độ xung: là giá trị điện áp của xung khi ở mức cao.
- Các khái niệm về tần số, chu kì.



Hình 1- 18. Sóng PWM

- Nếu Duty Cycle = 0% tương ứng điện áp đầu ra $U=5 \times 0\% = 0V$
- Nếu Duty Cycle = 30% tương ứng điện áp đầu ra $U=5 \times 30\% = 1.5V$
- Nếu duty cycle = 50% tương ứng điện áp đầu ra $U=5 \times 50\% = 2.5V$
- Nếu duty cycle = 100% tương ứng điện áp đầu ra $U=5 \times 100\% = 5V$

Ứng dụng của PWM: sử dụng để điều khiển băm xung, điều khiển tốc độ động cơ, các bộ điều áp, xung áp,...

PWM trong stm32f103

- Chức năng PWM của chip STM32 nằm trong khối Timer.
- Vi điều khiển STM32F1 có 3 kênh Timer có thể dùng để xuất xung PWM gồm TIM2, TIM3, TIM4, trong đó mỗi kênh còn gồm 4 Channel nhỏ.

Timer	Channel	Pin
TIM2	Channel 1	PA0
	Channel 2	PA1
	Channel 3	PA2
	Channel 4	PA3
TIM3	Channel 1	PA6
	Channel 2	PA7
	Channel 3	PA8
	Channel 4	PB1
TIM4	Channel 1	PB6
	Channel 2	PB7
	Channel 3	PB8
	Channel 4	PB9

Hình 1- 19. Kênh Timer để xuất xung PWM

1.8 Giới thiệu về IDE

1.8.1 Giới thiệu về STM32CubeMX

STM32CubeMX là một phần mềm được cung cấp miễn phí giúp ích cho việc cấu hình ngoại vi, clock, tính toán dòng tiêu thụ, tạo project với nhiều dòng chip ARM STM32... Việc tạo project trở nên đơn giản bằng việc lựa chọn các ngoại vi cần thiết, cấp lock tùy chỉnh mà không cần liên quan đến code.

Việc tạo project với thư viện chuẩn (standard library) là khá khó khăn vì cần nhiều bước để tạo ra project mới. STM32CubeMX ra đời như một lựa chọn để thay thế điều đó, với giao diện trực quan chúng ta sẽ dễ dàng lập trình và có cái nhìn tổng quan hơn.

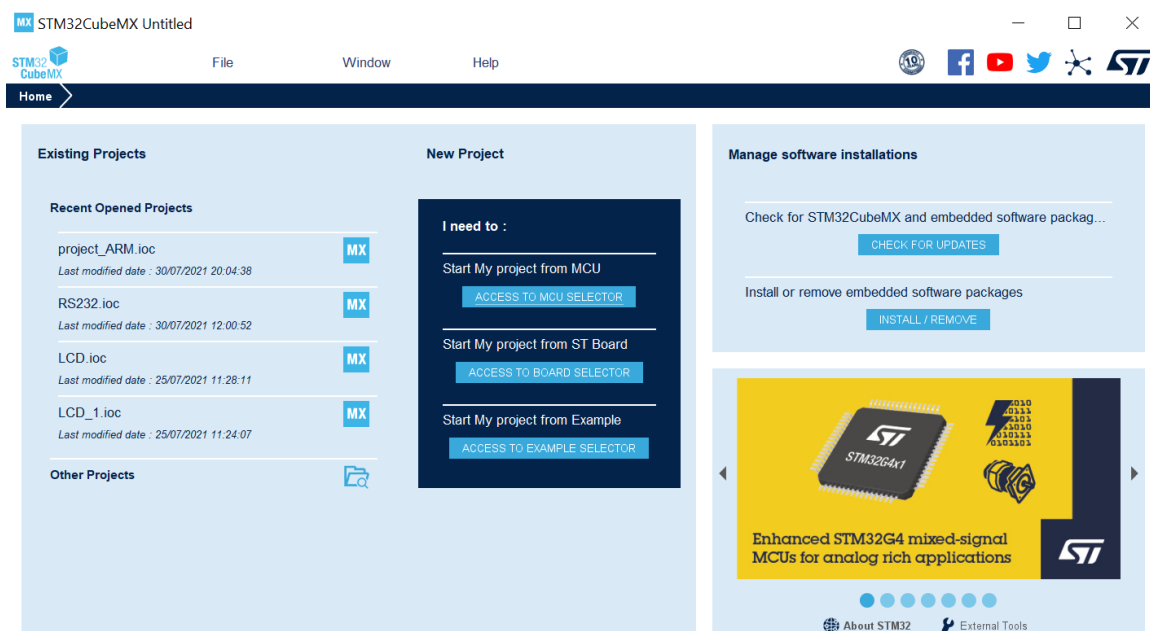
Thư viện đi kèm với phần mềm này là STM32Cube HAL, gọi tắt là thư viện HAL, bộ thư viện này được chuẩn hóa, giúp đồng nhất giữa các dòng F0, F1, F2,

F3, F4... Nhà sản xuất cũng cung cấp đầy đủ các ví dụ mẫu đi kèm với thư viện này. Cách tốt nhất để tự học là tự tìm hiểu các ví dụ mẫu này

STM32CubeMX không phải là một trình biên dịch, nó chỉ là công cụ để sinh ra code, chúng ta vẫn phải dùng các trình biên dịch thông thường để quan sát, debug, lập trình, sửa lỗi.... Nếu bạn muốn tìm hiểu sâu về vi điều khiển hãy bắt đầu nó bằng thư viện chuẩn (standard library) và ngược lại, nếu bạn muốn sử dụng một cách nhanh chóng hãy sử dụng thư viện HAL (STM32Cube HAL)

Một số tính năng chính:

- Dễ dàng quan sát các tính năng, thông số của IC.
- Có cái nhìn tổng quan, bao quát hơn về lựa chọn, cấu hình chân, clock, ngoại vi.
- Lựa chọn các board thí nghiệm của ST với các thông số cài đặt trước hoặc một MCU cụ thể.
- Dễ dàng cấu hình ngoại vi, clock, các thông số và tự động sinh ra code.
- Dễ dàng chuyển đổi giữa các loại MCU STM32 bằng cách tạo project mới một cách nhanh chóng.
- Dễ dàng xuất file PDF các chân, tính năng... đã cấu hình.
- Dễ dàng tạo ra project với nhiều trình biên dịch được lựa chọn.
- Tính toán, quản lý năng lượng dễ dàng hơn.
- Tự động cập nhật thư viện khi có bản cài đặt mới.
- Thư viện được đồng nhất nên sẽ không có quá nhiều thay đổi khi chuyển đổi giữa các loại MCU. Ví dụ chuyển từ F1 qua F3, hay F4 và ngược lại.

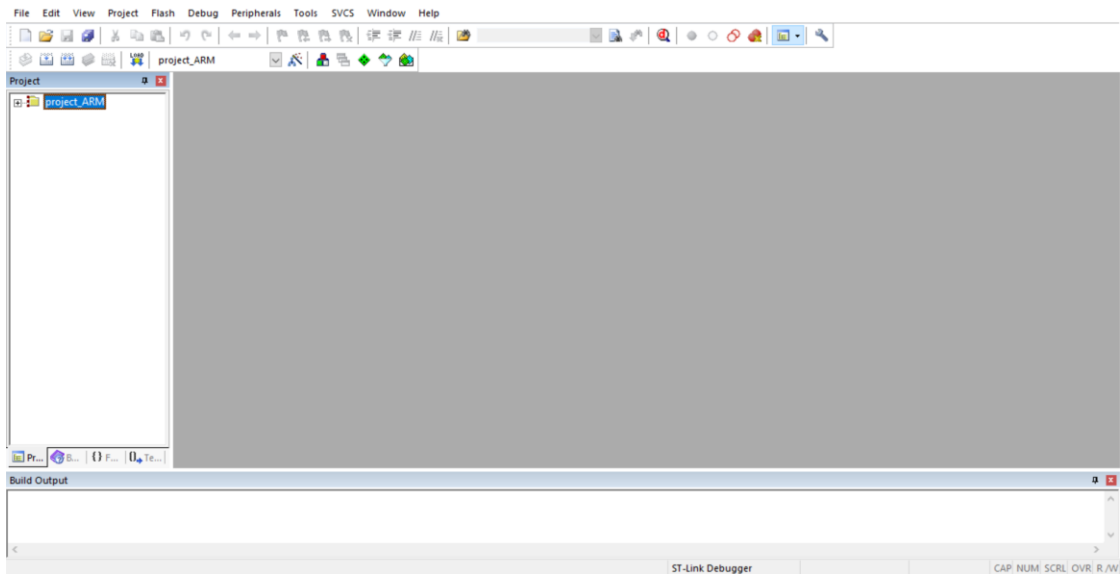


Hình 1- 20. Giao diện phần mềm STM32CubeM

1.8.2 Giới thiệu về Keil uVision 5

Sau khi STM32CubeMX cập nhật và khởi tạo mã nguồn. Keil uVision 5 được khởi tạo để chỉnh sửa và build mã nguồn

Trước tiên, giới thiệu qua về Keil uVision 5: Keil c uvision 5 là một phần mềm hỗ trợ cho người dùng trong việc lập trình cho vi điều khiển các dòng khác nhau (Atmel, AVR,..). Keil C giúp người dùng soạn thảo và biên dịch chương trình C hay cả ASM thành ngôn ngữ máy để nạp vào vi điều khiển giúp chúng ta tương tác giữa vi điều khiển và người lập trình.



Hình 1- 21. Giao diện Keil uVision 5

1.9 Quy trình tạo 1 Project

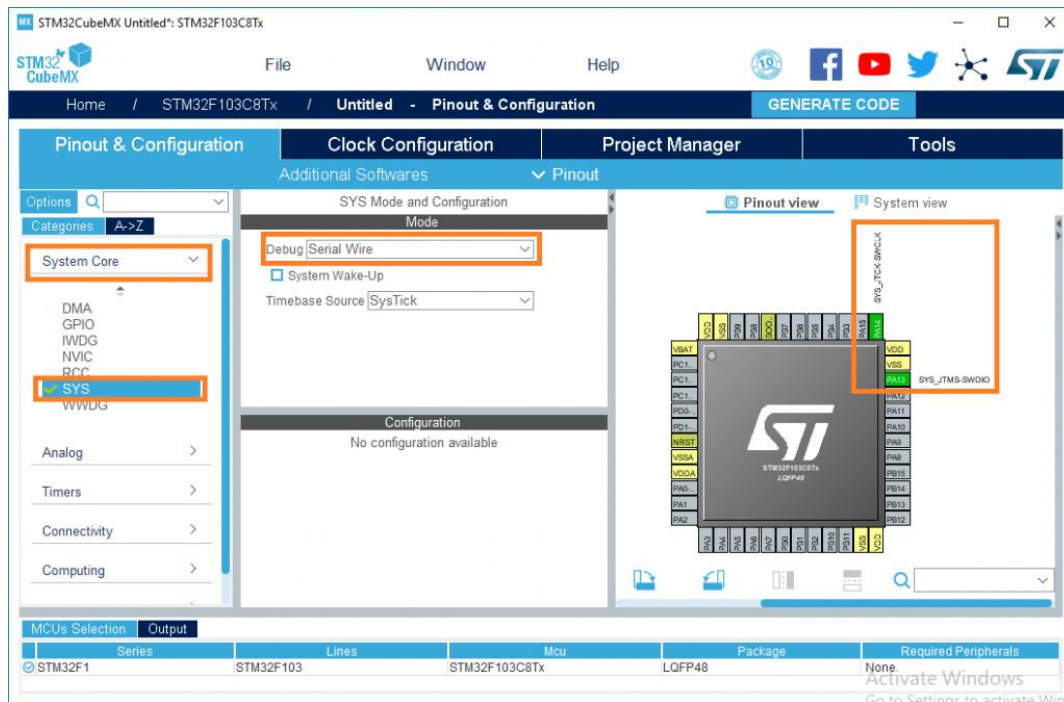
Sau khi nhấn vào File -> New Project thì giao diện chọn vi điều khiển STM32 sẽ hiện ra:

Chọn vi điều khiển: tại mục Part Number Search các bạn nhập vào tên vi điều khiển mà mình muốn cấu hình

Bắt đầu Project: nhấn vào Start Project

1.9.1 Tại mục Pinout & Configuration

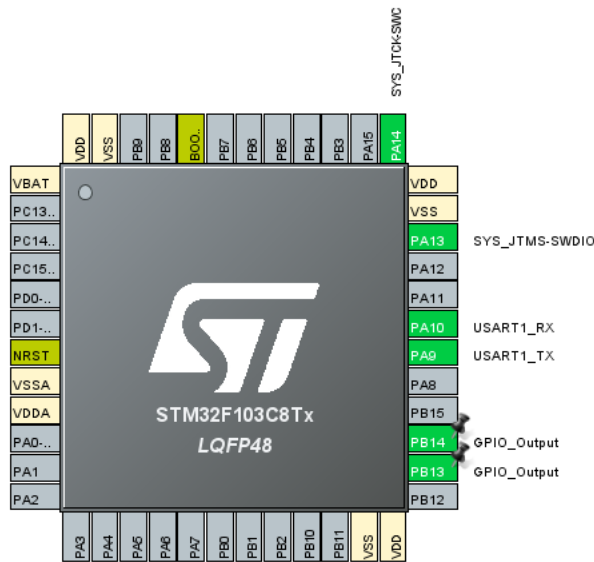
Cấu hình nạp code: Chọn System Core -> SYS -> Debug: Serial Wire để vi điều khiển được cấu hình nạp code thông qua chân SWDIO và SWCLK (sử dụng mạch nạp ST- Link V2 và kết nối với vi điều khiển STM32F103C8T6 thông qua các chân này



Hình 1- 24. Cấu hình nạp code

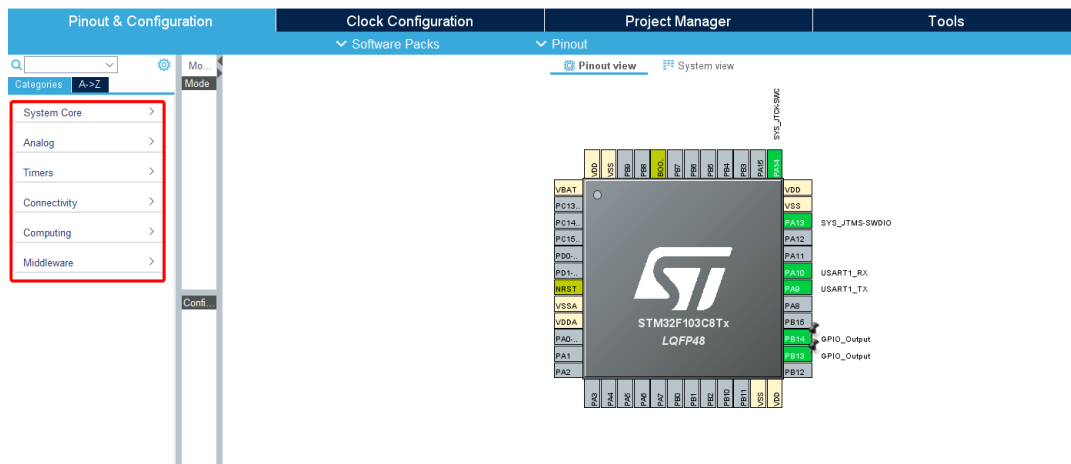
1.9.2 Cấu hình các ngoại vi

Việc cấu hình các ngoại vi như INPUT, INPUT, External Interrupt, ADC, TIMER, UART... có thể được thực hiện bằng cách chuột phải để chọn chân trực tiếp và kích chuột trái vào chân mà mình muốn cài đặt. Ví dụ: cài đặt chân PC13 hoạt động với chức năng OUTPUT



Hình 1- 25. Cấu hình các ngoại vi của STM32F103C8T6

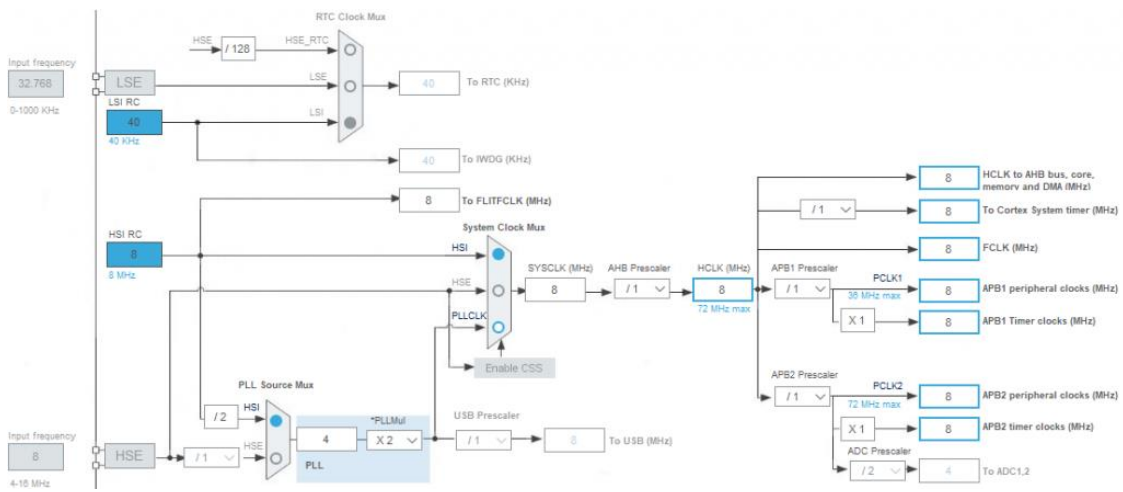
Ngoài ra, cũng có thể cấu hình các ngoại vi khác tại các mục: System Core, Analog, Timers, Connectivity...



Hình 1- 26. Các cấu hình ngoại vi khác

1.9.3 Cấu hình xung clock

Tại mục Clock Configuration: cấu hình lựa chọn nguồn tạo dao động và tần số hoạt động cho vi điều khiển (Bộ xử lý trung tâm – CPU và Peripherals – các ngoại vi) thông qua Clock tree. (Kết hợp với cấu hình RCC tại System Core)



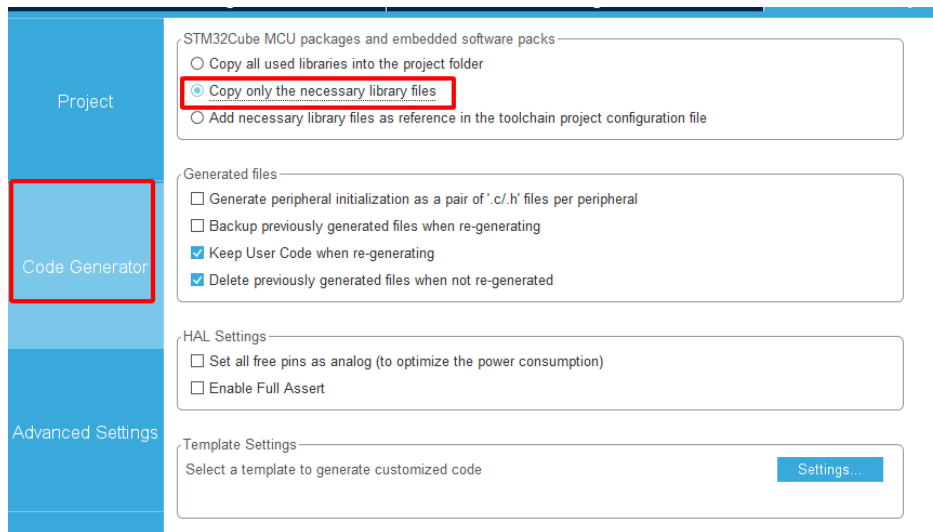
Hình 1- 27. Cấu hình xung clock của STM32F103C8T6

1.9.4 Lưu thông tin Project và sinh code

Tại Project Manager, đặt tên Project, nơi lưu trữ (lưu ý không sử dụng Tiếng Việt có dấu), và chọn Toolchain /IDE là MDK-ARM V5 vì sử dụng Keil C IDE để code và debug. Sau khi cấu hình xong, bấm vào GENERATE CODE để sinh code. Sau khi đã sinh code thì sẽ có thông báo Open Project. Lúc này Project sẽ được mở lên ở phần mềm Keil C với đầy đủ các cấu hình đã thực hiện trước đó

Pinout & Configuration	Clock Configuration	Project Manager
Project Settings Project Name: project_ARM Project Location: C:\Users\ADMIN\OneDrive\Documents\STM32F103\cuoiiky Application Structure: Basic <input type="checkbox"/> Do not generate the main()		
Code Generator Toolchain Folder Location: C:\Users\ADMIN\OneDrive\Documents\STM32F103\cuoiiky\project_ARM\		
Advanced Settings Linker Settings Minimum Heap Size: 0x200 Minimum Stack Size: 0x400		
Mcu and Firmware Package Mcu Reference: STM32F103C8Tx Firmware Package Name and Version: STM32Cube_FW_F1_V1.8.4 Migrate to the latest supported Firmware version <input checked="" type="checkbox"/> Use Default Firmware Location C:\Users\ADMIN\STM32Cube\Repository\STM32Cube_FW_F1_V1.8.4 Browse		

Hình 1- 28. Lưu thông tin project và sinh code

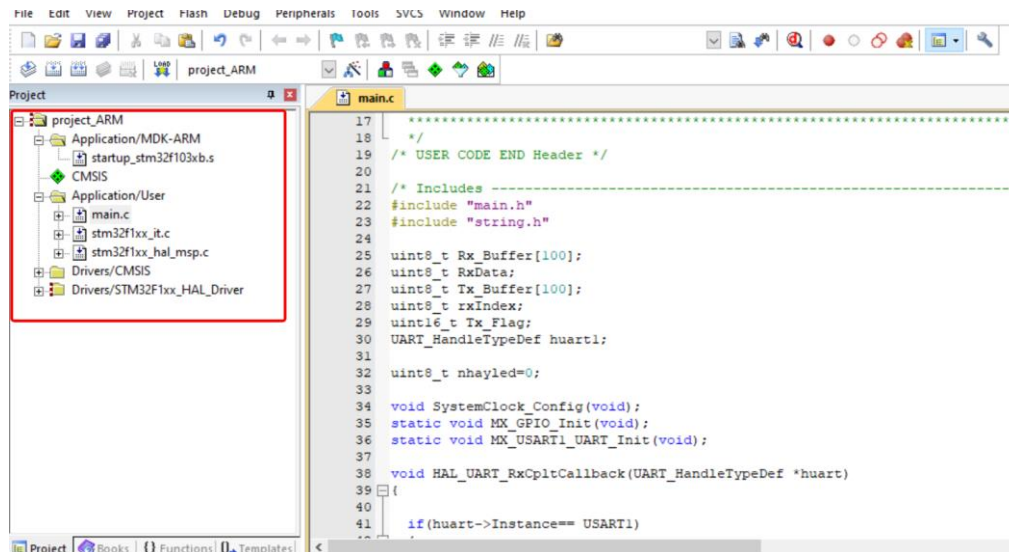


Hình 1- 29. Cài đặt Project

1.9.5 Các thao tác với Keil C

Sau khi cấu hình, sinh code từ phần mềm STM32CubeMX và mở Project Keil C, mở file main.c tại mục Application/User.

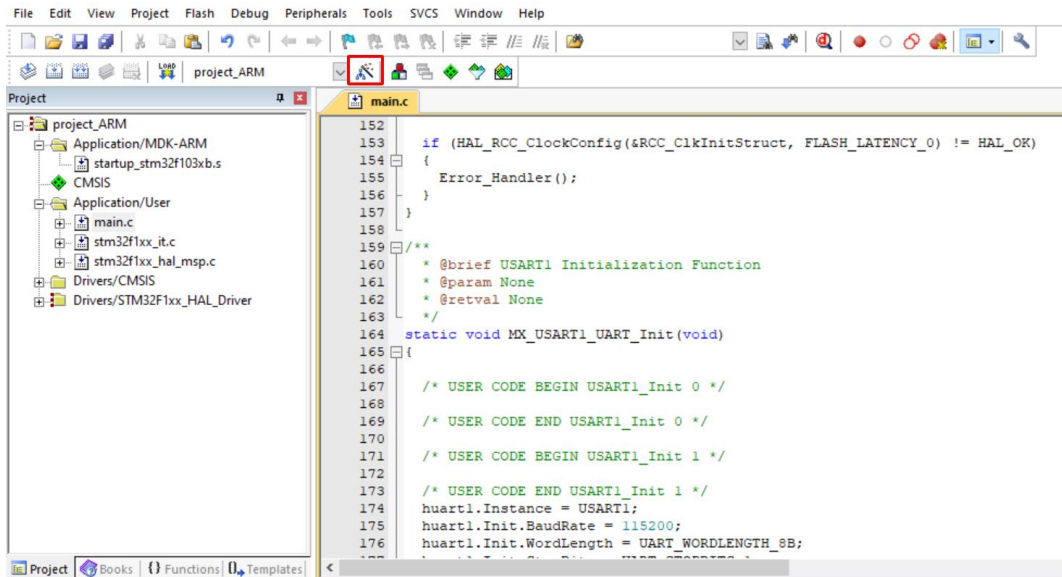
Có các biểu tượng Build (F7) để compile chương trình và Load (F8) để nạp chương trình.



Hình 1- 30. Các thao tác trên Keil uVision 5

1.9.6 Nạp chương trình

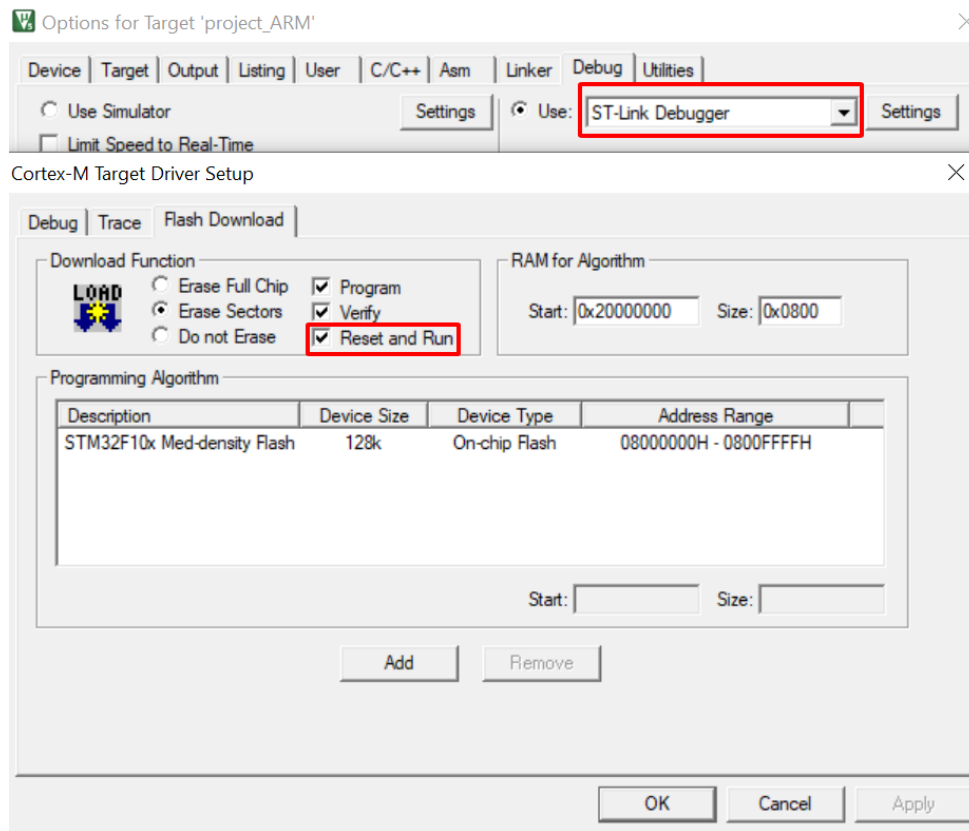
Nhấn vào biểu tượng Option for Target hoặc Project -> Option for Target để thực hiện 1 số cấu hình:



Hình 1- 31. Vị trí Option for Target trong Keil uVision 5

Tại cửa sổ Option for Target, chọn thẻ Debug và tick chọn Use ST-Link Debugger, để nạp chương trình xuống kit (nếu chọn Use Simulator thì sẽ ở chế độ mô phỏng).

Nhấn vào Settings tại Use: ST-Link Debugger, cửa sổ Cortex-M Target Driver Setup, chọn thẻ Flash Download, tại đây nếu tick chọn Reset and Run thì ngay sau khi nạp, chương trình sẽ chạy ngay. Nếu không tick chọn thì các bạn nạp code xong, nhấn vào nút reset trên board mạch thì chương trình mới chạy.



Hình 1- 32. Cửa sổ Option for Target trong Keil uVision 5

CHƯƠNG 2: THỰC NGHIỆM TRIỂN KHAI

2.1. Cài đặt, cấu hình

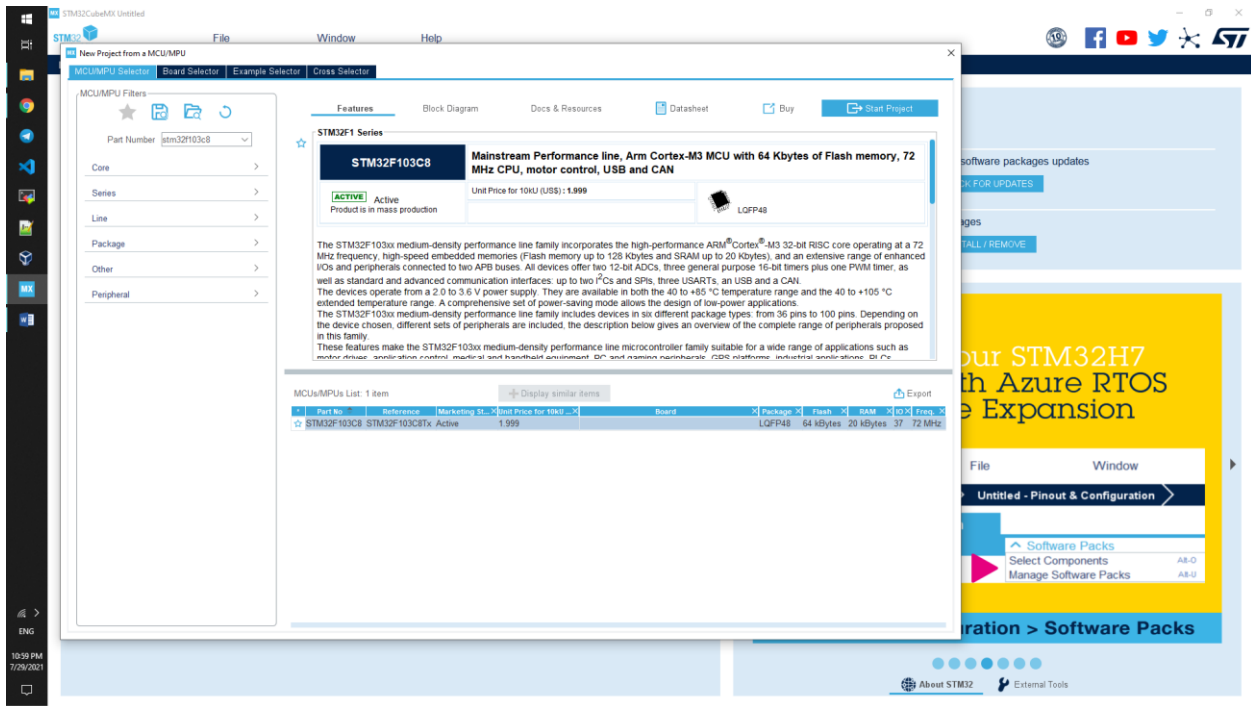
2.1.1. Thiết kế thuật toán cho chương trình

Xe được điều khiển từ xa thông qua thuật toán tự điều khiển bằng bluetooth. Sử dụng giao thức UART của stm32f103c8t6, khi ngắt UART nhận được 1 ký tự. Kiểm tra ký tự đầy đủ khớp với các ký tự đã khai báo tương ứng thì thực thi các câu lệnh HAL_GPIO_WritePin chứa các hành động GPIO_PIN_SET và GPIO_PIN_RESET. Các chân GPIO output thể hiện cho IN1,2,3,4 của module L298N. Từ đây ta sẽ điều khiển xe theo các hướng tiến lên, lùi xuống, dừng lại, rẽ trái, rẽ phải.

Xe kết hợp tính năng tránh vật cản sử dụng nguyên lý tránh vật cản. Module HC-SR04 có nhiệm vụ xem khoảng cách giữa xe với các vật cản. Nếu khoảng cách nhỏ hơn 20 cm thì servo sẽ xét 2 góc là 0 độ với 180 độ, đặt 2 biến distance 0 và distance 180 rồi so sánh giá trị của distance nào lớn hơn thì sẽ quay sang hướng đó

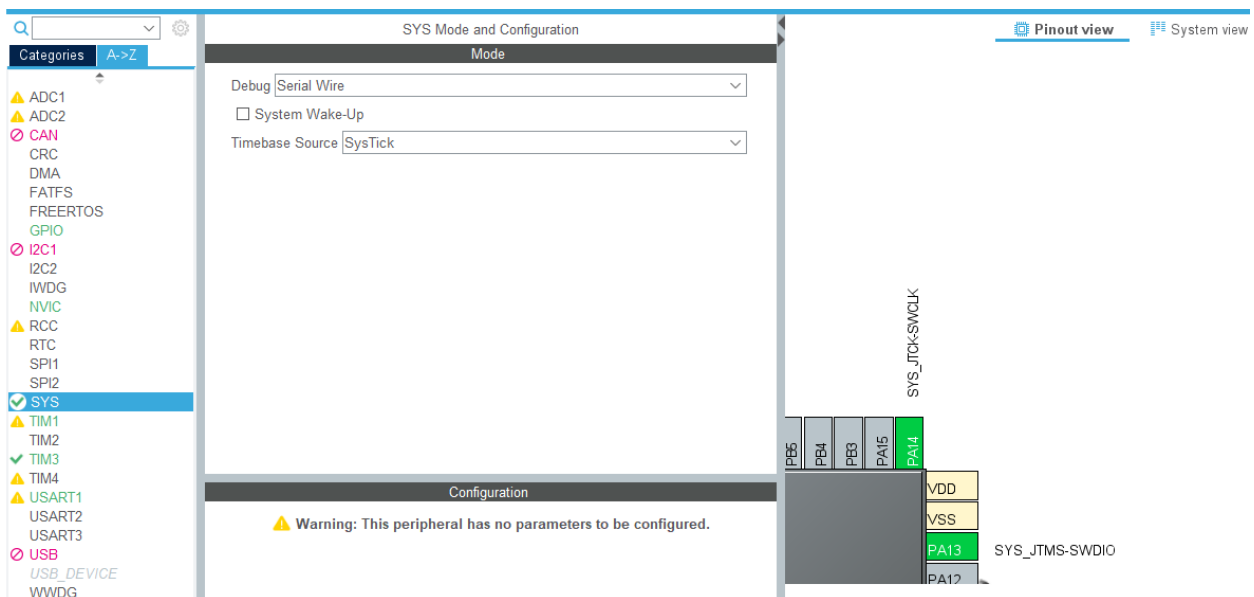
2.1.2. Cấu hình trên STM32CubeMX

Chọn mạch STM32F103C8T6



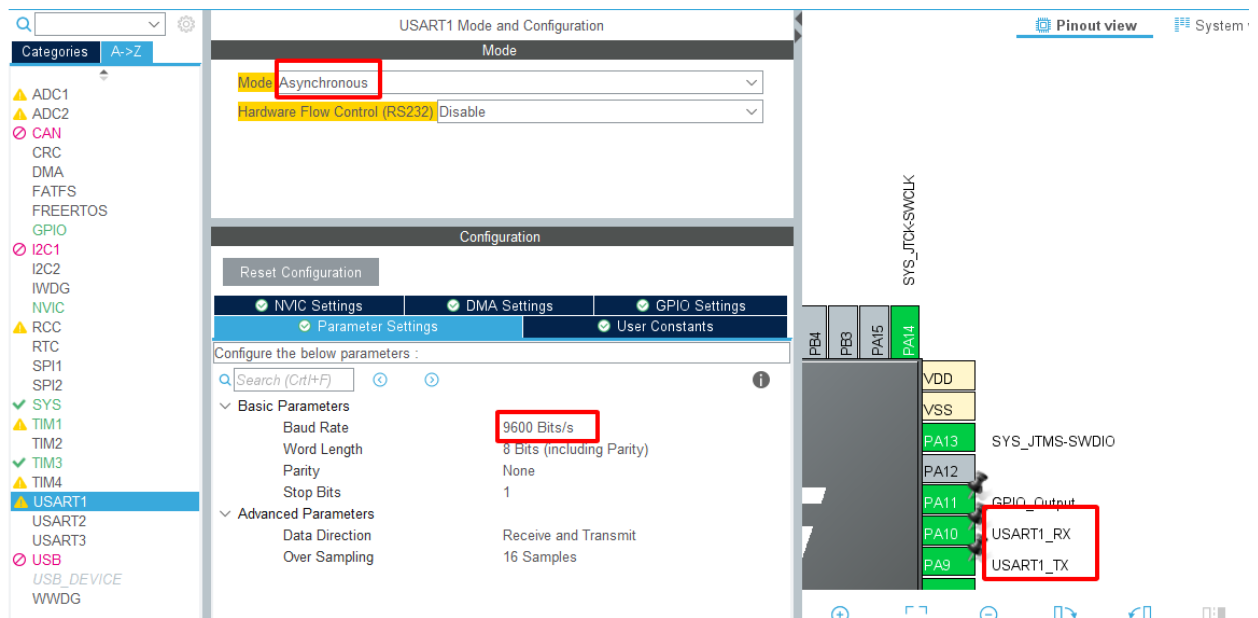
Hình 2- 1. Chọn mạch STM32F103C8T6 trên STM32CubeMX

Cấu hình việc nạp code ở module SYS sang Serial Wire



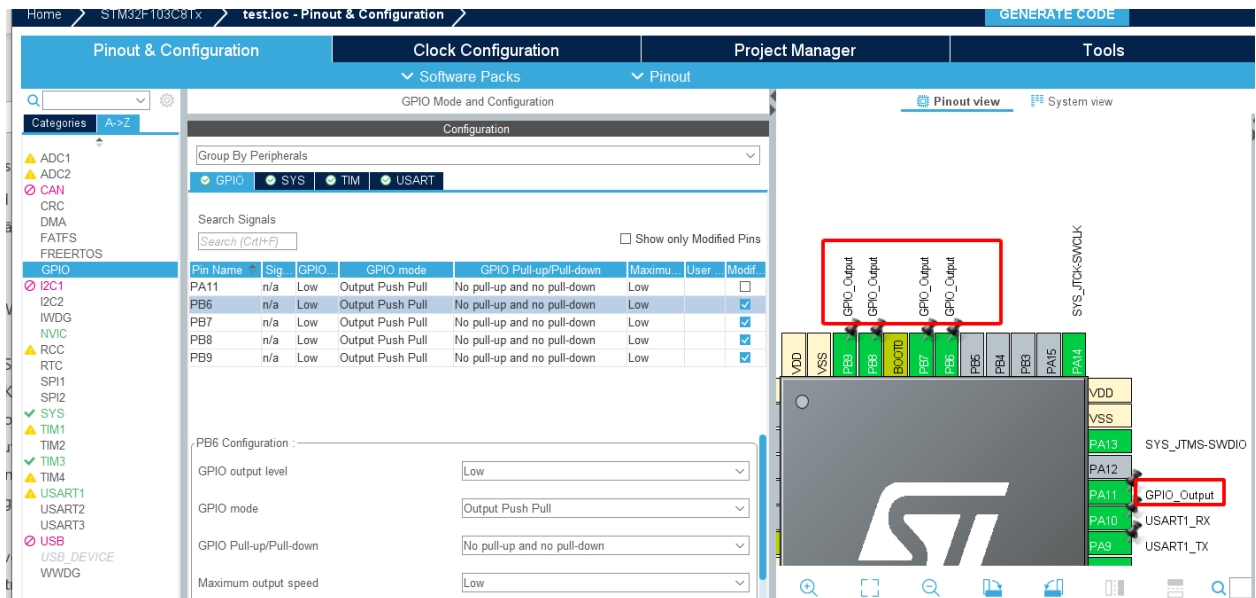
Hình 2- 2. Cấu hình sys

Cấu hình UART1 ở mode Asynchronous. Chọn tốc độ baudrate 9600



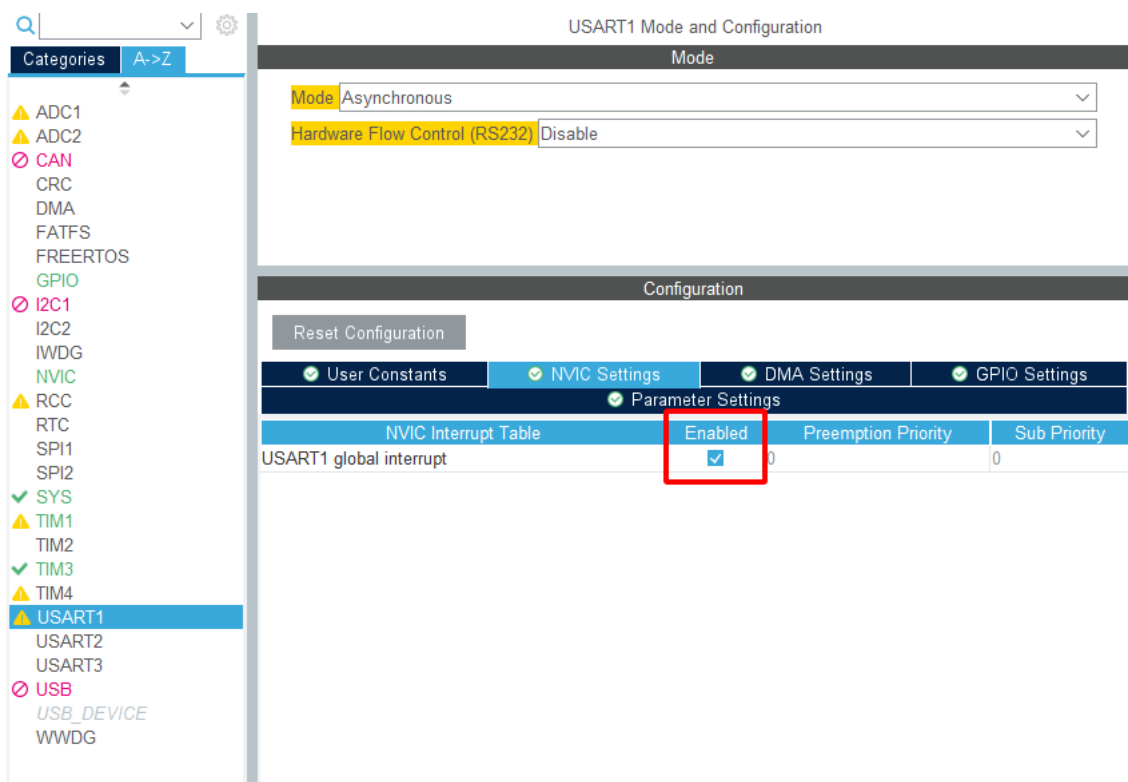
Hình 2- 3. Cấu hình UART

Set GPIO



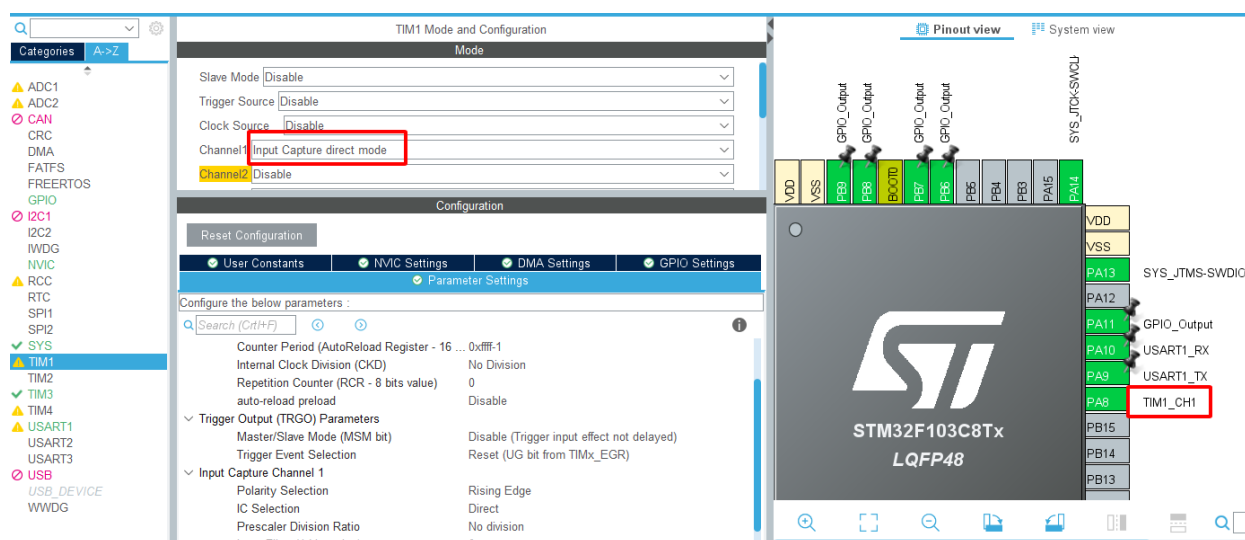
Hình 2- 4. Setting GPIO

Enable interrupt UART1



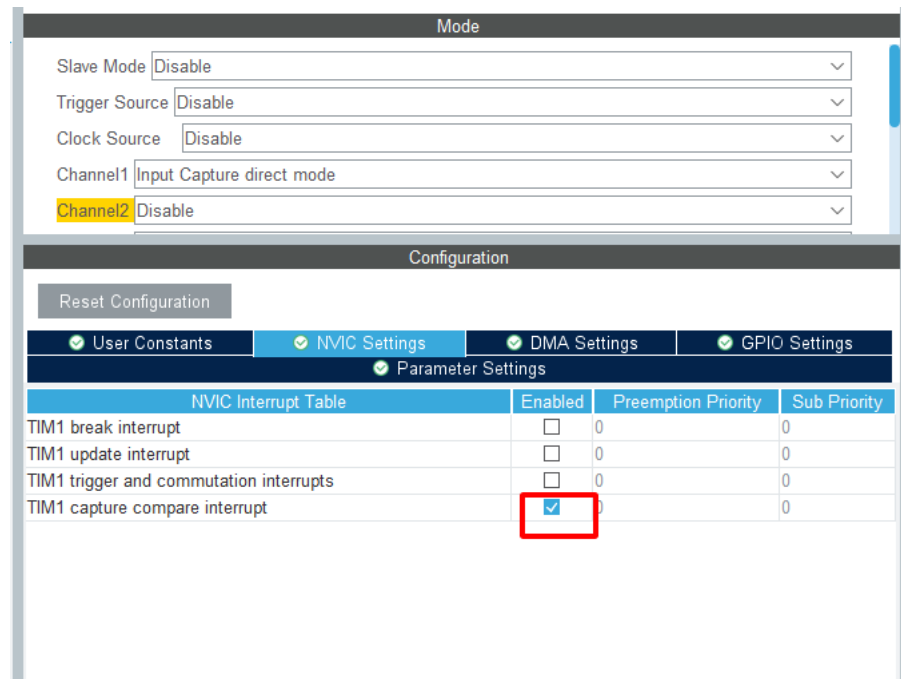
Hình 2- 5. Enable interrupt UART 1

Cấu hình TIM1: Channel 1 chọn Input capture direct mode



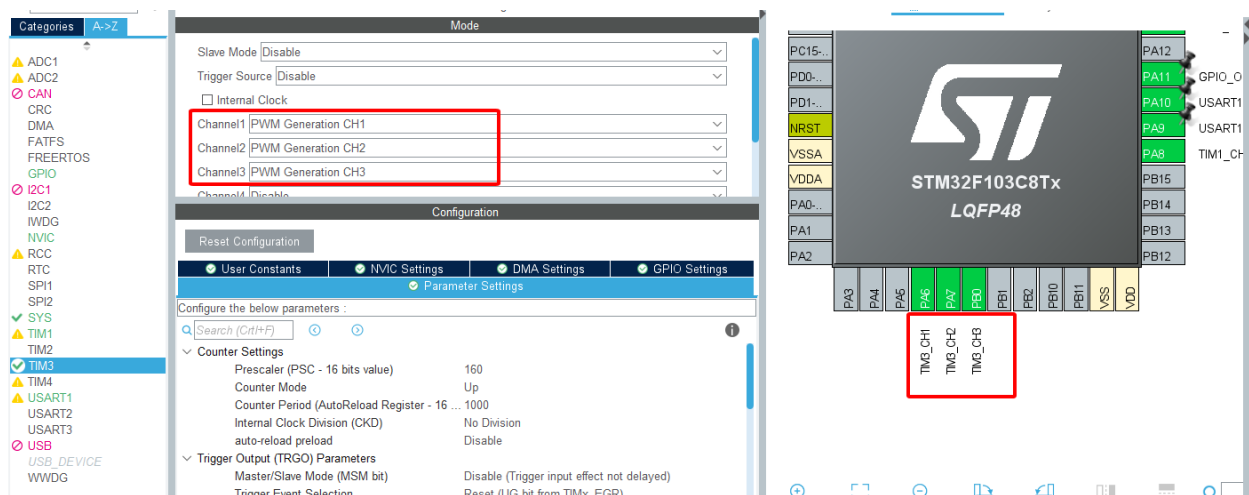
Hình 2- 6. Cấu hình TIM1

Enable interrupt TIM1



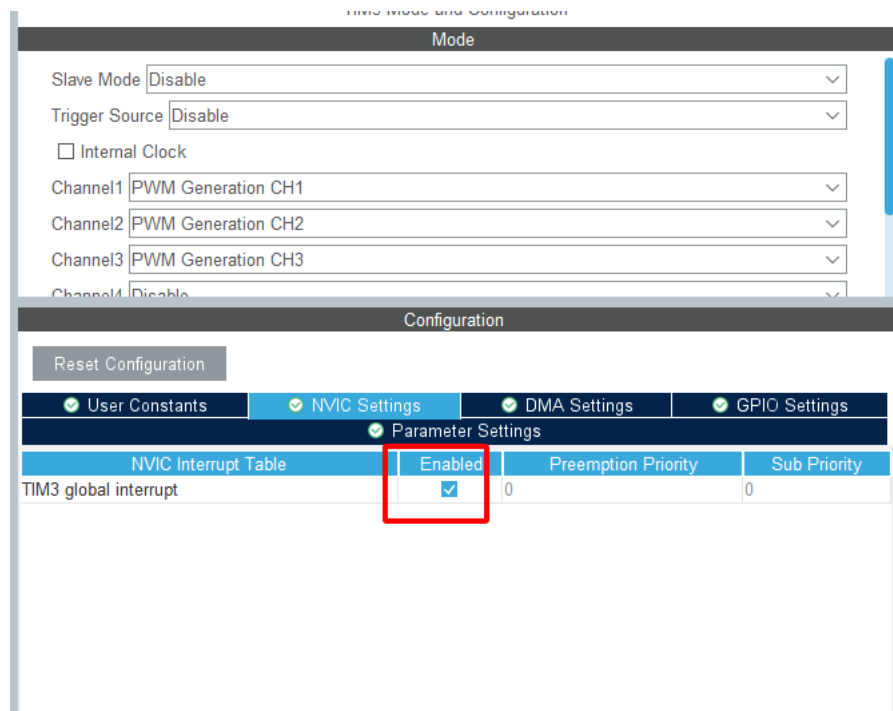
Hình 2- 7. Enable interrupt TIM1

Cấu hình TIM3: Ở Channel 1,2,3 chọn PWM Generation CH 1,2,3



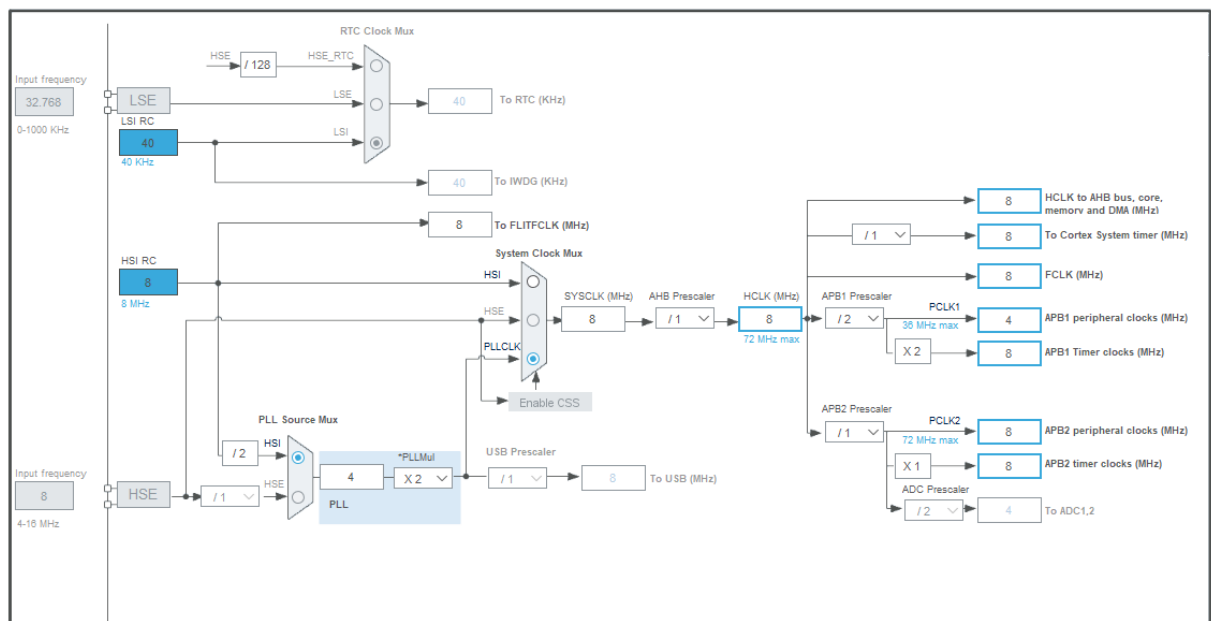
Hình 2- 8. Cấu hình TIM3

Enable interrupt TIM3



Hình 2- 9. Enable interrupt TIM3

Chọn xung Clock



Hình 2- 10. Setting xung clock cho STM32F103c8t6

Setting Project manager

The screenshot shows the 'Project Manager' dialog box in Keil uVision 5. The 'Project' tab is active. The 'Project Settings' section contains the following fields: 'Project Name' (test), 'Project Location' (C:\Users\ADMIN\OneDrive\Desktop\Nam5\ARM Nang cao\Robotcar\stm32robot\), 'Application Structure' (Basic), and 'Toolchain Folder Location' (C:\Users\ADMIN\OneDrive\Desktop\Nam5\ARM Nang cao\Robotcar\stm32robot\). There is a checkbox 'Do not generate the main()' and a checkbox 'Generate Under Root'. The 'Linker Settings' section includes 'Minimum Heap Size' (0x200) and 'Minimum Stack Size' (0x400). The 'Mcu and Firmware Package' section includes 'Mcu Reference' (STM32F103C8Tx) and 'Firmware Package Name and Version' (STM32Cube_FW_F1 V1.8.0). A button 'Migrate to the latest supported Firmware version' is present.

Hình 2- 11. Cấu hình project manager

Code trên Keil uVision 5

- Khai báo biến

```
char Rx_Data[2];  
  
uint32_t IC_Val1 = 0;  
uint32_t IC_Val2 = 0;  
uint32_t Difference = 0;  
uint8_t Is_First_Captured = 0;  
int d1,d2 =0;  
int Distance=0;  
static uint8_t vatcan=0;
```

```
static int counter=0;
static int tuhanh=0;
```

- Các hàm trạng thái của xe điều khiển: “up” là “tiền lên”; “back” là “lùi lại”; “stop” là “dừng lại”

```
void motor1_up()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_RESET);
}
void motor1_back()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_SET);
}
void motor1_stop()
{

```



```
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_RESET);

    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_RESET);
}

void motor2_up()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,GPIO_PIN_SET);

    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_RESET);
}

void motor2_back()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,GPIO_PIN_RESET);

    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_SET);
}

void motor2_stop()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,GPIO_PIN_RESET);

    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_RESET);
}
```

```
void motor_up()
{
    motor1_up();
    motor2_up();
}

void motor_back()
{
    motor1_back();
    motor2_back();
}

void motor_left()
{
    motor1_up();
    motor2_stop();
}

void motor_right()
{
    motor1_stop();
    motor2_up();
}
```

```

}

void motor_stop()

{

    motor1_stop();

    motor2_stop();

}

```

- Hàm Delay

```

void delay(uint16_t time)

{

    __HAL_TIM_SET_COUNTER(&htim1,0);

    while(__HAL_TIM_GET_COUNTER(&htim1) < time);

}

```

- Hàm HAL_TIM_IC_CaptureCallback: *Hàm xử lý khi có ngắt (khi bắt được sườn của xung)*

```

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)

{

    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)

```

```

{

    if (Is_First_Captured==0)

    {

        IC_Val1 = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_1);

        Is_First_Captured = 1;

    }

    else if (Is_First_Captured==1)

    {

        IC_Val2 = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_1);

        __HAL_TIM_SET_COUNTER(htim, 0);

        if (IC_Val2 > IC_Val1)

        {

            Difference = IC_Val2-IC_Val1;

        }

        else if (IC_Val1 > IC_Val2)

        {

```

```

        Difference = (0xffff - IC_Val1) + IC_Val2;

    }

    if(vatcan==1)

    {

        switch(counter)

        {

            case '0' : d1 = Difference * .034/2;

                        counter++;

                        break;

            case '1' : d2 = Difference * .034/2;

                        counter=0;

                        vatcan=0;

                        break;

        }

    }else

    {

        Distance = Difference * .034/2;

    }

    Is_First_Captured = 0;

```

```

        __HAL_TIM_DISABLE_IT(&htim1, TIM_IT_CC1);

    }

}

}

```

- Hàm HCSR04_Read

```

void HCSR04_Read (void)
{
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET);
    delay(10);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET);

    __HAL_TIM_ENABLE_IT(&htim1, TIM_IT_CC1);
}

```

- Hàm tuhanh_func

```

void tuhanh_func()
{
    HCSR04_Read();
    if(Distance <= 20)
    {
        motor_stop();
        HAL_Delay(100);
        htim3.Instance->CCR1=350;
    }
}

```

```

    htim3.Instance->CCR2=350;
    motor_back();
    HAL_Delay(300);
    motor_stop();
    HAL_Delay(200);
    htim3.Instance->CCR3=0;
    vatcan=1;
    HCSR04_Read();
    HAL_Delay(100);
    htim3.Instance->CCR3=300;
    HAL_Delay(100);
    htim3.Instance->CCR3=600;
    HAL_Delay(100);
    htim3.Instance->CCR3=900;
    HAL_Delay(100);
    htim3.Instance->CCR3=1000;
    HCSR04_Read();
    HAL_Delay(100);
    htim3.Instance->CCR3=500;
    if((d1 < 20)&& (d2 < 20))
    {
        motor_back();
        HAL_Delay(300);
        motor_stop();
        HAL_Delay(300);
    }
    else

```

```

        if( d1 >d2)
        {
            motor_right();
            HAL_Delay(750);
        }
        else
        if( d1 <=d2)
        {
            motor_left();
            HAL_Delay(750);
        }

    }else
    {
        motor_up();
    }
}

```

- Hàm nhận ngắt HAL_UART_RxCpltCallback:

```

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(Rx_Data[0]=='0')
    {
        htim3.Instance->CCR1=100;
        htim3.Instance->CCR2=100;
    }else
    if(Rx_Data[0]=='1')

```



```

{
    htim3.Instance->CCR1=150;
    htim3.Instance->CCR2=150;
}else if(Rx_Data[0]=='2')
{
    htim3.Instance->CCR1=250;
    htim3.Instance->CCR2=250;
}
else if(Rx_Data[0]=='3')
{
    htim3.Instance->CCR1=350;
    htim3.Instance->CCR2=350;
}
else if(Rx_Data[0]=='4')
{
    htim3.Instance->CCR1=450;
    htim3.Instance->CCR2=450;
}
else if(Rx_Data[0]=='5')
{
    htim3.Instance->CCR1=550;
    htim3.Instance->CCR2=550;
}
else if(Rx_Data[0]=='6')
{
    htim3.Instance->CCR1=650;
    htim3.Instance->CCR2=650;
}

```

```

    }
    else if(Rx_Data[0]=='7')
    {
        htim3.Instance->CCR1=750;
        htim3.Instance->CCR2=750;
    }
    else if(Rx_Data[0]=='8')
    {
        htim3.Instance->CCR1=850;
        htim3.Instance->CCR2=850;
    }
    else if(Rx_Data[0]=='9')
    {
        htim3.Instance->CCR1=900;
        htim3.Instance->CCR2=900;
    }
    else if(Rx_Data[0]=='q')
    {
        htim3.Instance->CCR1=1000;
        htim3.Instance->CCR2=1000;
    }
    if(Rx_Data[0]=='F')
    {
        motor_up();
        HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);
        HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
    }

```

```

    }
else
{

    if(Rx_Data[0]=='L')
    {
        if(tuhanh==0)
            motor_left();
        HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);
        HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);

    }
else
    if(Rx_Data[0]=='R')
    {
        if(tuhanh==0)
            motor_right();
        HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);
        HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
    }
else
    if(Rx_Data[0]=='S')
    {
        if(tuhanh==0)
            motor_stop();
        HAL_UART_Transmit(&huart1,(uint8_t
*)Rx_Data,1,1000);

```

```

        HAL_UART_Receive_IT(&huart1,(uint8_t
*)Rx_Data,1);

    }
    else
    if(Rx_Data[0]=='B')
    {
        if(tuhanh==0)
            motor_back();
        HAL_UART_Transmit(&huart1,(uint8_t
*)Rx_Data,1,1000);

        HAL_UART_Receive_IT(&huart1,(uint8_t
*)Rx_Data,1);

    }else
    if(Rx_Data[0]=='H')
    {
        tuhanh=1;
        HAL_UART_Transmit(&huart1,(uint8_t
*)Rx_Data,1,1000);

        HAL_UART_Receive_IT(&huart1,(uint8_t
*)Rx_Data,1);

    }
    else
        if(Rx_Data[0] == 'B')
        {
            tuhanh=0;

        HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);

```

```

        HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
    }

}
HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
}

```

- Trong while(1) của hàm Main gọi lại hàm tuhanh_func nếu biến tuhanh = 1

```

int main(void)
{
    HAL_Init();

    SystemClock_Config();

    HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);

    MX_GPIO_Init();
    MX_USART1_UART_Init();
    MX_TIM3_Init();
    MX_TIM1_Init();

    HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
    HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);

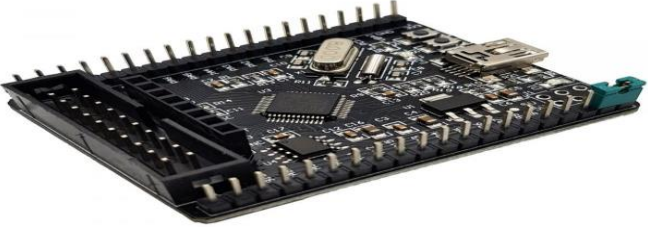



    while (1)
    {

```

```
if(tuhanh==1)
    {
        tuhanh_func();
    }
}
```

2.2. Thiết kế, lắp đặt phần cứng

2.2.1. Cơ sở thiết yếu

<i>STM32F103C8T6 Smart v2</i>	
<i>Mạch nạp ST-LINK V2</i>	
Khung robot car	
Module HC-SR04	


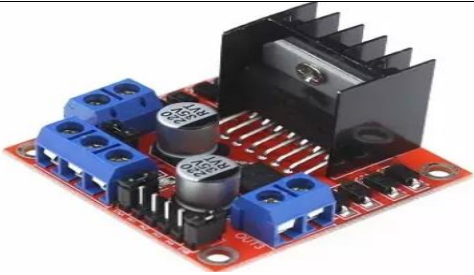
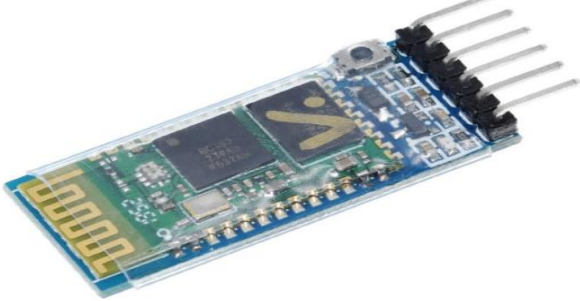
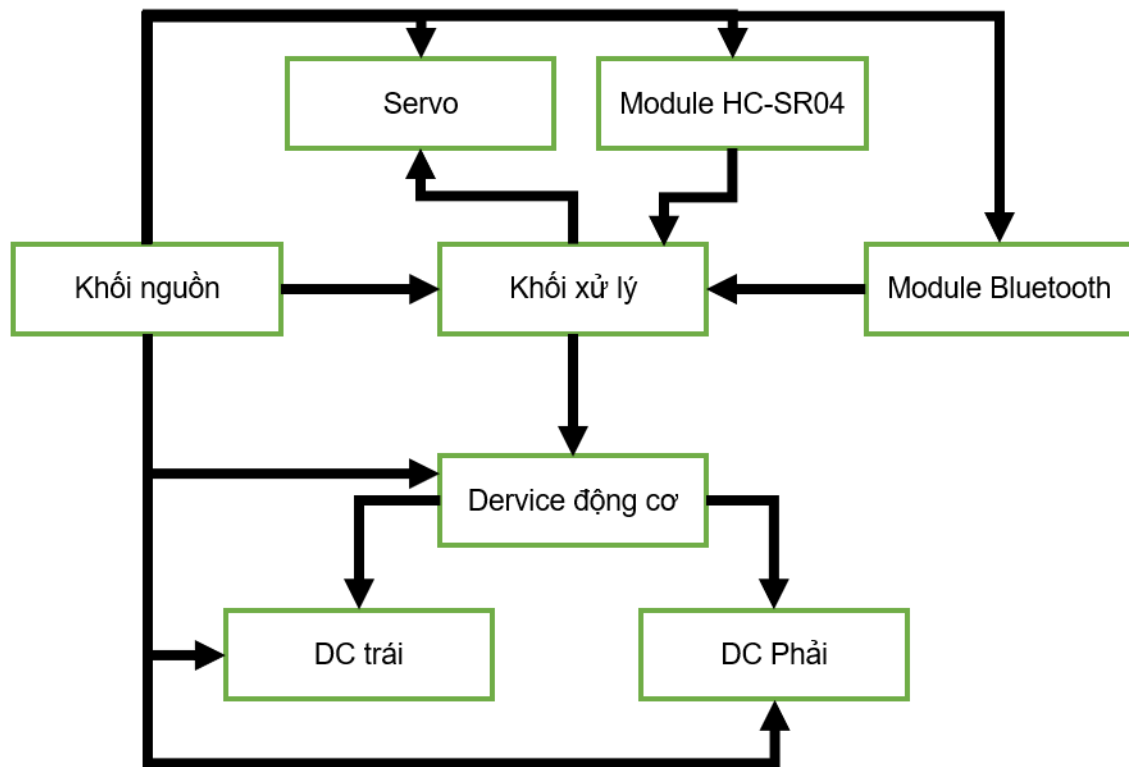
Servo GS90	
Module L298N	
Module bluetooth	

Table 2- 1. Các cơ sở thiết yếu

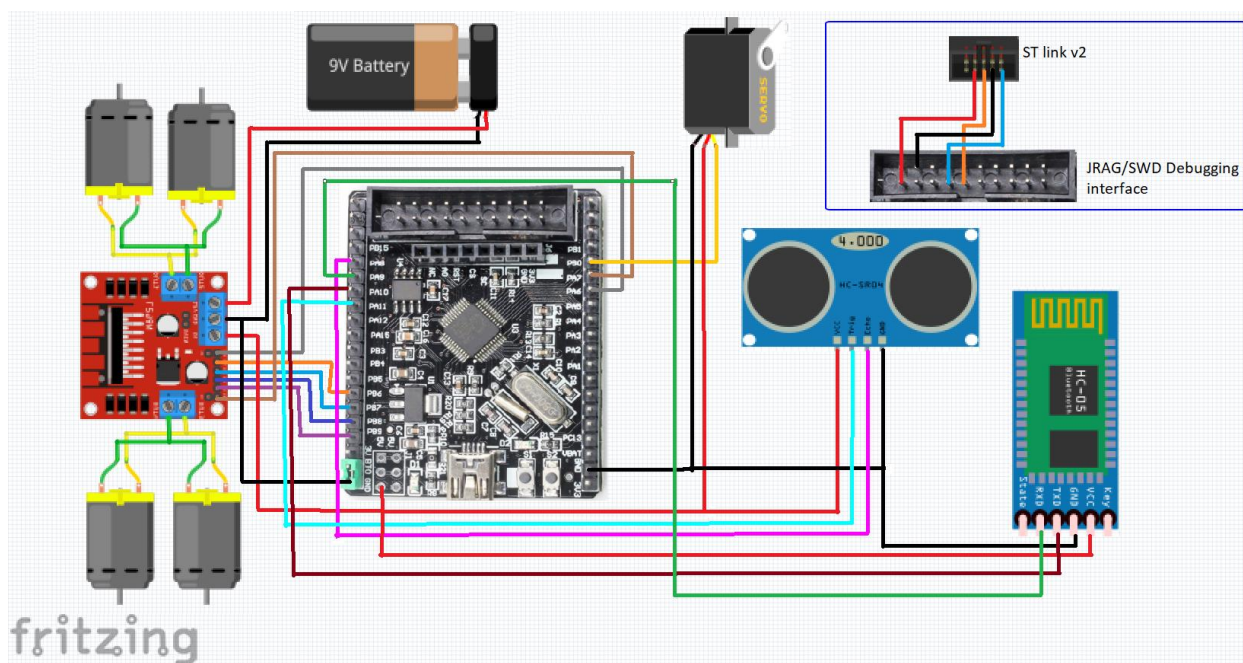
2.2.2. Sơ đồ

Sơ đồ khối



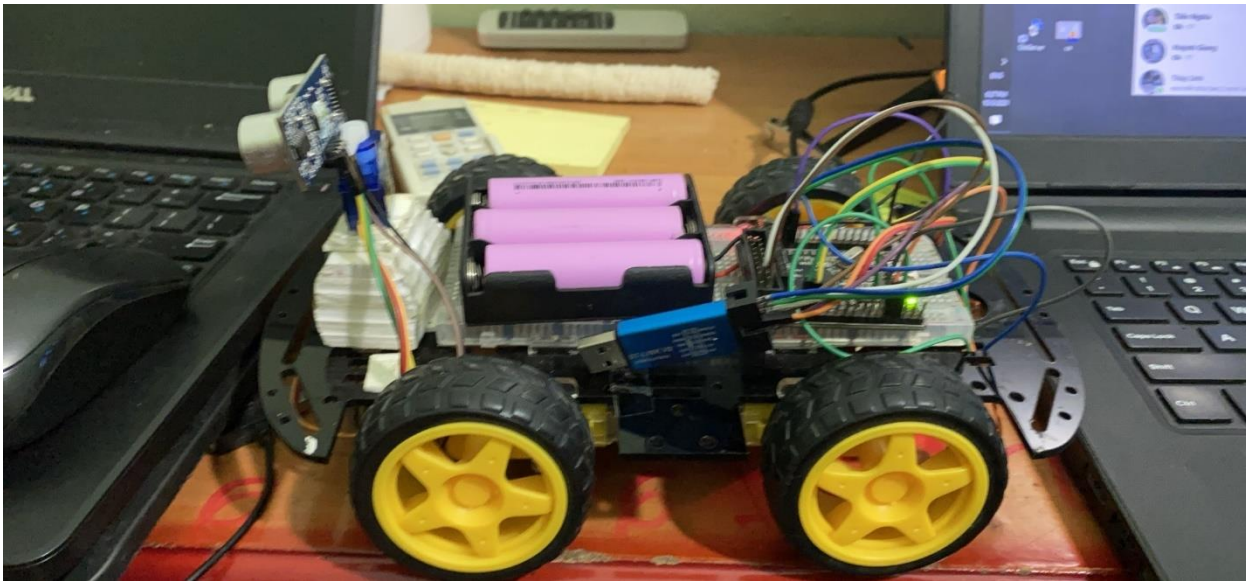
Hình 2- 12. Sơ đồ khối của hệ thống

Sơ đồ phần cứng

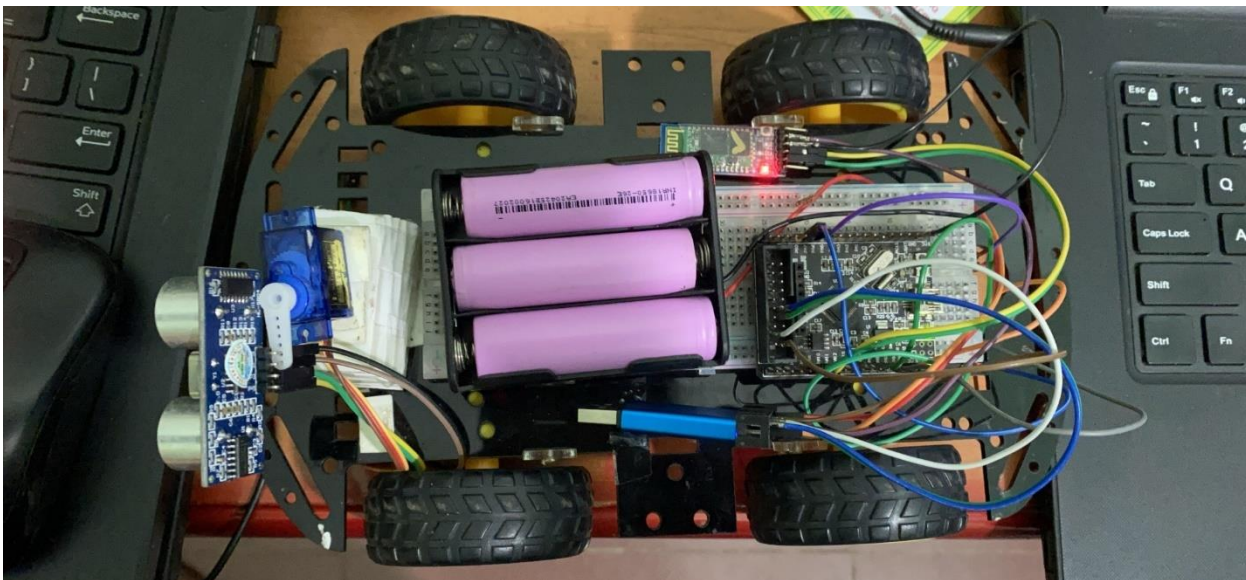


Hình 2- 13. Sơ đồ phân cứng hệ thống

2.2.3. Thi công, lắp ráp



Hình 2- 14. Mô hình sau khi lắp ráp 1



Hình 2- 15. Mô hình sau khi lắp ráp 2

KẾT LUẬN

Kết quả đạt được:

Giao tiếp thành công giữa các module. Hệ thống đã hoạt động ổn định

Như vậy, việc xây dựng xe điều khiển bằng Bluetooth và tránh vật cản giúp chúng ta hiểu hơn về nguyên lý hoạt động của mạch vi xử lý, module điều khiển động cơ L298 và mạch chân HC-05 từ đó có thể tự lập trình STM32 và tìm hiểu rõ hơn về cách thức hoạt động của việc tự động tránh vật cản. Cùng với sự dẫn dắt của thầy giáo, nhóm chúng em đã hoàn thành sản phẩm và có được nhiều hiểu biết hơn về nhúng và có cái nhìn tổng quan nhất về môn lập trình ARM nâng cao.

Hướng phát triển:

Tích hợp thêm nhiều chức năng cho xe như: truyền hình ảnh, đo nhiệt độ, độ ẩm, khoảng cách vật cản, đo độ nghiêng.

Phản hồi được các sự cố về thiết bị cầm tay.

Ứng dụng công nghệ Bluetooth vào các hệ thống khác

TÀI LIỆU THAM KHẢO

- [1] Nguyễn Đào Trường - NDTRUONG_GT L_trinh ARM Nang cao V2.pdf

PHỤ LỤC

Trích dẫn mã nguồn

```
#include "main.h"

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim3;

char Rx_Data[2];

UART_HandleTypeDef huart1;

uint32_t IC_Val1 = 0;
uint32_t IC_Val2 = 0;
uint32_t Difference = 0;
uint8_t Is_First_Captured = 0;
int d1,d2 =0;
int Distance=0;
static uint8_t vatcan=0;
static int counter=0;
static int tuhanh=0;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM1_Init(void);
```

```

void motor1_up()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_RESET);
}

void motor1_back()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_SET);
}

void motor1_stop()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_RESET);
}

void motor2_up()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_RESET);
}

void motor2_back()
{
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_SET);
}

void motor2_stop()
{

```

```
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_9,GPIO_PIN_RESET);
}

void motor_up()
{
    motor1_up();
    motor2_up();
}

void motor_back()
{
    motor1_back();
    motor2_back();
}

void motor_left()
{
    motor1_up();
    motor2_stop();
}

void motor_right()
{
    motor1_stop();
    motor2_up();
}

void motor_stop()
{
    motor1_stop();
```



```

        motor2_stop();
    }

void delay(uint16_t time)
{
    __HAL_TIM_SET_COUNTER(&htim1,0);
    while(__HAL_TIM_GET_COUNTER(&htim1) < time);
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
    {
        if (Is_First_Captured==0)
        {
            IC_Val1 = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_1);
            Is_First_Captured = 1;
        }
        else if (Is_First_Captured==1)
        {
            IC_Val2 = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_1);
            __HAL_TIM_SET_COUNTER(htim, 0);
            if (IC_Val2 > IC_Val1)
            {
                Difference = IC_Val2-IC_Val1;
            }
        }
    }
}

```

```

    }

    else if (IC_Val1 > IC_Val2)
    {
        Difference = (0xffff - IC_Val1) + IC_Val2;
    }
    if(vatcan==1)
    {
        switch(counter)
        {
            case '0' : d1 = Difference * .034/2;
                        counter++;
                        break;
            case '1' : d2 = Difference * .034/2;
                        counter=0;
                        vatcan=0;
                        break;

        }
    }else
    {
        Distance = Difference * .034/2;
    }
    Is_First_Captured = 0;
    __HAL_TIM_DISABLE_IT(&htim1, TIM_IT_CC1);
}
}

```

```

}

void HCSR04_Read (void)
{
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET);
    delay(10);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET);
    __HAL_TIM_ENABLE_IT(&htim1, TIM_IT_CC1);
}

void tuhanh_func()
{
    HCSR04_Read();
    if(Distance <= 20)
    {
        motor_stop();
        HAL_Delay(100);
        htim3.Instance->CCR1=350;
        htim3.Instance->CCR2=350;
        motor_back();
    }
    HAL_Delay(300);
    motor_stop();
    HAL_Delay(200);
    htim3.Instance->CCR3=0;
    vatcan=1;
    HCSR04_Read();
    HAL_Delay(100);
    htim3.Instance->CCR3=300;
    HAL_Delay(100);
}

```

```

        htim3.Instance->CCR3=600;
        HAL_Delay(100);
        htim3.Instance->CCR3=900;
        HAL_Delay(100);
        htim3.Instance->CCR3=1000;
        HCSR04_Read();
        HAL_Delay(100);
        htim3.Instance->CCR3=500;
        if((d1 < 20)&& (d2 < 20))
    {
        motor_back();
        HAL_Delay(300);
        motor_stop();
        HAL_Delay(300);
    }
    else
    if( d1 >d2)
    {
        motor_right();
        HAL_Delay(750);
    }
    else
    if( d1 <=d2)
    {
        motor_left();
        HAL_Delay(750);
    }

```

```

    }else
    {
        motor_up();
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(Rx_Data[0]=='0')
    {
        htim3.Instance->CCR1=100;
        htim3.Instance->CCR2=100;
    }else
    if(Rx_Data[0]=='1')
    {
        htim3.Instance->CCR1=150;
        htim3.Instance->CCR2=150;
    }else if(Rx_Data[0]=='2')
    {
        htim3.Instance->CCR1=250;
        htim3.Instance->CCR2=250;
    }
    else if(Rx_Data[0]=='3')
    {
        htim3.Instance->CCR1=350;
        htim3.Instance->CCR2=350;
    }
}

```

```

    }
    else if(Rx_Data[0]=='4')
    {
        htim3.Instance->CCR1=450;
        htim3.Instance->CCR2=450;
    }
    else if(Rx_Data[0]=='5')
    {
        htim3.Instance->CCR1=550;
        htim3.Instance->CCR2=550;
    }
    else if(Rx_Data[0]=='6')
    {
        htim3.Instance->CCR1=650;
        htim3.Instance->CCR2=650;
    }
    else if(Rx_Data[0]=='7')
    {
        htim3.Instance->CCR1=750;
        htim3.Instance->CCR2=750;
    }
    else if(Rx_Data[0]=='8')
    {
        htim3.Instance->CCR1=850;
        htim3.Instance->CCR2=850;
    }
    else if(Rx_Data[0]=='9')

```

```

{
    htim3.Instance->CCR1=900;
    htim3.Instance->CCR2=900;
}
else if(Rx_Data[0]=='q')
{
    htim3.Instance->CCR1=1000;
    htim3.Instance->CCR2=1000;
}
if(Rx_Data[0]=='F')
{
    motor_up();
    HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);
    HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);

}
else
{

    if(Rx_Data[0]=='L')
    {
        if(tuhanh==0)
            motor_left();
        HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);
        HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);

    }
}

```

```

else
    if(Rx_Data[0]=='R')
    {
        if(tuhanh==0)
            motor_right();
        HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);
        HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
    }
    else
        if(Rx_Data[0]=='S')
        {
            if(tuhanh==0)
                motor_stop();
            HAL_UART_Transmit(&huart1,(uint8_t
*)Rx_Data,1,1000);
            HAL_UART_Receive_IT(&huart1,(uint8_t
*)Rx_Data,1);
        }
        else
            if(Rx_Data[0]=='B')
            {
                if(tuhanh==0)
                    motor_back();
                HAL_UART_Transmit(&huart1,(uint8_t
*)Rx_Data,1,1000);
                HAL_UART_Receive_IT(&huart1,(uint8_t
*)Rx_Data,1);

```



```

        }else
        if(Rx_Data[0]=='H')
        {
            tuhanh=1;
            HAL_UART_Transmit(&huart1,(uint8_t
*)Rx_Data,1,1000);

            HAL_UART_Receive_IT(&huart1,(uint8_t
*)Rx_Data,1);
        }
        else
            if(Rx_Data[0] == 'B')
            {
                tuhanh=0;

                HAL_UART_Transmit(&huart1,(uint8_t *)Rx_Data,1,1000);

                HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
            }

        }
        HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
    }
}

```

```

int main(void)
{

    HAL_Init();

    SystemClock_Config();
    HAL_UART_Receive_IT(&huart1,(uint8_t *)Rx_Data,1);
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    MX_TIM3_Init();
    MX_TIM1_Init();
    HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
    HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
    while (1)
    {
        if(tuhanh==1)
        {
            tuhanh_func();
        }
    }
}

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

```

```

RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}

RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0)
!= HAL_OK)
{
    Error_Handler();
}
}

```

```

static void MX_TIM1_Init(void)
{

    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_IC_InitTypeDef sConfigIC = {0};
    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 8-1;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 0xffff-1;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_IC_Init(&htim1) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig)
!= HAL_OK)
    {
        Error_Handler();
    }
    sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
    sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;

```

```

sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 0;
if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC,
TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
}

static void MX_TIM3_Init(void)
{

    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 160;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 1000;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;

```

```

    if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig)
    != HAL_OK)
    {
        Error_Handler();
    }
    sConfigOC.OCMode = TIM_OCMODE_PWM1;
    sConfigOC.Pulse = 0;
    sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
    if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_2) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
TIM_CHANNEL_3) != HAL_OK)
    {
        Error_Handler();
    }
    HAL_TIM_MspPostInit(&htim3);
}

```

```

static void MX_USART1_UART_Init(void)
{
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 9600;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart1) != HAL_OK)
    {
        Error_Handler();
    }
}

static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET);

```

```

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9, GPIO_PIN_RESET);

/*Configure GPIO pin : PA11 */
GPIO_InitStruct.Pin = GPIO_PIN_11;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : PB6 PB7 PB8 PB9 */
GPIO_InitStruct.Pin =
GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}
void Error_Handler(void)
{

}

#ifdef USE_FULL_ASSERT
/**

```



```

* @brief Reports the name of the source file and the source line number
*       where the assert_param error has occurred.
* @param file: pointer to the source file name
* @param line: assert_param error line source number
* @retval None
*/
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line
number,
    tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/***** (C) COPYRIGHT STMicroelectronics
*****/

```