Giáo trình ARM STM32F4
VIAM Lab

www.viamlab.com
SĐT: 0987814161

GIÁO TRÌNH ARM STM32F4

Mục lục

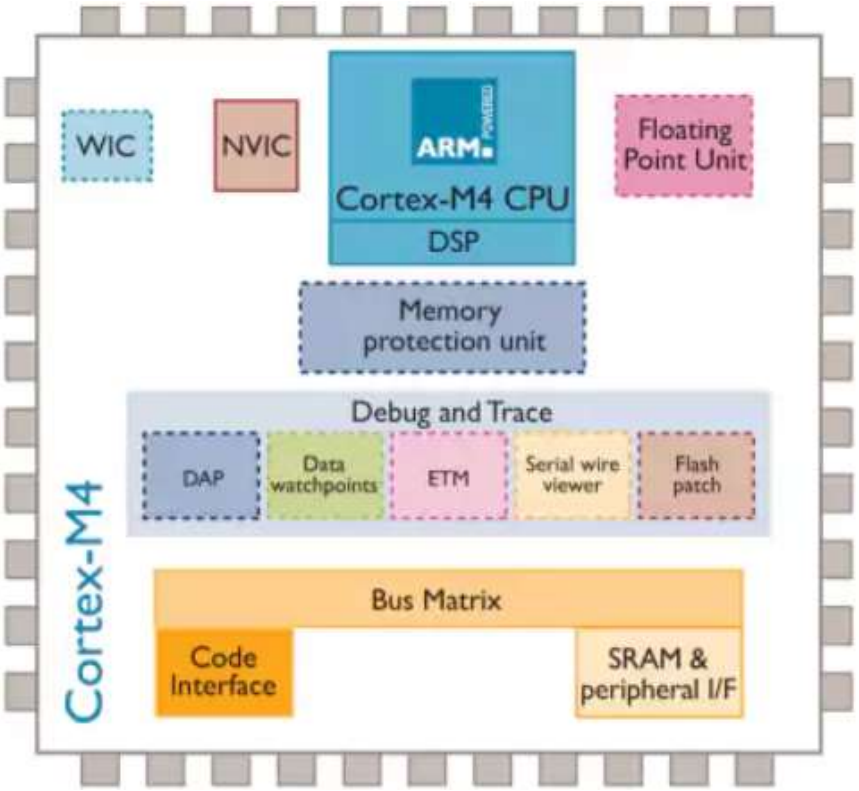
Phần 1: GIỚI THIỆU TỔNG QUAN VỀ ARM STM32F4

1. Tổng quan về dòng chip ARM Cortex M4:

Dòng ARM Cortex là một bộ vi xử lí thế hệ mới đưa ra một kiến trúc chuẩn, được xây dựng dựa trên kiến trúc RSIC, nó là một lõi xử lí hoàn thiện gồm 3 phân nhánh:

- Dòng A dành cho các ứng dụng cao cấp.
- Dòng R dành cho các ứng dụng thời gian thực.
- Dòng M dùng cho các ứng dụng vi điều khiển chi phí thấp.

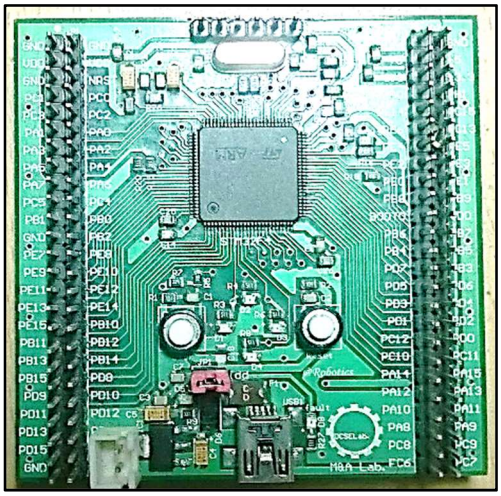
Dòng Cortex M-4 là một sự nâng cấp đáng kể của dòng Cortex M-3 với ưu điểm nâng cao hiệu suất hệ thống, kết hợp với tiêu thụ năng lượng thấp. Nó được sử dụng như một lõi vi điều khiển chuẩn nhằm cung cấp một cấu trúc tổng quát đầy đủ chức năng như hệ thống ngắt, SysTick timer (thiết kế cho hệ điều hành thời gian thực), hệ thống kiểm lỗi (debug system) và memory map. Các địa chỉ của Cortex M-4 được chia thành các vùng cho mã chương trình, SRAM, ngoại vi và ngoại vi hệ thống. Cortex M-4 được thiết kế dựa trên cấu trúc Harvard với điểm đặc trưng là bộ nhớ chương trình và bộ nhớ dữ liệu tách biệt nhau, nó cung cấp một số lượng lớn bus cho phép thực hiện nhiều thao tác song song với nhau, làm tăng hiệu suất của chip trong xử lý đa nhiệm.



Cấu trúc nhân Cortex-M4

2. Sơ lược Kit STM32F407 Cortex-M4:

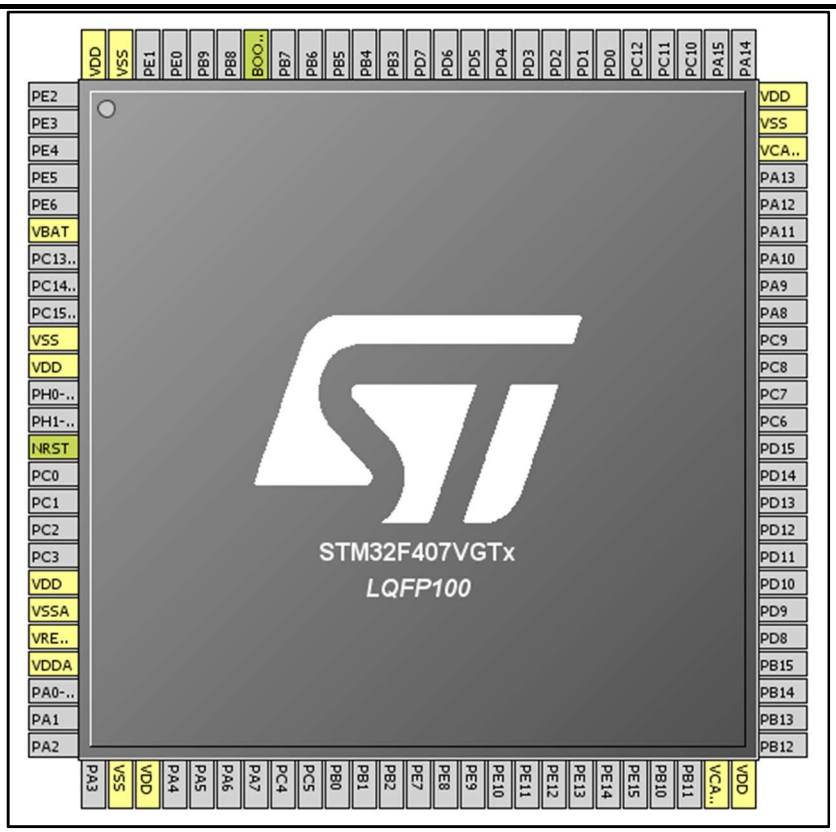
- Vi điều khiển chính: STM32F407VGT6 32-bit ARM Cortex-M4 core, 1 MB Flash, 192 KB RAM.
- Nguồn cấp từ cổng Mini USB qua các IC nguồn chuyển thành 3V3 để cấp cho MCU.
- Có các chân nguồn: 3 V and 5 V.
- Có 4 Led và 2 nút nhấn trong đó có một nút Reset.
- Có led thông báo trạng thái nguồn.



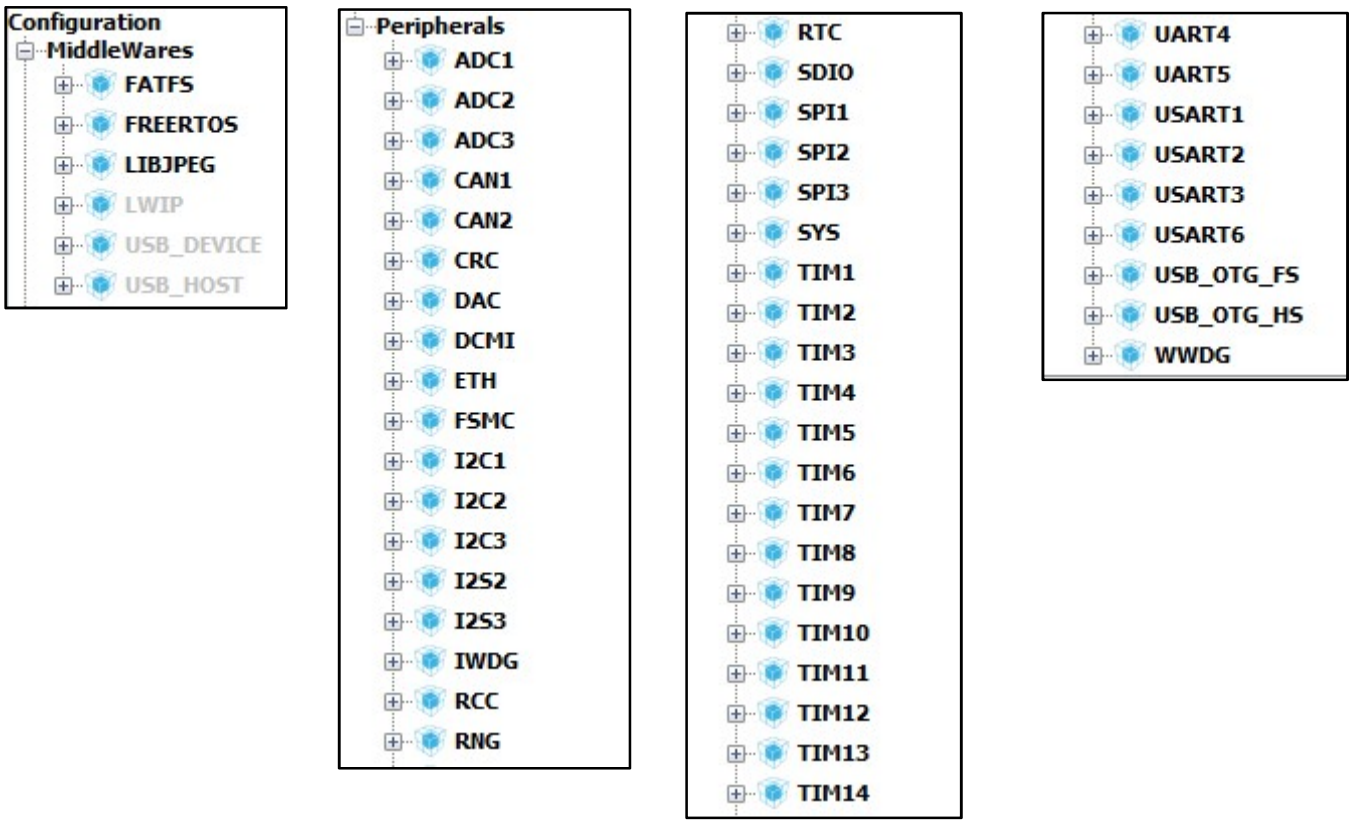
Kit STM32F407VGTxCortex_M4
VIAM Lab

- Chip STM32F407VGTx thuộc dòng hiệu suất cao ARM Cortex-M4 32bit STM32F4 của STMicroelectronics. STM32F407VGTx được trang bị 1MB Flash, 192KB RAM, tốc độ lên đến 168MHz. Nó có đầy đủ chức năng của vi điều khiển cơ bản với:

  - 3 Bộ ADC 12 bit với 16 kênh 2.4 MSPS
  - 2 Bộ DAC 12 bit 7.2 MSPS.
  - 12 Timers 16 bit và 2 timers 32 bit có hỗ trợ encoder.
  - 2 Watchdog timers, RTC (Real Time Clock).
  - 82 I/Os, 2 CAN, 3 I2C, 3 SPI 42Mbits/s, 2 I2S, 4 USART, 2 UART 10.5 Mbits.
  - Ngoài ra còn hỗ trợ DMA, 1 USB OTG FS và 1 USB OTG FS/HS, Ethenet, camera.



Package STM32F407VGTx



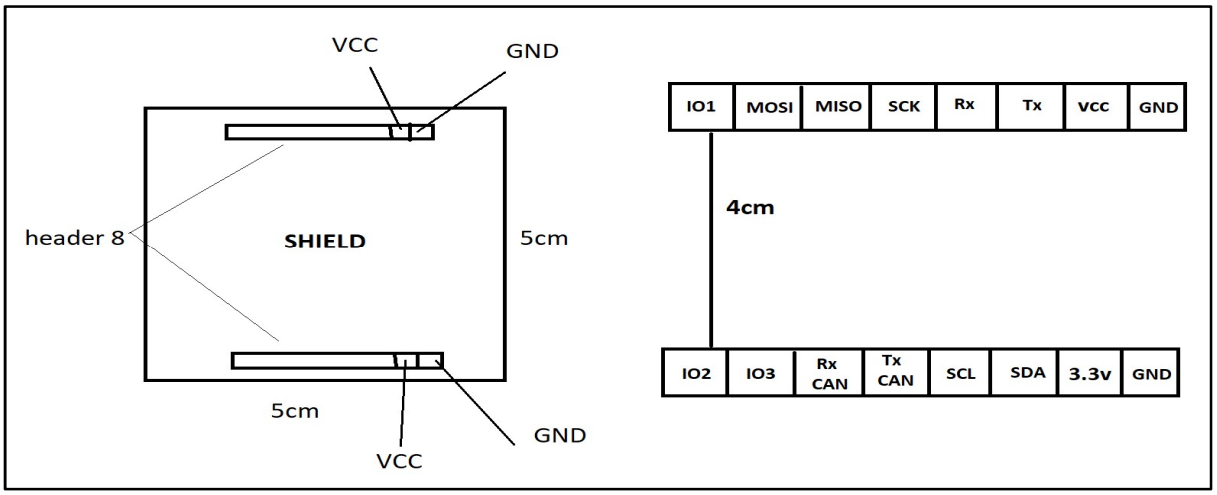
Chức năng dòng STM32F407VGT6

3. Giới thiệu board phát triển STM32F4

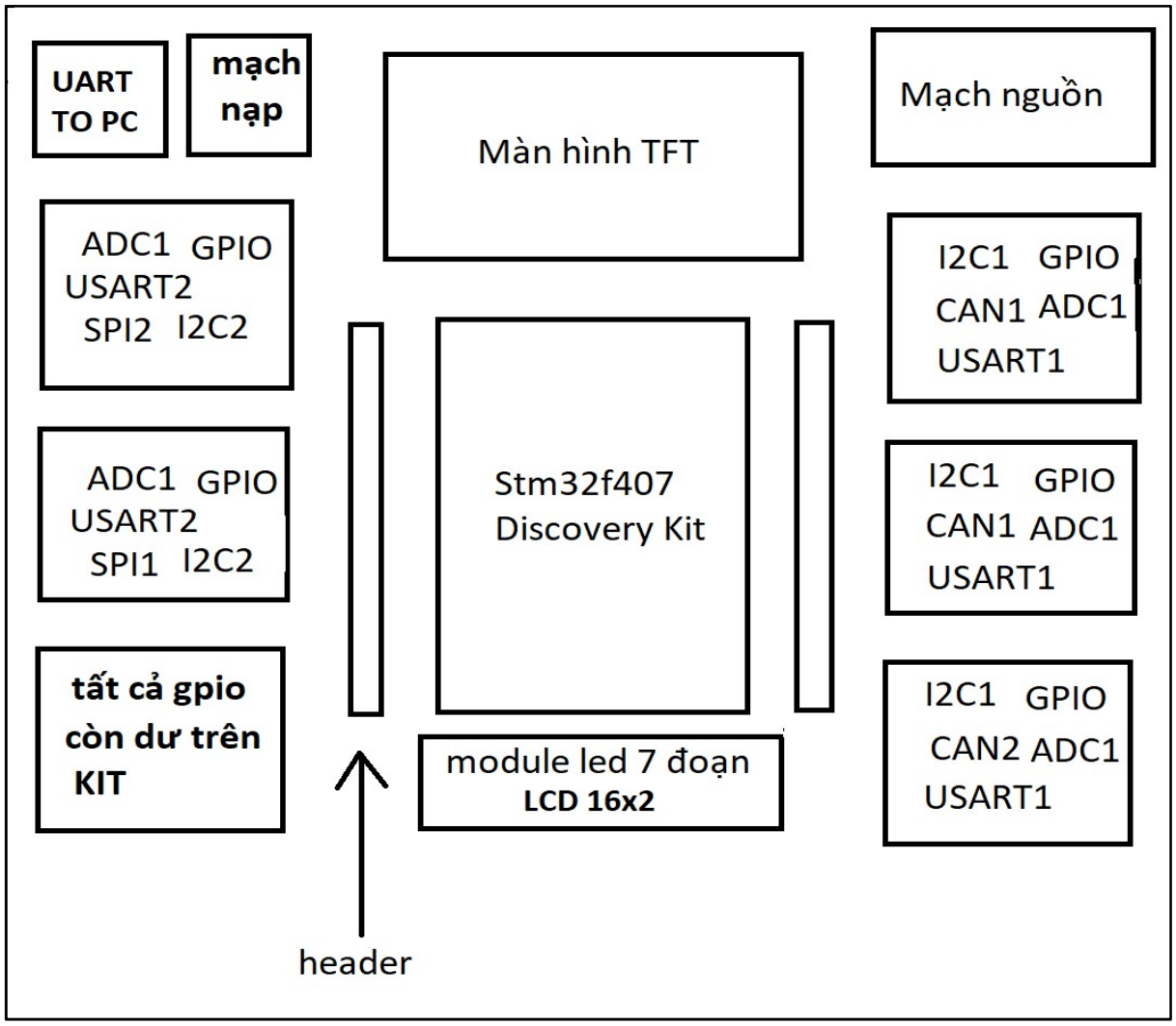
Board phát triển STM32F4 được thiết kế nhằm tối ưu hóa chức năng của Kit STM32F4, hỗ trợ người dùng trong việc tiếp cận lập trình nhúng, sử dụng ngoại vi theo chức năng, đào sâu nghiên cứu khai thác các chức năng của dòng vi điều khiển ARM.

Board phát triển STM32F4 được thiết kế bao gồm một board mạch mẹ, các shield module ngoại vi tích hợp thiết kế theo chức năng.

Board mạch mẹ được thiết kế bao gồm các liên kết giữa board STM32F407 Discovery và các kiến trúc ngoại vi, thiết kế header và jumper giúp người dùng có thể dễ dàng tháo lắp các shield module trên đó, tinh gọn board mạch, tránh cồng kềnh, thuận tiện cho việc sửa chữa.



Cấu trúc phần cứng liên kết giữa board mẹ và shield ngoại vi
(Cấu trúc áp dụng cho các shield 1,2,3,4 - shield 5,6 được thiết kế chuyên biệt)



Sơ đồ các module ngoại vi của Board phát triển STM32F4

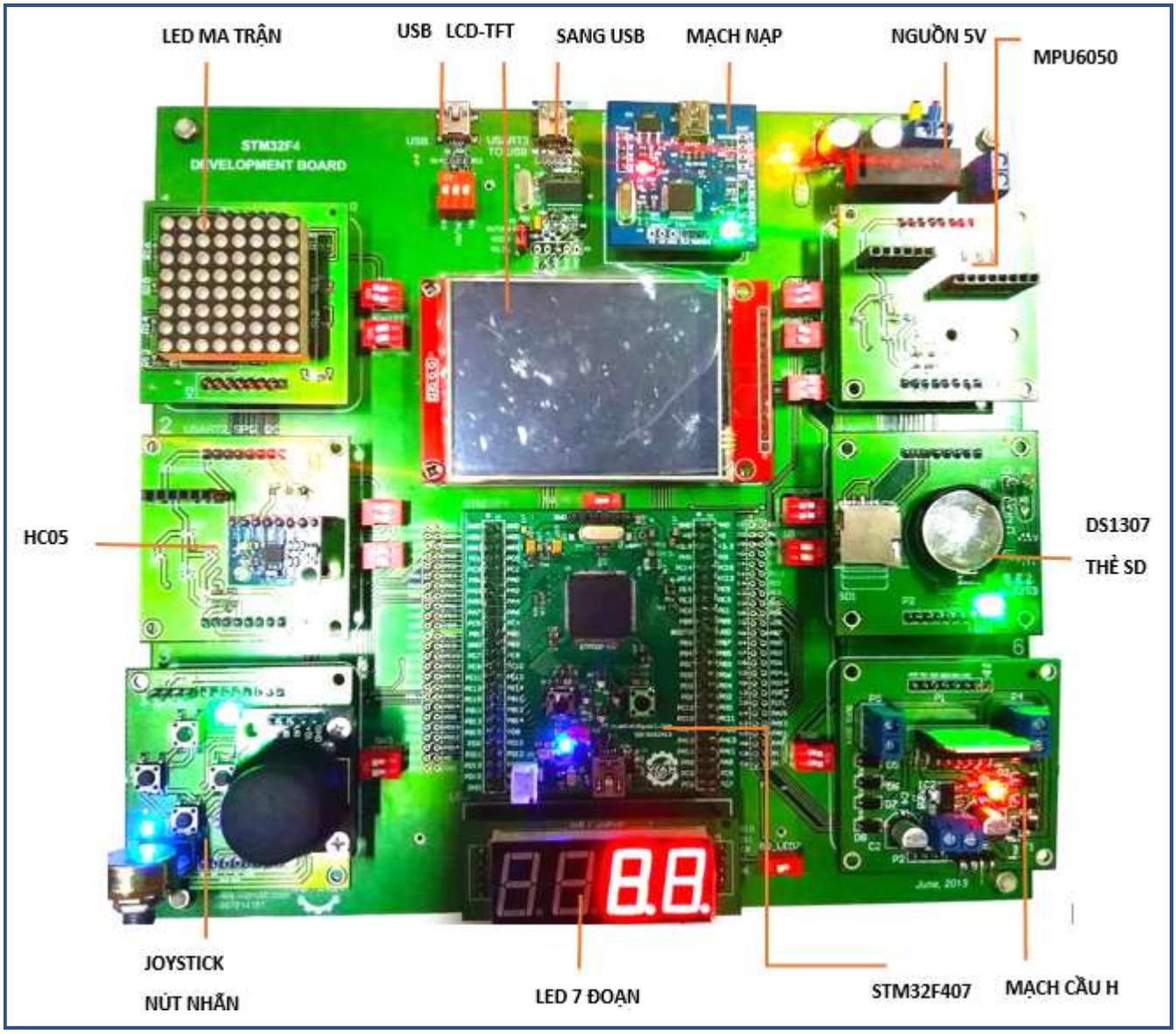
3.1 CẤU TRÚC BOARD PHÁT TRIỂN ARM STM32F4:

3.1.1 Các shield ngoại vi tích hợp:

Board phát triển STM324 bao gồm các shield:

- Shield LED 7seg x4.
- Shield LED Matrix 8x8.
- Màn hình TFT LCD SPI.
- Shield ADC Joystick: ADC biến trở, cảm biến nhiệt độ, độ ẩm, cảm biến joystick.
- Shield IC realtime DS1307 (I2C) + SD Card.
- Shield cảm biến gia tốc MPU6050 + Bluetooth.
- Shield Hbridge dual driver.
- Giao tiếp: chuẩn USB và UART to PC.

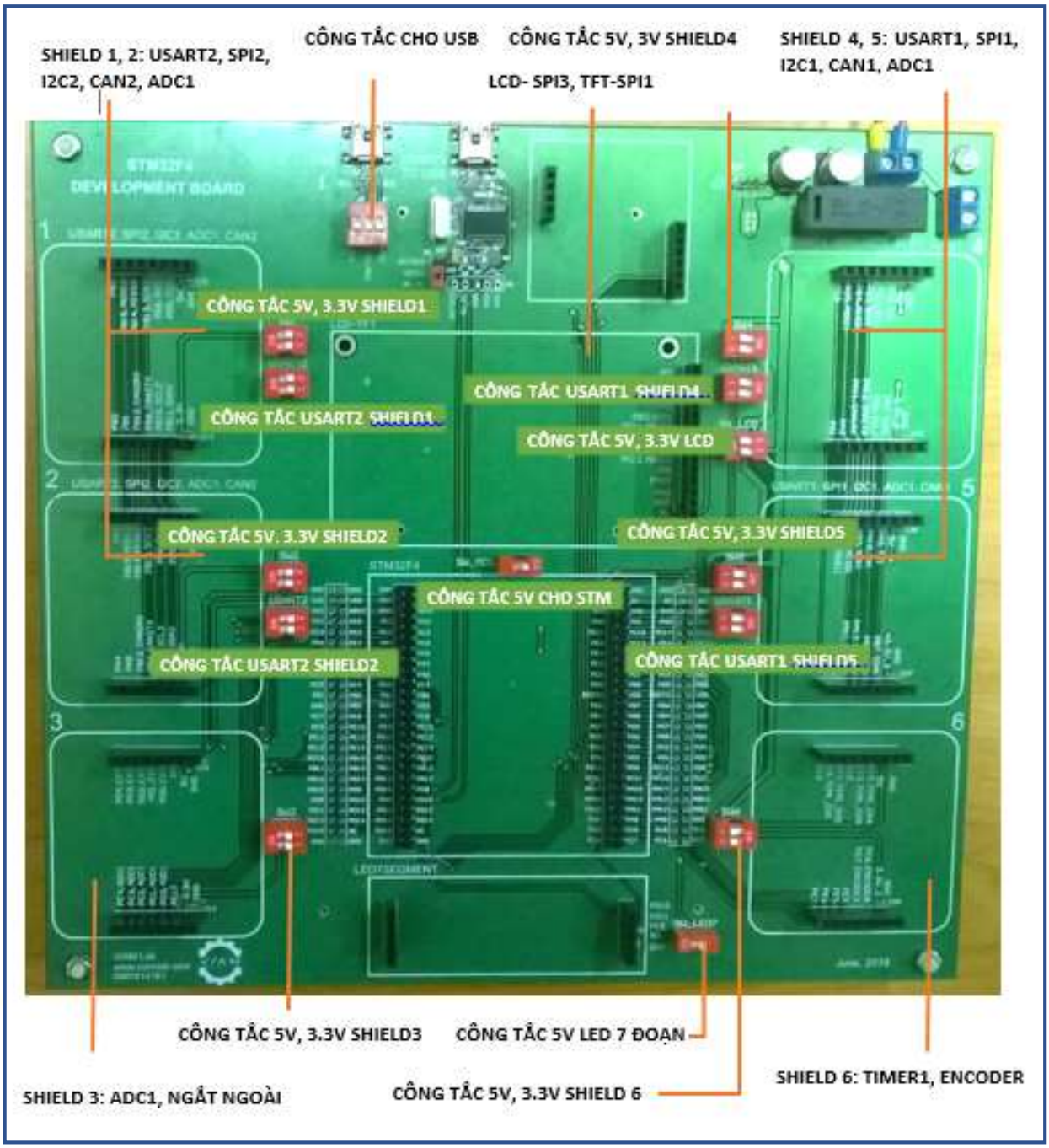
3.2 Bố trí board phát triển:



Bố trí Board phát triển STM32F4

3.3 Nguồn cấp cho board phát triển:

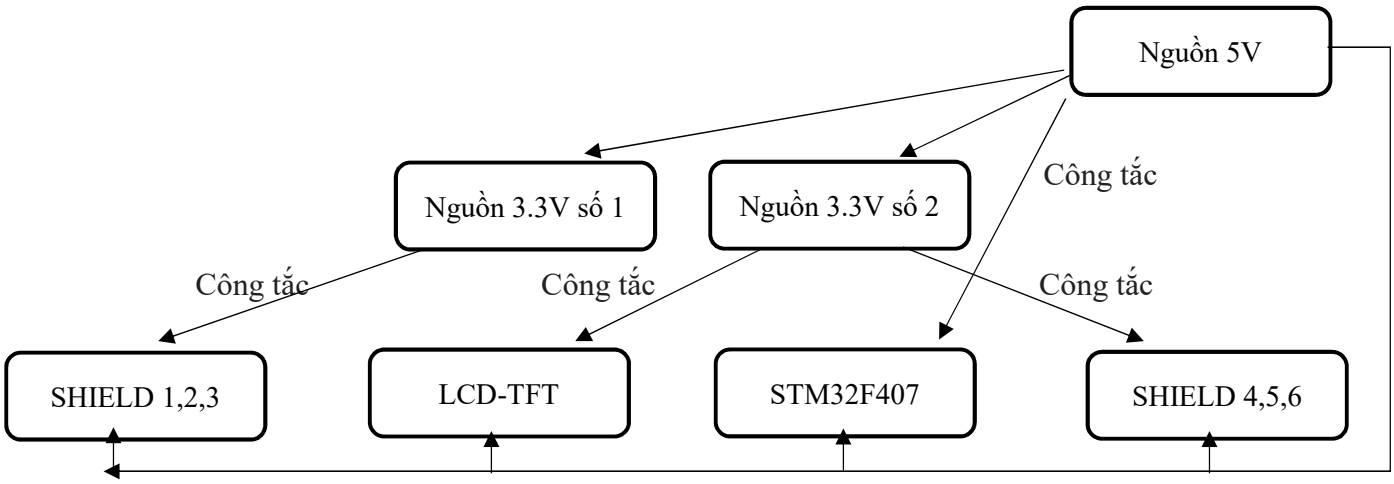
Board phát triển được thiết kế sử dụng nguồn ngoài ổn định áp, ở mỗi khu vực shield có công tắc nguồn cho từng shield (cấp nguồn ngoài cho cầu H), công tắc chuyển đổi UART cũng được tích hợp để tránh trường hợp trùng lắp truyền nhận UART trên các Shield có tích hợp chức năng UART.



Công tắc nguồn cho từng shield, công tắc chuyển uart

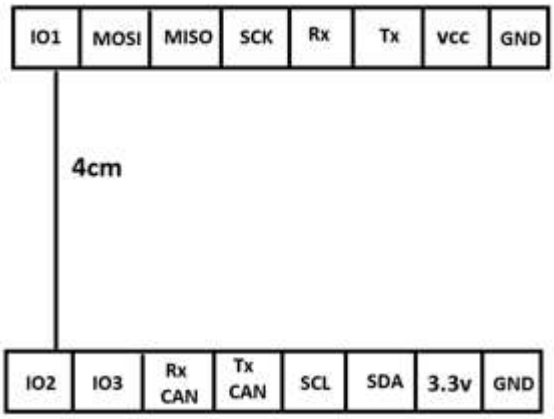
Lưu ý: Các công tắc nguồn có số 1 là 5V, số 2 là 3V. Công tắc USART có số 1 là TX, số 2 là RX.

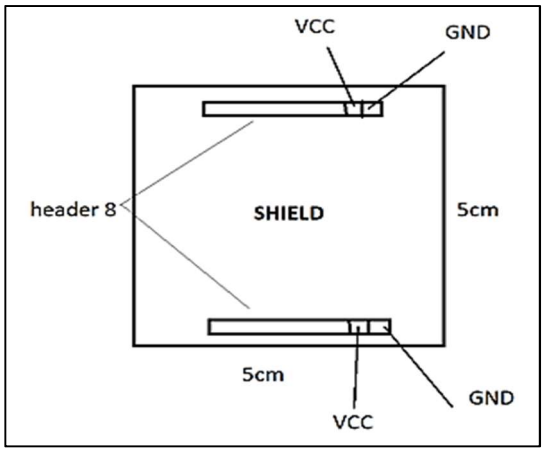
Phần nguồn điện: Nguồn 5V từ ngoài và từng nguồn 5V từ USB là tách biệt với nhau.



3.4 Cấu hình chân chức năng:

Các Pin của vi điều khiển STM32F4 trung tâm được phân chia đến từng Shield theo chức năng được tích hợp trong từng Shield, kèm theo chân nguồn 5V-3.3V-GND theo một chuẩn nhất định. (2 hàng – 8 pin - chiều nhìn thẳng vào trong) . Do đó, các shield cũng chuẩn pin có thể thay đổi vị trí cắm trên board mẹ => Tạo sự linh hoạt cho board phát triển và khả năng mở rộng sau này.

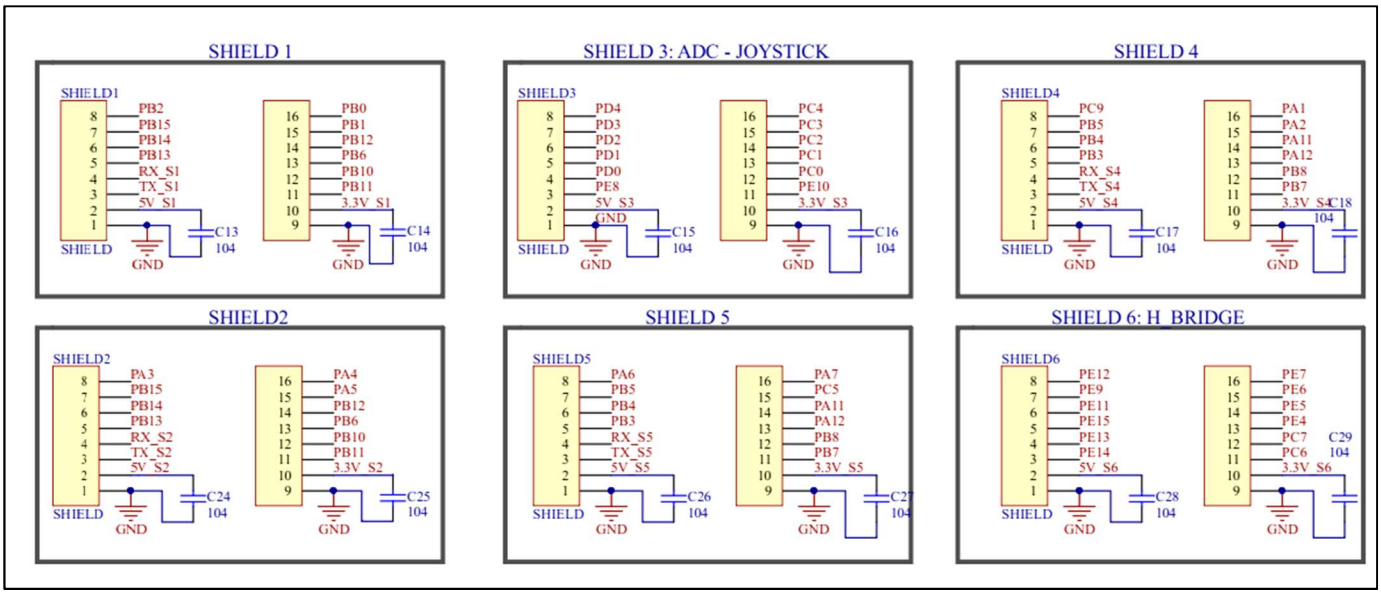


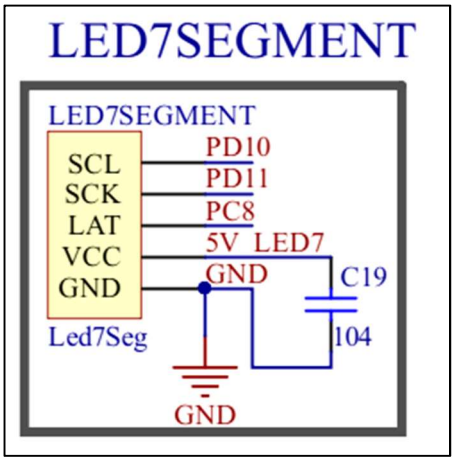


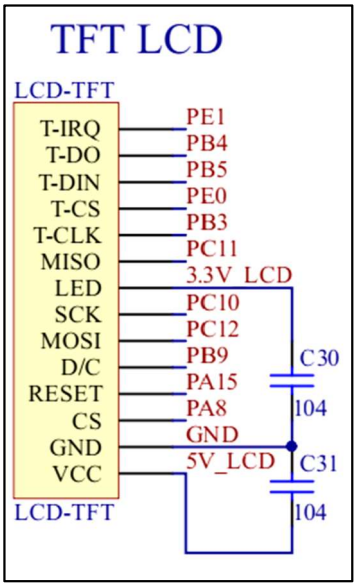
Shield1	**Hàng trên**	**PB2**	**PB15**	**PB14**	**PB13**	**PD6**	**PD5**	**5V**	**GND**
Shield2	**Hàng dưới**	**PB0**	**PB1**	**PB12**	**PB6**	**PB10**	**PB11**	**3.3V**	**GND**
Shield3	**Hàng trên**	**PD4**	**PD3**	**PD2**	**PD1**	**PD0**	**PE8**	**5V**	**GND**
	Hàng dưới	**PC4**	**PC3**	**PC2**	**PC1**	**PC0**	**PE10**	**3.3V**	**GND**
Shield4	**Hàng trên**	**PC9**	**PB5**	**PB4**	**PB3**	**PA10**	**PA9**	**5V**	**GND**
Shield5	**Hàng dưới**	**PA1**	**PA2**	**PA11**	**PA12**	**PB8**	**PB7**	**3.3V**	**GND**
Shield6	**Hàng trên**	**PE12**	**PE9**	**PE11**	**PE15**	**PE13**	**PE14**	**5V**	**GND**
	Hàng dưới	**PE7**	**PE6**	**PE5**	**PE4**	**PC7**	**PC6**	**3.3V**	**GND**

Bảng cấu hình chân trên từng Shield

Schematic cấu hình chân trên board mẹ







❖ ***Giao tiếp với PC thông qua chuẩn USB và UART.***

UART-Ta dùng mạch chuyển USB-USART PL2303 với:

- PD8 - TX
- PD9 - RX.

USB - Sử dụng:.

- PA11 - USB DM
- PA12 - USB DP

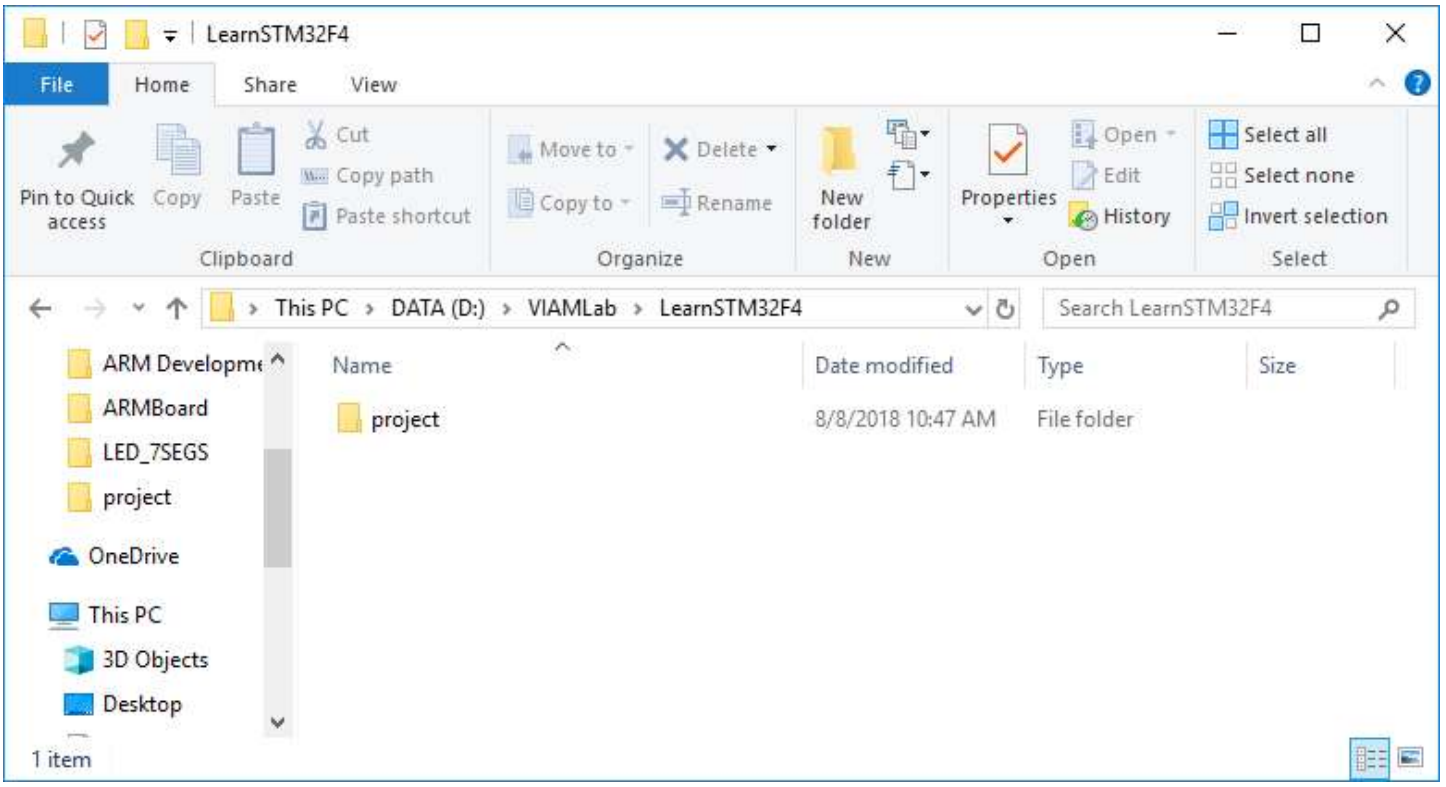
Phần 2: LẬP TRÌNH ARM STM32F4

1. Tạo Project sử dụng thư viện Standard Peripheral Libraries (STD) trên Keil C ARM.

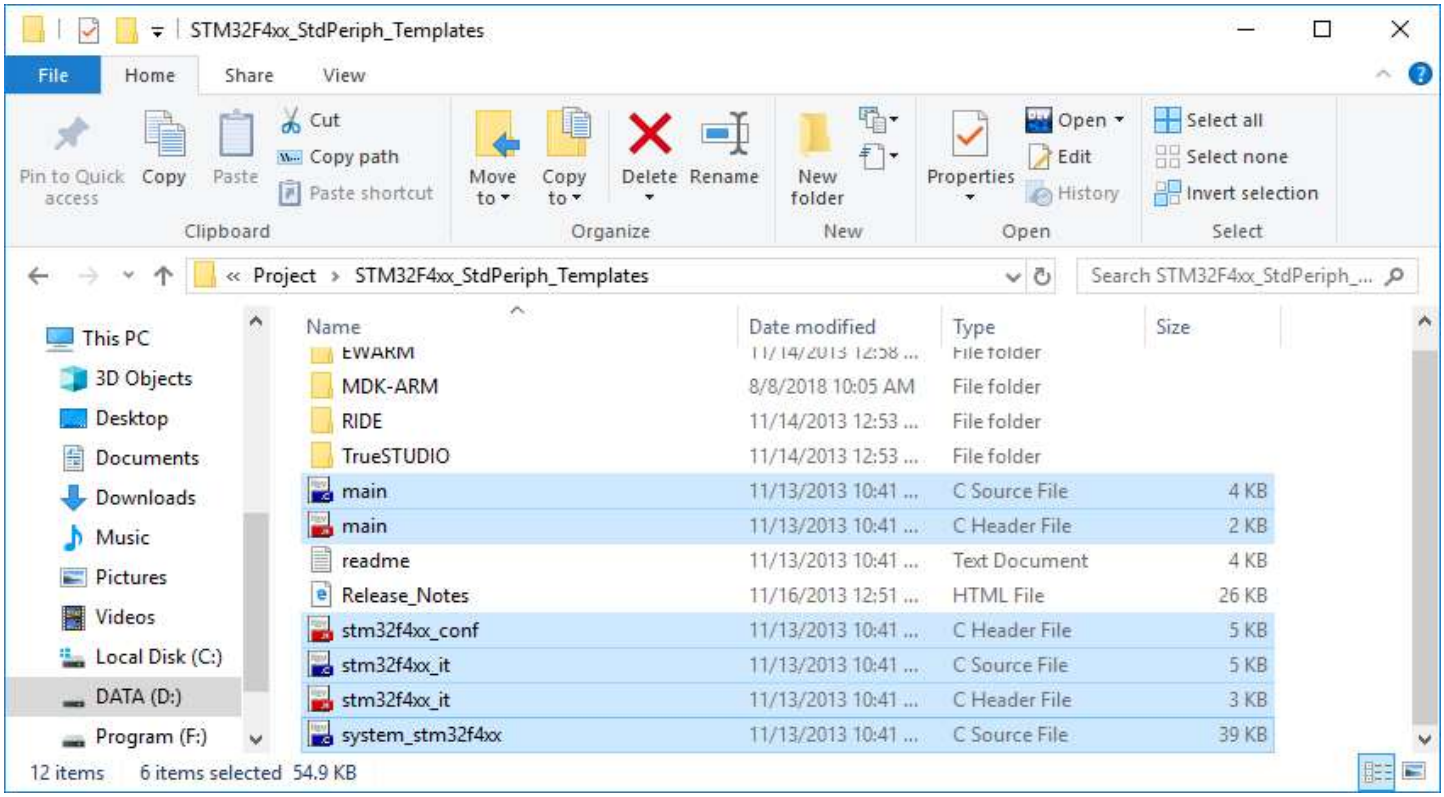
Standard Peripheral Libraries (STD) là một trong những thư viện hỗ trợ lập trình dòng ARM STM32 của ST. STD gồm nhiều thư viện C cho các ngoại vi, phù hợp với người có kiến thức lập trình C tốt. STD hỗ trợ hầu hết các ngoại vi ngoại trừ các ngoại vi phức tạp như USB/TCP-IP/Graphics/Touchsensor. STD không hỗ trợ các dòng chip STM32 L0, L4, L7, người dùng có thể dùng các thư viện khác như HAL để lập trình các dòng chip này.

Tạo project

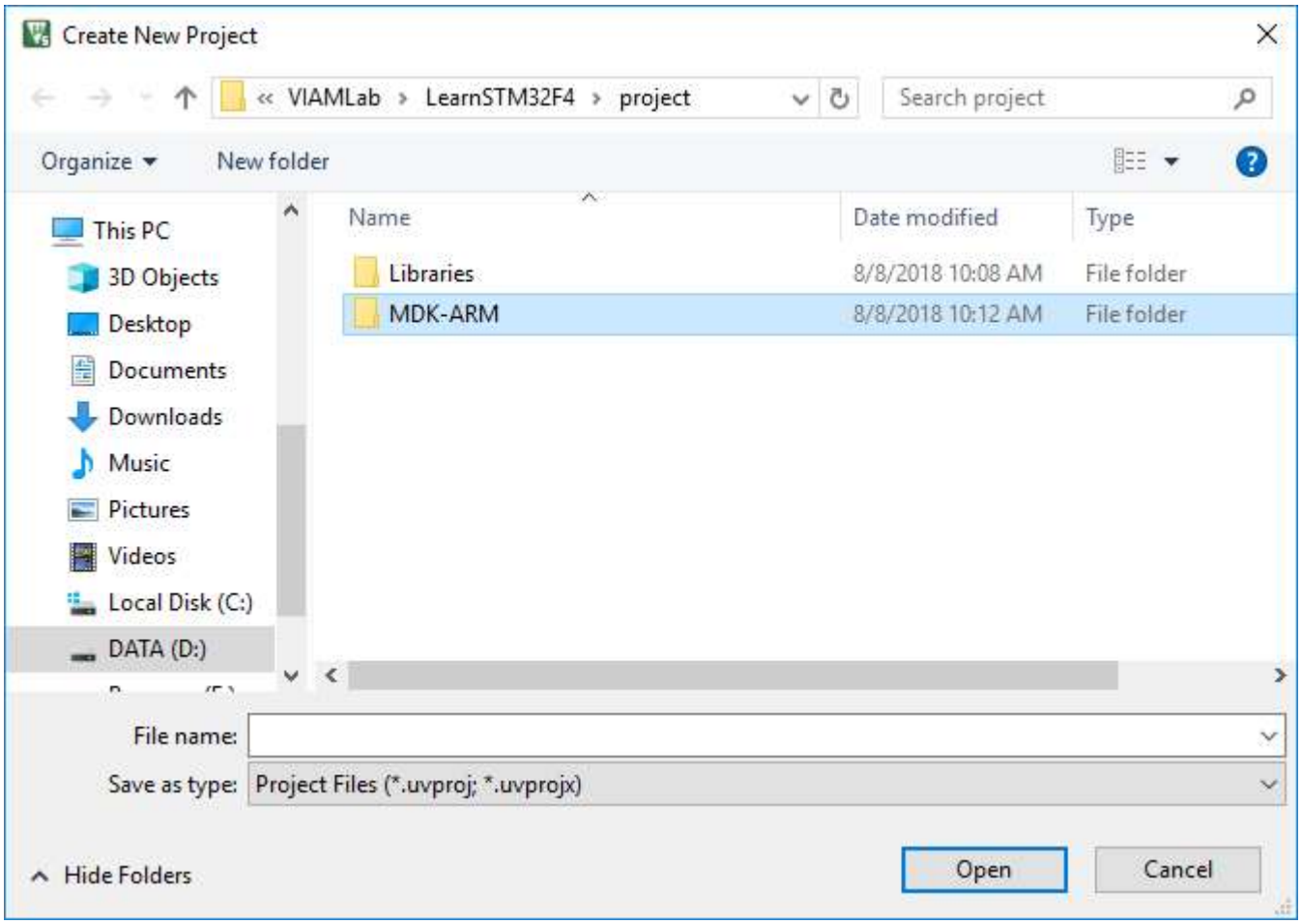
- Tạo folder chứa project và thư mục MDK-ARM trong folder này.



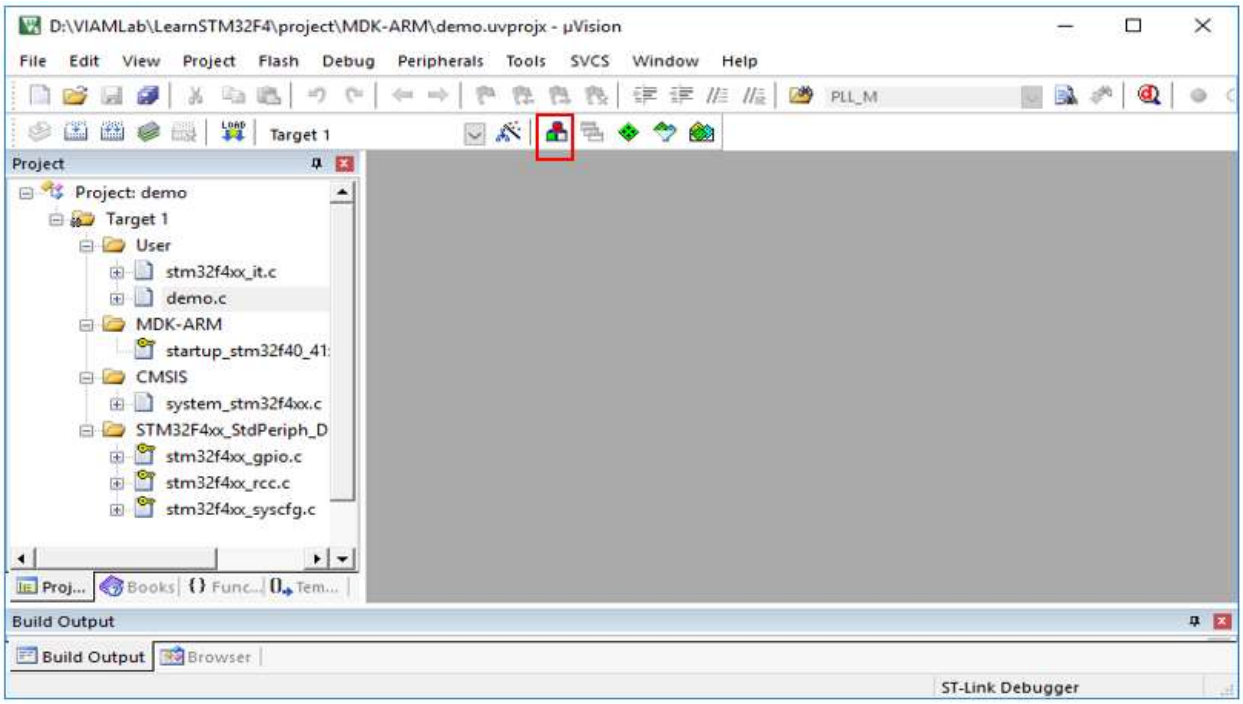
- Sao chép folder Libraries từ thư viện **STM32F4xx_DSP_StdPeriph_Lib_V1.3.0** sang thư mục vừa tạo.
- Sao chép các file: main.c, main.h, stm32f4xx_conf.h, stm32f4xx_it.c, stm32f4xx_it.h, system_stm32f4xx.c từ đường dẫn.:
  STM32F4xx_DSP_StdPeriph_Lib_V1.3.0\Project\STM32F4xx_StdPeriph_Templates trong project mẫu của ST sang project.



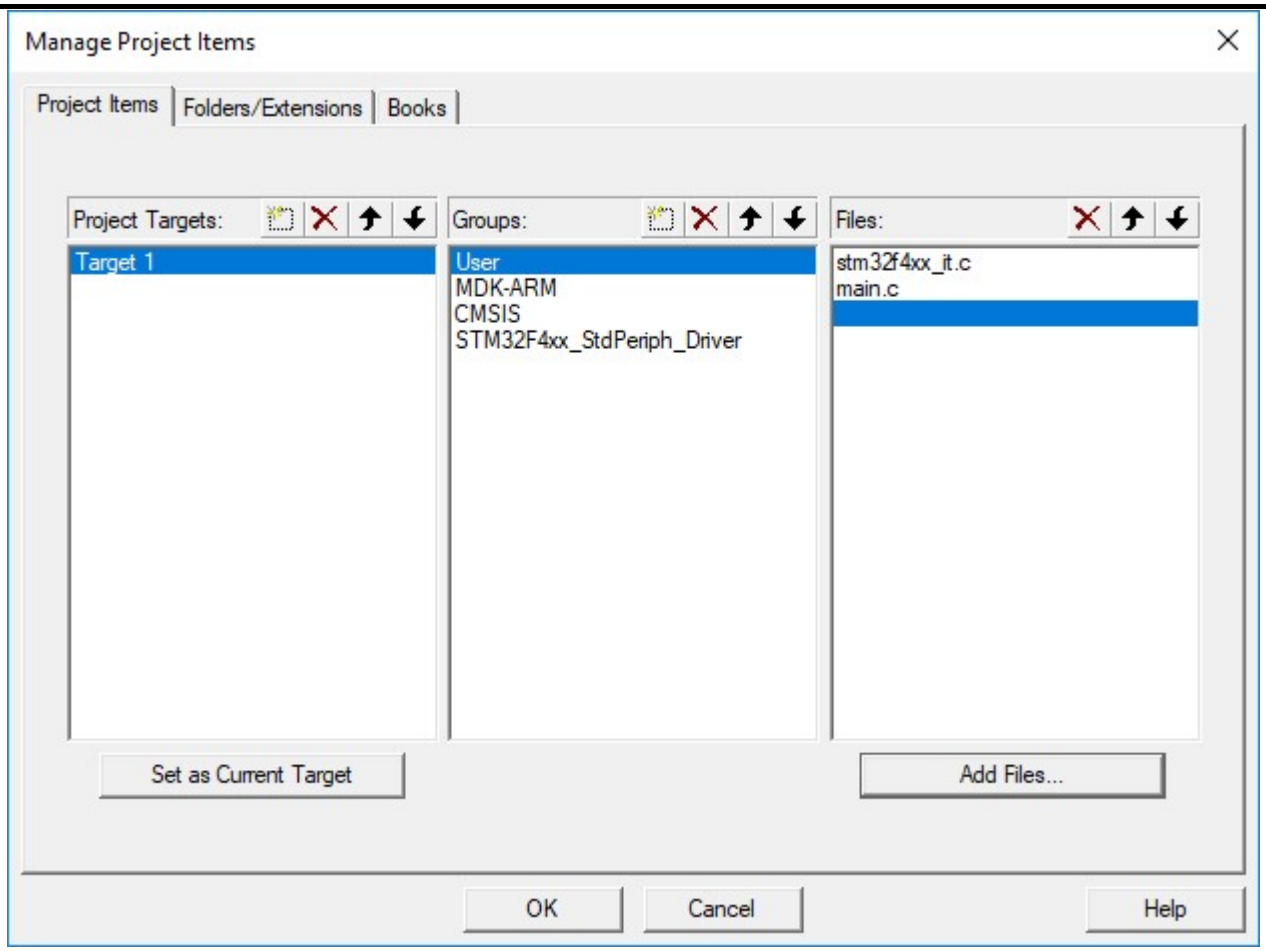
- Tạo project mới trong Keil C và lưu trong thư mục project \MDK-ARM.



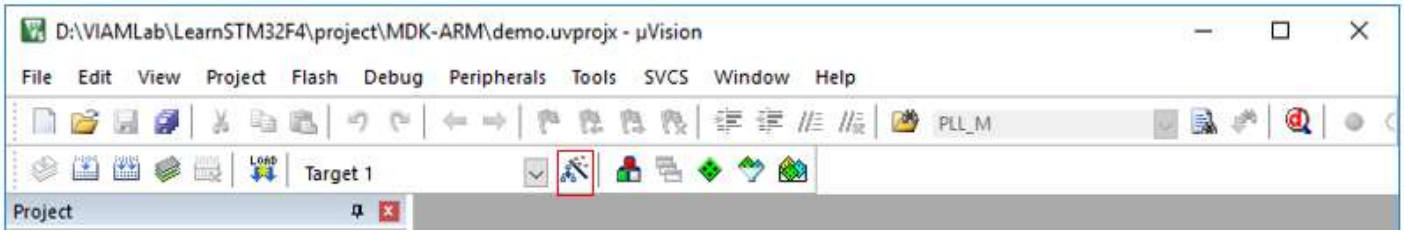
- Mở File Extensions: Bố trí các file source, thư viện theo chức năng



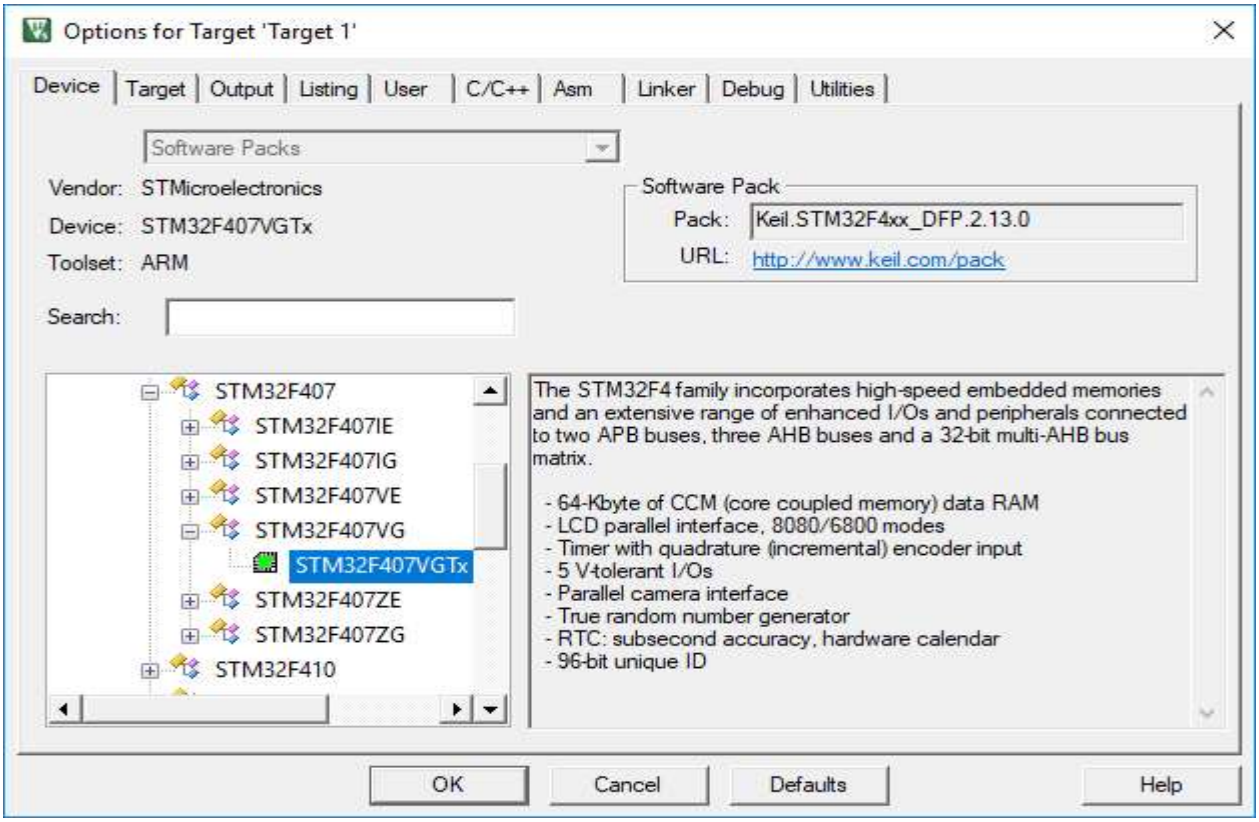
- Tạo các group và thêm các file cần thiết: (Tùy ý)
  - Group User thêm file stm32f4xx_it.c, main.c. File stm32f4xx_it.c là file chương trình ngắt của người dùng, main.c là chương trình chính.
  - Group MDK-ARM thêm file *startup_stm32f40_41xxx.s* trong đường dẫn: \project\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm.
  - Group CMSIS thêm file system_stm32f4xx trong đường dẫn \project.
  - Group STM32F4xx_StdPeriph_Driver thêm các file driver cần thiết trong \project\Libraries\STM32F4xx_StdPeriph_Driver\src



- Nhấn vào Option for target cài đặt cần thiết cho project.



- Trong tag Device chọn chip STM32F4VGTx



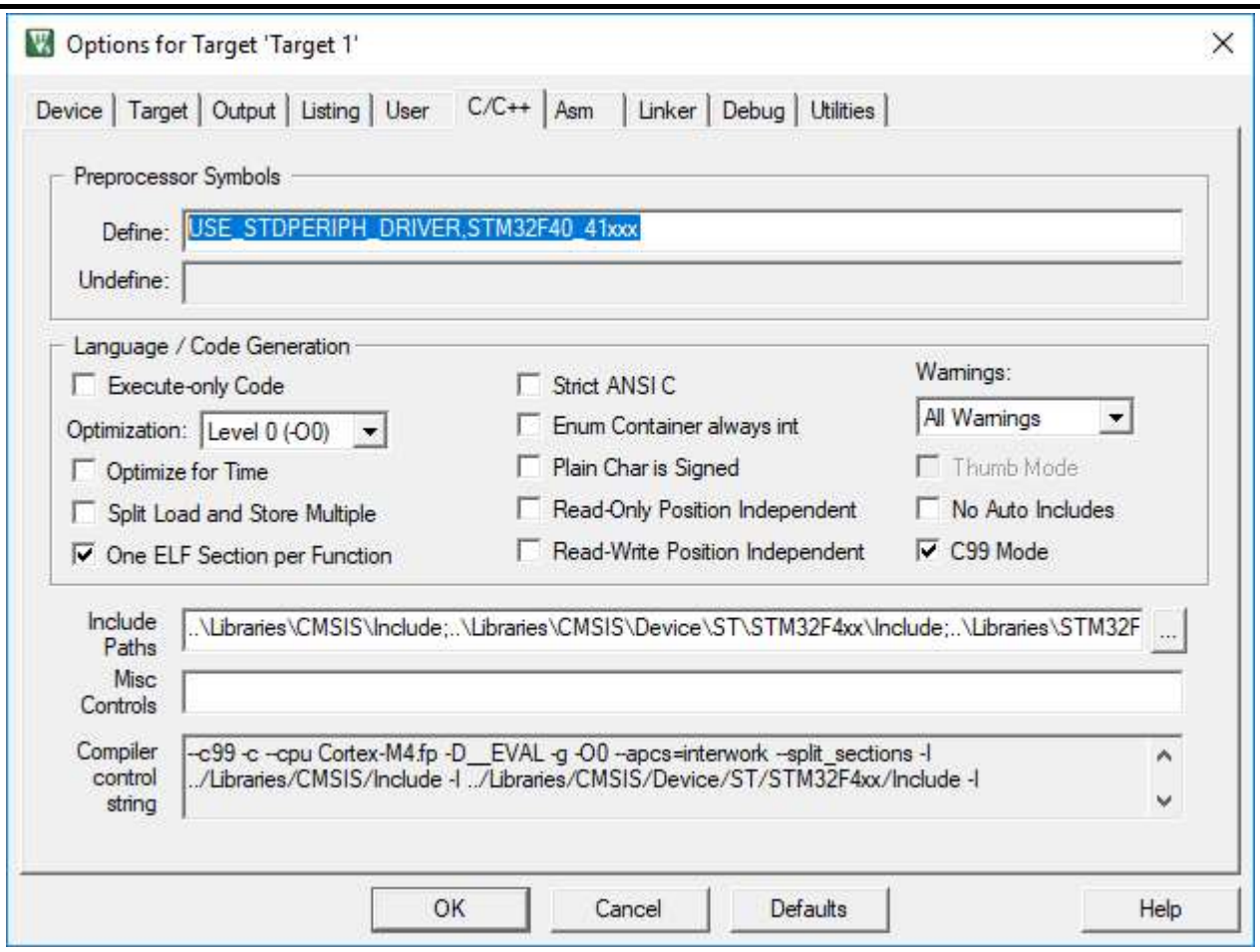
- Trong Tag C/C++:
  - Define sử dụng thư viện STD và chip STM32F4 bằng dòng USE_STDPERIPH_DRIVER,STM32F40_41xxx
  - Trong Include Paths ta thêm các đường dẫn đến các file header để trình biên dịch truy cập đến.

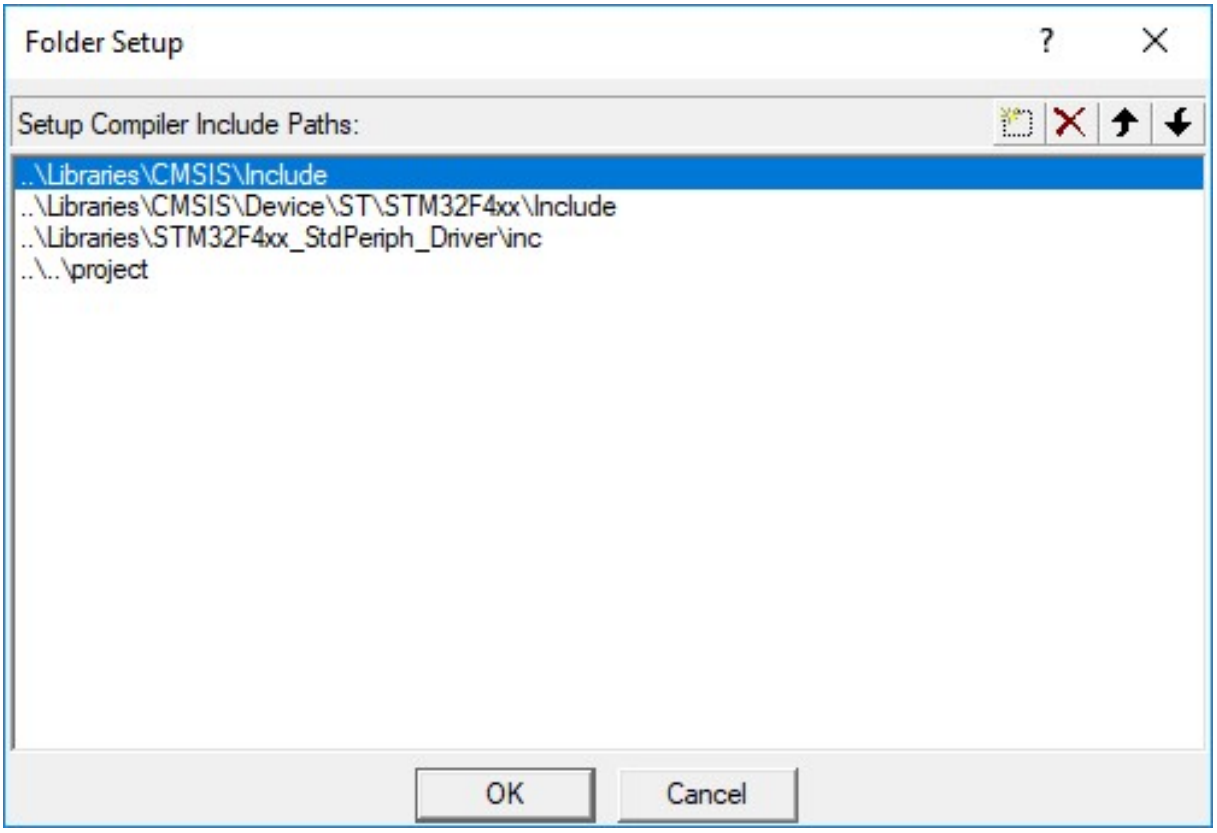
    ..\Libraries\CMSIS\Include

    ..\Libraries\CMSIS\Device\ST\STM32F4xx\Include

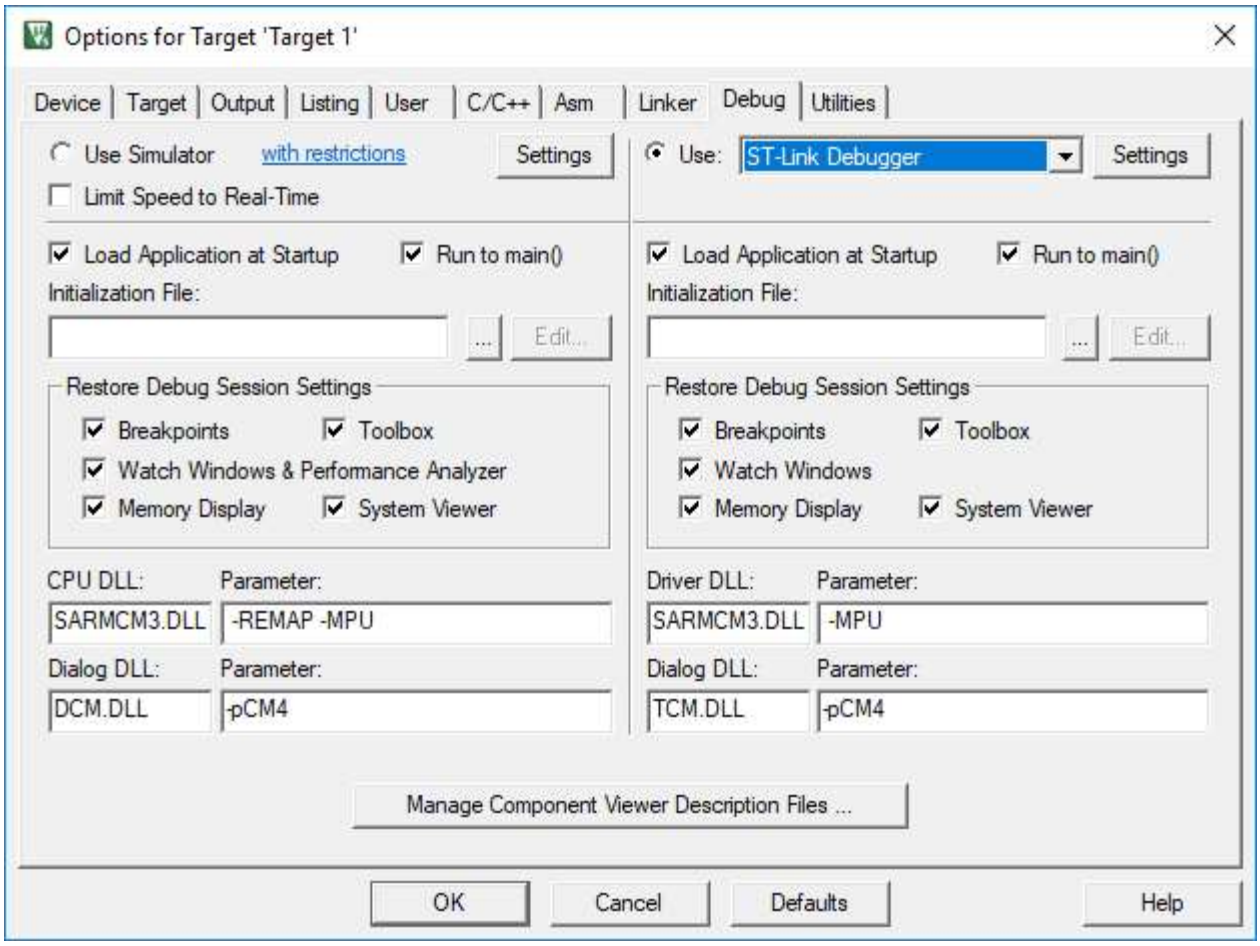
    ..\Libraries\STM32F4xx_StdPeriph_Driver\inc

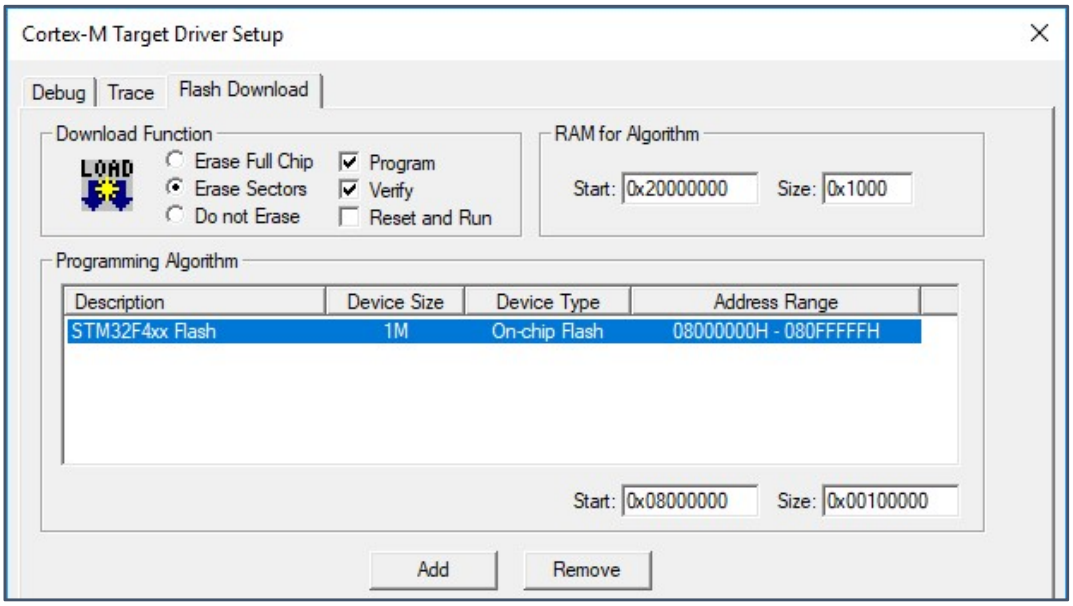
    ..\..\project


Options for Target 'Target 1'
Device
Target
Output
Listing
User
C/C++
Asm
Linker
Debug
Utilities
Preprocessor Symbols
Define:
USE_STDPERIPH_DRIVER,STM32F40_41xxx
Undefine:
Language / Code Generation
Execute-only Code
Optimization:
Level 0 (-O0)
Optimize for Time
Split Load and Store Multiple
One ELF Section per Function
Strict ANSI C
Enum Container always int
Plain Char is Signed
Read-Only Position Independent
Read-Write Position Independent
Warnings:
All Warnings
Thumb Mode
No Auto Includes
C99 Mode
Include Paths
..\Libraries\CMSIS\Include;..\Libraries\CMSIS\Device\ST\STM32F4xx\Include;..\Libraries\STM32F
Misc Controls
Compiler control string
--c99 -c --cpu Cortex-M4.fp -D__EVAL -g -O0 --apcs=interwork --split_sections -I
../Libraries/CMSIS/Include -I ../Libraries/CMSIS/Device/ST/STM32F4xx/Include -I
OK
Cancel
Defaults
Help

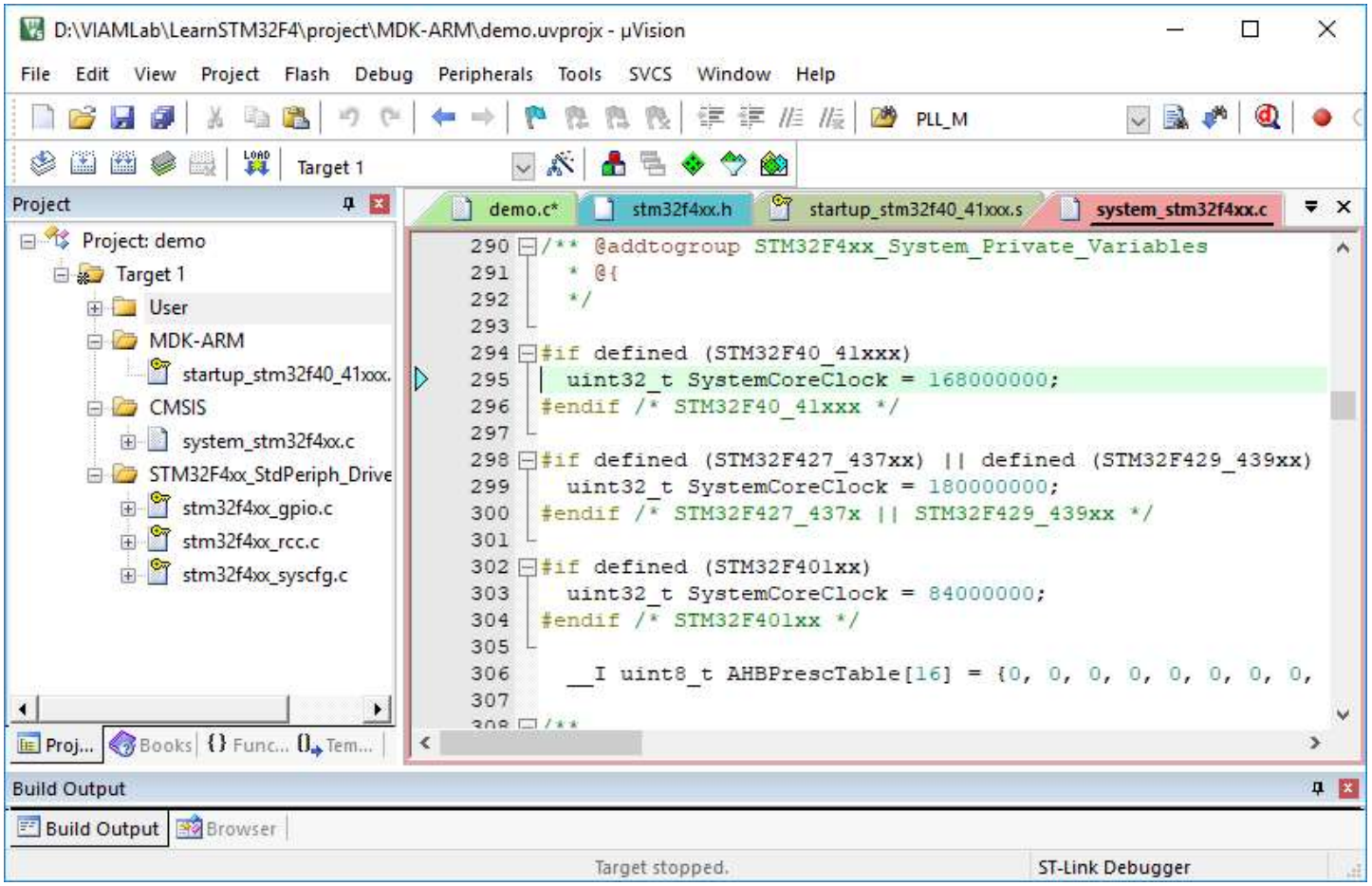

Folder Setup
Setup Compiler Include Paths:
..\Libraries\CMSIS\Include
..\Libraries\CMSIS\Device\ST\STM32F4xx\Include
..\Libraries\STM32F4xx_StdPeriph_Driver\inc
..\..\project
OK
Cancel

- Tag Debug chọn mạch nạp cho chip là ST-Link, chọn Setting và chọn Flash Download như hình, sau khi nạp codenếu muốn chương trình chạy ngay thì chọn Reset and Run, nếu muốn bấm nút reset thì chương trình mới chạy thì bỏ chọn mục này.





- Mở file system_stm32f4xx.c và sửa thông số PLL_M ở dòng 254 thành 8 để clock hệ thống đúng là 168MHz. Vì kit đang dùng thạch anh ngoài 8 MHz, không phải là 25MHz.



- Biên dịch code và tiến hành nạp code.

2. GPIO

2.1 Giới thiệu khái niệm:

GPIO (General-purpose input/output) là đầu vào, đầu ra sử dụng chung.

Dòng STM32 mỗi port có 16 chân IO.

Mỗi chân IO bên trong chip đều gắn thêm điện trở nội pull up và pull down.

2.2 Ứng dụng GPIO trong sáng tắt led

2.2.1 Sơ đồ chân Led

Sử dụng trực tiếp led trên board mạch, ở đây ta sử dụng led 12, 13, 14

2.2.2 Các hàm trong Code

Định nghĩa các hàm trong chương trình

```
#include "stm32f4xx.h"

GPIO_InitTypeDef GPIO_InitStructure; /* Đoi ten GPIO_InitTypeDef thanh
GPIO_InitStructure*/
void GPIO_Configuration(void);
void Delay(_IO uint32_t nCount);
```

Chương trình chính: Vòng lặp vô hạn sẽ đảo bit chân 12 13 14 với thời gian delay là 1 giây.Các chân này ta nối với đèn led

```
int main(void)
{
 GPIO_Configuration();
 while (1)
   {
  GPIO_ToggleBits(GPIOD,GPIO_Pin_12);/*Dao bit chan 12*/
 Delay(10000000);
 GPIO_ToggleBits(GPIOD,GPIO_Pin_13);/*Dao bit chan 13*/
 Delay(10000000);
 GPIO_ToggleBits(GPIOD,GPIO_Pin_14);/*Dao bit chan 14*/
 Delay(10000000);
}
}
```

Trong hàm main ta đã gọi hàm GPIO_Configuration() để khai báo các chân

```
void GPIO_Configuration(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD,ENABLE); /*Bat clock khoi D*/
  /* Configure PD12 PD13 in output pushpull mode */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13| GPIO_Pin_14 | GPIO_Pin_15; /*Khai bao dung chan PD12->PD15*/
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; /*Xac dinh day la chan xuat*/
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; /*/Chon che do pushpull */
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;/*Chon toc do dau ra*/
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;/*Chon che do khong dung dien tro keo len*/
  GPIO_Init(GPIOD, &GPIO_InitStructure) ;/*truyen cac doi so xuong thiet lap thanh ghi cho phan cung*/
}
```

Hàm delay:

```
void Delay(_IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}
```

3. Timer

3.1 Giới thiệu khái niệm cơ bản

Timer trong STM32F4 có rất nhiều chức năng chẳng hạn như bộ đếm counter, PWM, input capture ngoài ra còn một số chức năng đặt biệt để điều khiển động cơ như encoder, hall sensors.

Trong STM32F4 Timer 1 và timer 8 có cùng tần số với xung clock hệ thống, các timer còn lại chỉ bằng một nửa. Riêng timer 2 và timer 5 là timer 32bit, các timer còn lại 16bit. Timer 6 và timer 7 là 2 timer với các chức năng cơ bản không giao tiếp được với môi trường.

3.2 Code ví dụ

Đoạn code dưới đây tương tự như đã giải thích ở phần GPIO.
Chương trình chính sẽ đảo set và reset bit chân 14 với thời gian delay là 1 giây

```
#include "stm32f4xx.h"
GPIO_InitTypeDef GPIO_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
void GPIO_Configuration(void);
void TIMbase_Configuration(void);
void  Delay(_IO  uint32_t  nCount);
volatile int32_t debug;
int main(void)
{
 GPIO_Configuration();
 TIMbase_Configuration();
 while (1)
 {
     GPIO_ResetBits(GPIOD,GPIO_Pin_14);
  Delay(1000);
         GPIO_SetBits(GPIOD,GPIO_Pin_14);
  Delay(1000);
  }
}
void GPIO_Configuration(void)
{
 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE); /*Bat
clock khoi D*/
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14 | GPIO_Pin_15| GPIO_Pin_13|
GPIO_Pin_12; /*Khai bao dung chan PD12->PD15*/
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; /*Chon mode out*/
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; /*Chon che do pushpull*/
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; /*Chon toc do dau
ra*/
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; /*Chon che do khong
dung dien tro keo len*/
```

```
  GPIO_Init(GPIOD, &GPIO_InitStructure); /*truyen cac doi so xuong thiet lap thanh ghi cho phan cung*/
}
```

* Khởi tạo timer:

- Ở dòng khởi tạo timer prescaler ta cần tần số clock timer là 1 Mhz.Tần số cao nhất của timer 4 là 84 MHz nên ta có SystemCoreClock/2)/1000000)-1 = 83.

- Vậy cứ 1us counter sẽ đếm 1 lần nên ta cần 1000 xung.Dòng thứ 2 ta khai báo TIM_TimeBaseStructure.TIM_Period = 1000 – 1 = 999 do ta bắt đầu đếm từ 0.

```
void TIMbase_Configuration(void)
{
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
  /* Time base configuration */
TIM_TimeBaseStructure.TIM_Prescaler((SystemCoreClock/2)/1000000)-1;     // frequency = 1000000
  TIM_TimeBaseStructure.TIM_Period = 1000 - 1;/*so xung trong 1 chu ky*/
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;/* chon mode counter dem tang*/
  TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
TIM_ITConfig(TIM4,TIM_IT_Update,ENABLE);/*thiet lap ngat khi tran bo nho co thong so TIM_IT_Update*/
  TIM_Cmd(TIM4, ENABLE);
   NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

void Delay(_IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}
```

- Chương trình timer : stm32f4xx_it.c

```
#include "stm32f4xx_it.h"
void NMI_Handler(void)
{
}
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {}
}
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {}
}
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {}
}
void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {}
}
void DebugMon_Handler(void)
{}
void SVC_Handler(void)
{}
void PendSV_Handler(void)
{}
void SysTick_Handler(void)
{}
// Chương trình timer
void TIM4_IRQHandler(void)
{
  static uint32_t time=0;

if (TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET)
  {
    if(++time>1000)
    {
```

```
                    GPIO_ToggleBits(GPIOD,GPIO
                    _Pin_15); Delay(100);
                    GPIO_ToggleBits(GPIOD,GPIO_
                    Pin_12); Delay(100);
                    GPIO_ToggleBits(GPIOD,GPIO_
                    Pin_13); Delay(100);

          time = 100;
   }
  TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
 }
}
```

4. Ngắt ngoài EXTI

4.1 Khái niệm

Ngắt có nghĩa là một sự gián đoạn chương trình chính, ngưng thực thi ở chương trình chính mà nhảy đến một địa chỉ nào đó để giải quyết vấn đề ở đó, sau khi xử lý xong sẽ quay về chương trình chính ngay tại nơi mà ban nãy nó đã thoát khỏi chương trình chính. Ngắt ngoài thì ta sử dụng một tín hiệu bên ngoài để tạo ra ngắt, chẳng hạn input trên GPIO.

Những pin có cùng số sẽ kết nối cùng một line.VD: PA0 PB0 PC0.

Chú ý: không thể dùng cùng 1 lúc 2 pin trong cùng một line khi ngắt.VD: PA0 và PA5 khác line nên có thể dùng trong hàm ngắt cùng 1 lúc.

4.2 Code ví dụ

```
#include
"stm32f4xx.h" int a;
GPIO_InitTypeDef   GPIO_InitStructure;
NVIC_InitTypeDef  NVIC_InitStructure;
EXTI_InitTypeDef  EXTI_InitStructure;

void GPIO_Configuration(void);
void EXTILine0_Config(void);
void Delay(_IO uint32_t nCount);
int main(void)
{
     a=1;
  GPIO_Configuration();
  EXTILine0_Config();
     GPIO_SetBits(GPIOD,GPIO_Pin_13);
  while (1)
  {
}
}

void GPIO_Configuration(void)
```

```
{
  GPIO_InitTypeDef GPIO_InitStructure;
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

  GPIO_InitStructure.GPIO_Pin =
GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOD, &GPIO_InitStructure);
}
```

- Hàm khai báo ngắt

```
void EXTILine0_Config(void)
{
  /* bat clock GPIOA */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  /*bat clock SYSCFG */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

  /* Configure PA0 pin as input floating */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* Connect EXTI Line0 to PA0 pin */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
  /* Configure EXTI Line0 */
/* PD0 is connected to EXTI_Line0 */
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
/* Interrupt mode */
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
 /* Triggers on rising or falling edge or both */
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
/* Enable interrupt */
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
/* Add to EXTI *
```

```
  EXTI_Init(&EXTI_InitStructure);
```

- Để sử dụng được ngắt thì việc đầu tiên bạn cần khai báo với khối NVIC, khối này sẽ quản lý ngắt và độ ưu tiên các ngắt, cần khai báo tên ngắt EXTI0_IRQn với NVIC, với khai báo này thì hàm ngắt chắc chắn phải có tên là EXTI0_IRQHandler, hàm này không có đối số truyền và nhận nên trong stm32f4xx_it.c tên của nó là "void EXTI0_IRQHandler(void)" hàm này do bạn tạo chứ trình biên dịch không tự sinh ra. Bạn nên chú ý chữ "n" ở khai báo và "Handler" ở hàm ngắt còn "EXTI0_IRQ" tên chung.

```
  /* Enable and set EXTI Line0 Interrupt to the lowest priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
/* Set priority */
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

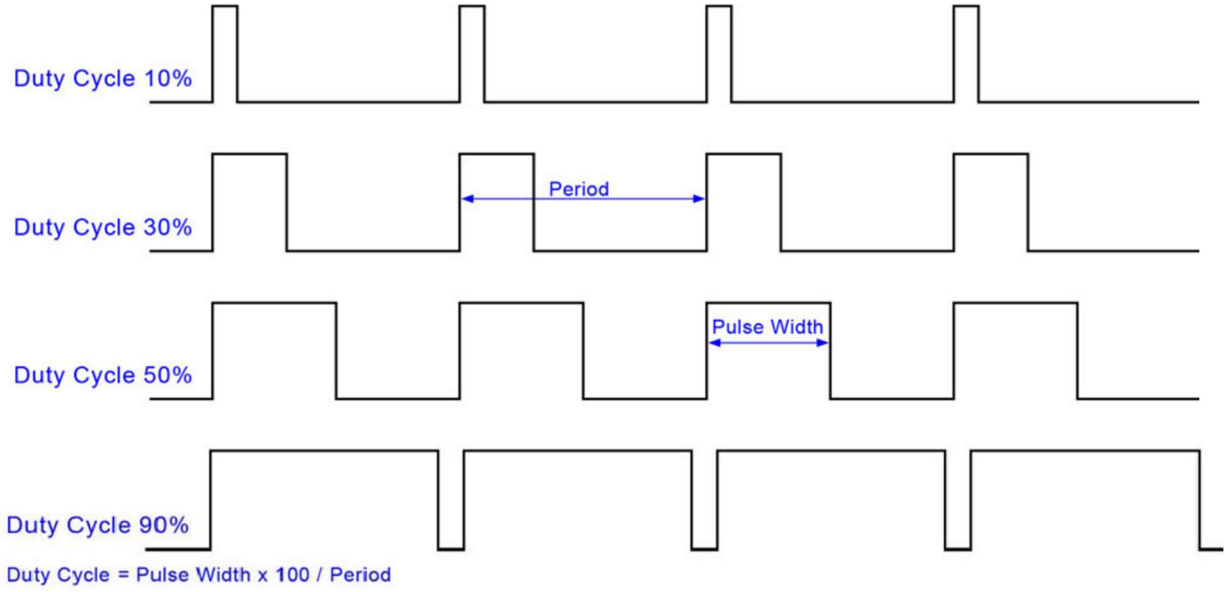
- Hàm ngắt đảo bit chân PD13.Hàm này có thể khai báo ở hàm main hoặc file stm32f4xx_it.c

```
void EXTI0_IRQHandler(void)
{
      a=~a;
if(a==1){
      GPIO_SetBits(GPIOD,GPIO_Pin_13);
}
else
{
      GPIO_ResetBits(GPIOD,GPIO_Pin_13);
}
EXTI->PR = EXTI_Line0;
  }
void Delay(__IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}
```

5. PWM

5.1 Giới thiệu cơ bản về PWM

PWM điều chế độ rộng xung (**Pulse-width modulation**) là ngoại vi phổ biến được hỗ trợ hầu hết trong các loại vi điều khiển, ứng dụng rộng rãi và nhất là trong lĩnh vực điều khiển động cơ. Đơn giản nhất để hiểu PWM tôi VD cho bạn như sau : cấu hình output cho chân GPIO điều khiển bóng LED, nếu bạn muốn làm sáng bóng LED theo hiệu ứng mờ hay tỏ, giải pháp đó là bạn sẽ cho bóng LED sáng tắt ở tần số cao sao cho mắt bạn mất khả năng phân tích sự sáng tắt của một bóng LED. Bạn muốn bóng LED sáng tỏ thì thời gian sáng sẽ nhiều hơn thời gian tắt, và ngược lại. Trên đây chỉ là ví dụ cơ bản nhất để bạn hiểu như thế nào là PWM, bạn vui lòng xem hình ảnh bên dưới để hiểu rõ hơn.



Như trên hình đã rất rõ

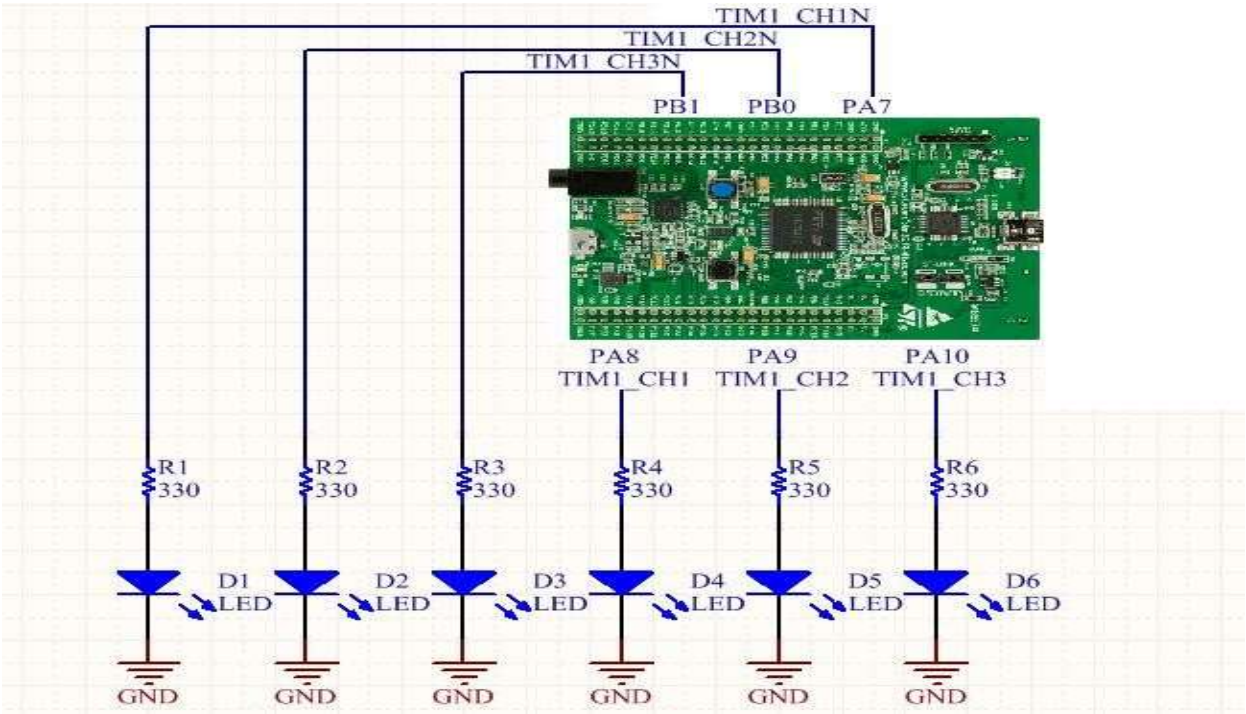
- Duty cycle là tỷ lệ phần trăm mức cao.
- Period là chu kỳ xung.
- Pulse width là giá trị mức cao so với period.

Dựa trên nguyên lý bên trên mà trong STM32 hay các loại vi điều khiển khác điều hỗ trợ bộ tạo độ rộng xung tự động mà bạn không phải tốn thời gian bật tắt chân IO, có thể thiết lập chu kỳ, tần số, độ rộng xung và một số chức năng đặc biệt. PWM thuộc khối timer.

5.2 Ứng dụng PWM

5.2.1 Ứng dụng trong điều chỉnh độ sáng đèn led

5.2.1.1 Kết nối sơ đồ chân như hình sau:



- Sơ lược nội dung của code:

Trong bài tôi thiết lập PWM1-2-3 và 3 kênh đảo của nó tổng cộng 6 kênh PWM xuất ra môi trường.

Dòng 25 : chọn sử dụng Mode AF (mode tính năng phụ trợ) cho chân IO, mode này không phải là mode In hay Out mà do ngoại vi đó tự động thiết lập In-Out.

Dòng 34 -> 39 bạn phải kết nối chân IO đến khối timer 1 cho đúng, bạn xem datasheet phần "Table 6. STM32F40x pin and ball definitions"

Dòng 49 : Chọn mode PWM1, trong timer có nhiều mode mà tùy theo ứng dụng để bạn lựa chọn mode cho phù hợp. Từ OC mà bạn thấy trong dòng khai báo là viết tắt của từ Output Compare.

Dòng 50 : Cho phép chân PWM hoạt động.

Dòng 51 : Chọn cực mức cao cho đầu ra, nếu bạn chọn mức thấp thì nó sẽ bị đảo ngược hình dạng của đầu ra, Chu kỳ, tần số đều như nhau.

Dòng 52 - 53 : tương tự như dòng 50-51 nhưng khai báo cho kênh đảo có nghĩa là chân được ký hiệu thêm chữ "N". Bạn cần lưu ý do vốn dĩ kênh PWM và kênh đảo của nó đã được định nghĩa trong chip là trạng thái đảo của nhau nên ở dòng 50-51 và 52-53 tôi khai báo giống nhau và nó sẽ tự đảo không cần bạn phải can thiệp đảo lại các cực của PWM kênh N.

Sau khi thiết lập xong các thông số bạn phải truyền biến cấu trúc xuống cho hàm TIM_OC1Init() để thiết lập cho kênh PWM1, vẫn giữ lại các thông số đó, tương tự truyền cho PWM2 và PWM3.

Dòng 58-61-64-66 : bật chức năng preload cho OC1, OC2 và OC3.

Bước cuối cùng trong hàm TIM_PWM_Configuration() bạn phải bật counter để bắt đầu hoạt động timer.

Ở dòng 70 do sử dụng timer1 là timer đặc biệt nên bạn phải thực hiện bước này. Sử dụng các timer khác trừ timer1 và timer8 thì bạn không cần bước này.

Quay lại chương trình chính từ dòng 12 -> 14 tôi gán giá trị vào thanh ghi chứa Pulse width của PWM1-PWM2-PWM3 tương đương với 10-50-90% duty cycle.

5.2.1.2 Code

chính Main.c

```
01  #include "stm32f4xx.h"
02
03  TIM_TimeBaseInitTypeDef    TIM_TimeBaseStructure;
04  TIM_OCInitTypeDef          TIM_OCInitStructure;
05  GPIO_InitTypeDef           GPIO_InitStructure;
06
07  void TIM_PWM_Configuration(void);
08
09  int main(void)
10  {
11    TIM_PWM_Configuration();
12    TIM1->CCR1 = 10 * 65535 / 100;  // 10% Duty cycle
13    TIM1->CCR2 = 50 * 65535 / 100;  // 50% Duty cycle
14    TIM1->CCR3 = 90 * 65535 / 100;  // 90% Duty cycle
15    while (1)
16    {}
17  }
18
19  void TIM_PWM_Configuration(void)
20  {
21    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
22    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOB,
ENABLE);
23
24    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10;
25    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
26    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
27    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
28    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
29    GPIO_Init(GPIOA, &GPIO_InitStructure);
30
31    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
32    GPIO_Init(GPIOB, &GPIO_InitStructure);
33
34    GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_TIM1);
35    GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);
36    GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_TIM1);
37    GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_TIM1);
38    GPIO_PinAFConfig(GPIOB, GPIO_PinSource0, GPIO_AF_TIM1);
39    GPIO_PinAFConfig(GPIOB, GPIO_PinSource1, GPIO_AF_TIM1);
40
41    /* Time base configuration */
42    TIM_TimeBaseStructure.TIM_Prescaler = 0;
```

```
43    TIM_TimeBaseStructure.TIM_Period = 0xFFFF;   // 65535
44    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
45    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
46
47    TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
48
49    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
50    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
51    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
52    TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
53    TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
54    TIM_OCInitStructure.TIM_Pulse = 0;
55    //TIM_OCStructInit(&TIM_OCInitStructure);
56
57    TIM_OC1Init(TIM1, &TIM_OCInitStructure);
58    TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
59
60    TIM_OC2Init(TIM1, &TIM_OCInitStructure);
61    TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);
62
63    TIM_OC3Init(TIM1, &TIM_OCInitStructure);
64    TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Enable);
65
66    TIM_ARRPreloadConfig(TIM1, ENABLE);
67
68    /* TIM1 enable counter */
69    TIM_Cmd(TIM1, ENABLE);
70  TIM_CtrlPWMOutputs(TIM1, ENABLE);
71  }
72
73  #ifdef  USE_FULL_ASSERT
74
75  /**
76    * @brief  Reports the name of the source file and the source line
number
77    *         where the assert_param error has occurred.
78    * @param  file: pointer to the source file name
79    * @param  line: assert_param error line source number
80    * @retval None
81    */
82  void assert_failed(uint8_t* file, uint32_t line)
83  {
84    /* User can add his own implementation to report the file name and line
number,
85       ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
86
87    while (1)
88    {}
89  }
90  #endif
91
92
```

Giải thích (Led sáng tắt dần)

```
/ *Cài đặt tỉ lệ phần trăm mức cao */
 TIM1->CCR1 = 90* 65535 / 100; // 90% Duty cycle
TIM8->CCR1 = 100* 65535 / 100; // 100% Duty cycle
void TIM_PWM_Configuration(void)
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA , ENABLE);
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
/* chon che do Mode AF (mode tính nang phu tro) cho chân IO,
     mode này không phai là mode In hay Out mà do ngoai vi do tu dong thiet lap In-
Out */

  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
     /* Thiet lap A7 A8 A9 A10 ngo ra gan voi LED */
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);
   /*
A7 => CH1N  TIM1
  A8  => CH1    //
A9  => CH2    //
A10 => CH3    //
```

```
B0 => CH2N //
 B1 => CH3N  //
Khai báo các cổng bạn cần sử dụng     */
  /* Time base configuration */
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
/*
      + Neu muon chon tan so max thi set Prescaler = 0
      + timer_tick_frequency = Timer_default_frequency / (prescaller_set + 1)
 => timer_tick_frequency = 84000000 / (0 + 1) = 84000000  */
  TIM_TimeBaseStructure.TIM_Period = 0xFFFF; // 65535
/*+ PWM_frequency = timer_tick_frequency / (TIM_Period + 1)
      => PWM_frequency= 84000000/(65535 + 1) = 1281Hz


            ********** Truong hop muon cai dat PWM_prequency 10kHZ
    TIM_Period = timer_tick_frequency / PWM_frequency - 1
   => TIM_Period = 84000000 / 10000 - 1 = 8399
   Chu y : Neu TIM_Period > Max timer ( trong TH nay la 65535) thi phai chon
            prescaler lon hon => lam cham timer tick frequency */


  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
      /* TIM_ClockDivision co chuc nang chong hien tuong dead-time trong cau H
+    Hien tuong dead-time : google
+ Cach khac phuc: 1_ Bat TIM_ClockDivision len ( khoang 5-10us)
```

```
                    2_ Su dung IC tich hop san. Vd IRF2184 ( silde cuoi datasheet) , L298...*/

TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;

    /* Dùng TIM1 OC: OUTPUT COMPARE */

     /* Duoi day la thiet lap OUTPUT cho cac chan CH* */

  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; /* Cho phep TIMER chay */

 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

    /* => Chon ngo ra muc cao. Neu chon che do muc thap thi hinh dang ngo ra se bi dao nguoc

    Tuy nhien chu ky , tan so khong doi */

/* Duoi day la thiet lap OUTPUT cho cac chan CH*N */

 TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;

 TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;

 TIM_OCInitStructure.TIM_Pulse = 0;

 TIM_OCStructInit(&TIM_OCInitStructure);

  TIM_OC1Init(TIM1, &TIM_OCInitStructure); /* Truyen bien cau truc xuong cho ham TIM_OC1Init=> Thiet lap PWM1 */

  TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable); /* Bat chuc nang PRELOAD cho TIM1 */

 TIM_ARRPreloadConfig(TIM1, ENABLE); /* Cho phep PreLoad tong the TIM1 *//* TIM1 enable counter */

  TIM_Cmd(TIM1, ENABLE);

  TIM_CtrlPWMOutputs(TIM1, ENABLE); /* TIM1 va TIM8 thi can dong nay. Cac TIM khac thi khong can */
```

```
}
void Delay(__IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}
```

Code LED

```
#include "stm32f4xx.h"
TIM_TimeBaseInitTypeDef    TIM_TimeBaseStructure;
TIM_OCInitTypeDef          TIM_OCInitStructure;
GPIO_InitTypeDef           GPIO_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;


void GPIO_Configuration(void);
void TIM_PWM_Configuration(void);
void TIM8_PWM_Configuration(void);
void TIMbase_Configuration(void);
void Delay(__IO uint32_t nCount);


int main(void)
{
      int32_t a,b,c,d,i=0;
```

```
      GPIO_Configuration();
   TIM_PWM_Configuration();
      TIM8_PWM_Configuration();
a=50;
b=0;
c=90;
TIM1->CCR1 = 90* 65535 / 100; // 90% Duty cycle
TIM8->CCR1 = 100* 65535 / 100; // 100% Duty cycle
while(1)
{
GPIO_SetBits(GPIOB ,GPIO_Pin_0 );
GPIO_ResetBits(GPIOB , GPIO_Pin_1 );
}}
void GPIO_Configuration(void)
{ RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB , ENABLE);
/* Configure PB0 PB1 in output pushpull mode */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOB, &GPIO_InitStructure);     }
void TIM_PWM_Configuration(void)
```

```
{ RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);

 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA , ENABLE);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10;

 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;

/* chon che do Mode AF (mode tính nang phu tro) cho chân IO,

mode này không phai là mode In hay Out mà do ngoai vi do tu dong thiet lap In-Out */

 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;

 GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;

 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;

 GPIO_Init(GPIOA, &GPIO_InitStructure);

/* Thiet lap A7 A8 A9 A10 ngo ra gan voi LED */

 GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);

/* A7 => CH1N TIM1

A8  => CH1    //

 A9   =>   CH2   //

 A10  =>  CH3  //

 B0 => CH2N  //

 B1  => CH3N   //     */

  /* Time base configuration */

  TIM_TimeBaseStructure.TIM_Prescaler = 0;

/*

+ Neu muon chon tan so max thi set Prescaler = 0
```

```
+ timer_tick_frequency = Timer_default_frequency / (prescaller_set + 1)
 => timer_tick_frequency = 84000000 / (0 + 1) = 84000000 */
  TIM_TimeBaseStructure.TIM_Period = 0xFFFF; // 65535
/*+ PWM_frequency = timer_tick_frequency / (TIM_Period + 1)
=> PWM_frequency= 84000000/(65535 + 1) = 1281Hz
    ********** Truong hop muon cai dat PWM_prequency 10kHZ
  TIM_Period = timer_tick_frequency / PWM_frequency - 1
    => TIM_Period = 84000000 / 10000 - 1 = 8399
    Chu y : Neu TIM_Period > Max timer ( trong TH nay la 65535) thi phai chon
                prescaler lon hon => lam cham timer tick frequency */

  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
      /* TIM_ClockDivision co chuc nang chong hien tuong dead-time trong cau H
+     Hien tuong dead-time : google
+ Cach khac phuc: 1_ Bat TIM_ClockDivision len ( khoang 5-10us)
                2_ Su dung IC tich hop san. Vd IRF2184 ( silde cuoi datasheet) , L298...*/
      TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
      /* Dùng TIM1 OC: OUTPUT COMPARE */
       /* Duoi day la thiet lap OUTPUT cho cac chan CH* */
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; /* Cho phep
TIMER chay */
```

```
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

      /* => Chon ngo ra muc cao. Neu chon che do muc thap thi hinh dang ngo ra se bi
dao nguoc . Tuy nhien chu ky , tan so khong doi */

/* Duoi day la thiet lap OUTPUT cho cac chan CH*N */

 TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;

 TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;

 TIM_OCInitStructure.TIM_Pulse = 0;

//TIM_OCStructInit(&TIM_OCInitStructure);

   TIM_OC1Init(TIM1, &TIM_OCInitStructure); /* Truyen bien cau truc xuong cho
ham TIM_OC1Init=> Thiet lap PWM1 */

TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable); /* Bat chuc nang
PRELOAD cho TIM1 */

  TIM_ARRPreloadConfig(TIM1, ENABLE); /* Cho phep PreLoad tong the TIM1 */

  /* TIM1 enable counter */

  TIM_Cmd(TIM1, ENABLE);

  TIM_CtrlPWMOutputs(TIM1, ENABLE); /* TIM1 va TIM8 thi can dong nay. Cac
TIM khac thi khong can */

}

void TIM8_PWM_Configuration(void)
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC         , ENABLE);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
```

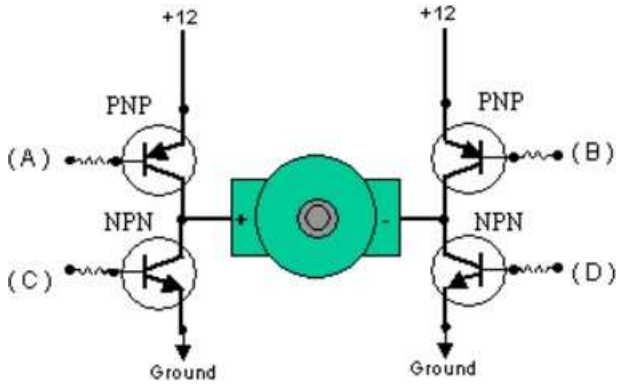
```
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM8);

  TIM_TimeBaseStructure.TIM_Prescaler = 3;
  TIM_TimeBaseStructure.TIM_Period = 41999;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM8, &TIM_TimeBaseStructure);
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;

TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;

TIM_OCInitStructure.TIM_Pulse = 0;
 //TIM_OCStructInit(&TIM_OCInitStructure);
  TIM_OC1Init(TIM8, &TIM_OCInitStructure;
  TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable

  TIM_ARRPreloadConfig(TIM8, ENABLE); /* Cho phep PreLoad tong the TIM1 */
  /* TIM1 enable counter */
  TIM_Cmd(TIM8, ENABLE);
  TIM_CtrlPWMOutputs(TIM8, ENABLE); /* TIM1 va TIM8 thi can dong nay. Cac
TIM khac thi khong can */
}
void Delay(_IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}
```

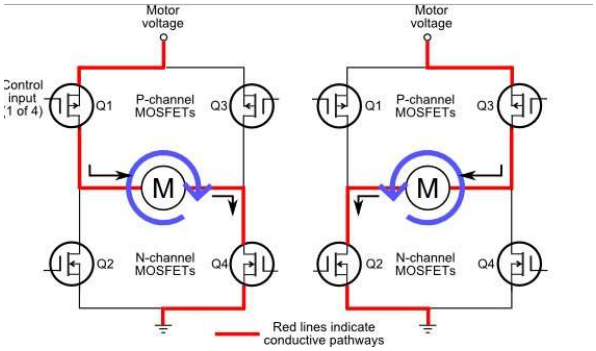
5.2.2 Ứng dụng trong điều khiển 1 động cơ DC IR2184 (chịu dòng cao)

5.2.2.1Giới thiệu modul L298

(Có thể sử dụng modul L298 điều khiển 2 động cơ DC. Tuy nhiên dòng chịu của L298 chịu 2A nên thích hợp với motor loại nhỏ)

Tìm hiểu : Mạch cầu H Nguyên lí hoạt động





Hiện tượng deadtime : Khi PWM hoạt động Q1 – Q2 có khả năng cùng dẫn 1 lúc (Q1 lên – Q2 xuống tuy nhiên khi tần số cao thì trong khi Q2 chuẩn bị clear (đang ở mức cao) thì Q1 đã set => Gây hiện tượng chập mạch VCC- GND)

IR2184 Deadtime:

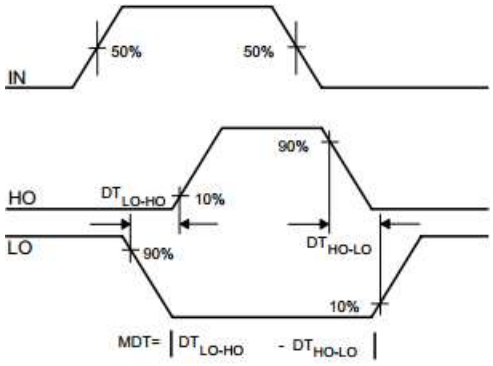
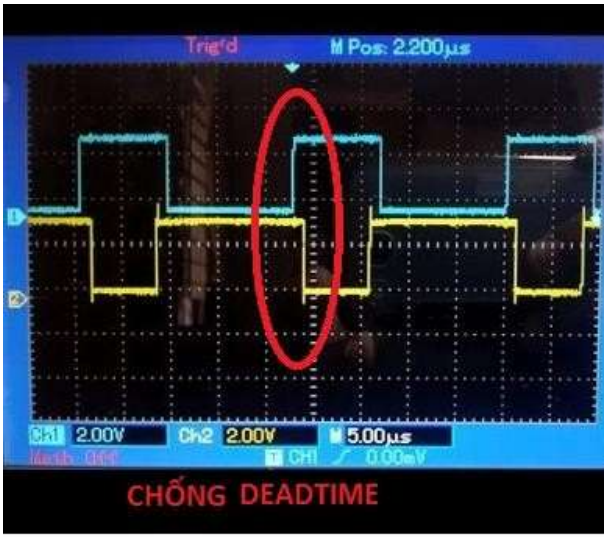
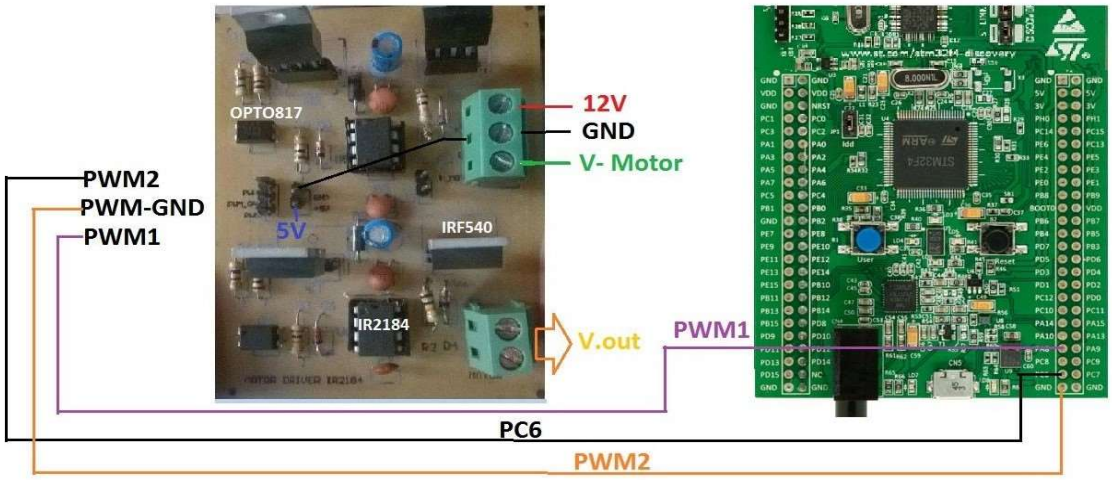


Figure 4. Deadtime Waveform Definitions





5.2.2.2Sơ đồ đấu dây

Sơ đồ nguyên lí (file Altium đính kèm)

Cách vận hành : Tầm điều khiển 0- 80% độ rộng xung

Cách kết nối và vận hành:
1) Cấp nguồn 12V - GND - V motor vào DOMINO 3
2) Cấp nguồn 5v - GND
3) Cắm jumper J2 nếu muốn xài nguồn cấp cho IR2184 chung với nguồn cho động cơ (Lưu ý 5v) .
Khi đó, để hở V_MOTOR.
4) GND_MCU được nối với GND của vi điều khiển STM32 , cách ly với GND của MOTOR và IR2184
qua OPTO 817
OPTO : PWM =0 => LED off => BJT OFF => IN = 5V
PWM =1 => LED on => BJT ON => Nối mass => IN=0
IR2184:
HO cùng tín hiệu với IN
LO ngược tín hiệu với IN
5) Cách điều khiển MOTOR :
+) Quay thuận (Cùng chiều kim đồng hồ) :
PWM2 = 100, PWM1 điều khiển tốc độ.
PWM càng nhỏ => Tốc độ càng tăng
+) Quay nghịch (Ngược chiều kim đồng hồ) :
PWM1 = 100, PWM2 điều khiển tốc độ.

Code

```
#include "stm32f4xx.h"
TIM_TimeBaseInitTypeDef    TIM_TimeBaseStructure;
TIM_OCInitTypeDef          TIM_OCInitStructure;
GPIO_InitTypeDef           GPIO_InitStructure;
void GPIO_Configuration(void);
void TIM2_Configuration(void);
void TIM1_PWM_Configuration(void);
void TIM8_PWM_Configuration(void);

void Delay(__IO uint32_t nCount);
volatile int32_t debug;

int main(void)

{
          int32_t nowtime,lasttime;
  GPIO_Configuration();
 TIM2_Configuration();
  TIM1_PWM_Configuration();
          TIM8_PWM_Configuration();
while (1)
  { debug=0;
lasttime = TIM_GetCounter(TIM2);
while( debug < 50000)
/* Vì TIMER2 cai dat 10KHz => t=(1/f).n= (1/10000).50000= 5 giay */
{
          TIM1->CCR1 = 50* 65535 / 100; // 50% Duty cycle
```

```
        TIM8->CCR1 = 100* 65535 / 100; // 100% Duty cycle
        nowtime=                TIM_GetCounter(TIM2);
        debug= nowtime- lasttime;
        }
while ( debug >= 50000)
        {
TIM1->CCR1 = 100* 65535 / 100; // 100 Duty cycle
TIM8->CCR1 = 50* 65535 / 100; // 50% Duty cycle
        }}}
void GPIO_Configuration(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
  /* Configure PB0 PB1 in output pushpull mode */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13| GPIO_Pin_14|
GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOD, &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);
}
void TIM1_PWM_Configuration(void)
{
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA , ENABLE);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 ;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_TIM1);
  /* Time base configuration */
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_Period = 0xFFFF; // 65535
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
TIM_OCInitStructure.TIM_Pulse = 0;
//TIM_OCStructInit(&TIM_OCInitStructure);

TIM_OC1Init(TIM1, &TIM_OCInitStructure);
```

```
  TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);


  TIM_ARRPreloadConfig(TIM1, ENABLE);
 /* TIM1 enable counter */
  TIM_Cmd(TIM1, ENABLE);
  TIM_CtrlPWMOutputs(TIM1, ENABLE);
}
void TIM8_PWM_Configuration(void)
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC , ENABLE);


  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 ;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);


  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM8);
   /* Time base configuration */
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseStructure.TIM_Period = 0xFFFF; // 65535
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;


  TIM_TimeBaseInit(TIM8, &TIM_TimeBaseStructure);
```

```
 TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
 TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
 TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
 TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
 TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
 TIM_OCInitStructure.TIM_Pulse = 0;
 //TIM_OCStructInit(&TIM_OCInitStructure);

 TIM_OC1Init(TIM8, &TIM_OCInitStructure);
 TIM_OC1PreloadConfig(TIM8, TIM_OCPreload_Enable);

TIM_ARRPreloadConfig(TIM8, ENABLE);
 /* TIM1 enable counter */
 TIM_Cmd(TIM8, ENABLE);
 TIM_CtrlPWMOutputs(TIM8, ENABLE);
}
void TIM2_Configuration(void)
{
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
 /* Time base configuration */
 TIM_TimeBaseStructure.TIM_Prescaler = 8399;
 TIM_TimeBaseStructure.TIM_Period = 0xFFFFFFFF;
/*
        TIMER2 32bit=> FFFFFFFF => chon tan so max cua timer 2 la 84MHZ
TIM_Period khong anh huong den tan so timer. Anh huong den tan so PWM
        BUT chu y: TIM_Period = 0 => Timer khong nhay duoc        */
 TIM_TimeBaseStructure.TIM_ClockDivision = 0;
```

```
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_Cmd(TIM2, ENABLE);
}
void Delay(__IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}

#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 can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  while (1)
  {}
```

```
}
#endif


/* Che do DC STOP : Set 2 chan dung => DC STOP */
```

6. ADC+ DMA

6.1 Giới thiệu khái niệm

-ADC (analog-to-digital converter) bộ chuyển đổi tín hiệu tương tự-số là thuật ngữ nói đến sự chuyển đổi một tín hiệu tương tự thành tín hiệu số hóa để dùng trong các hệ số(digital) hay vi điều khiển.

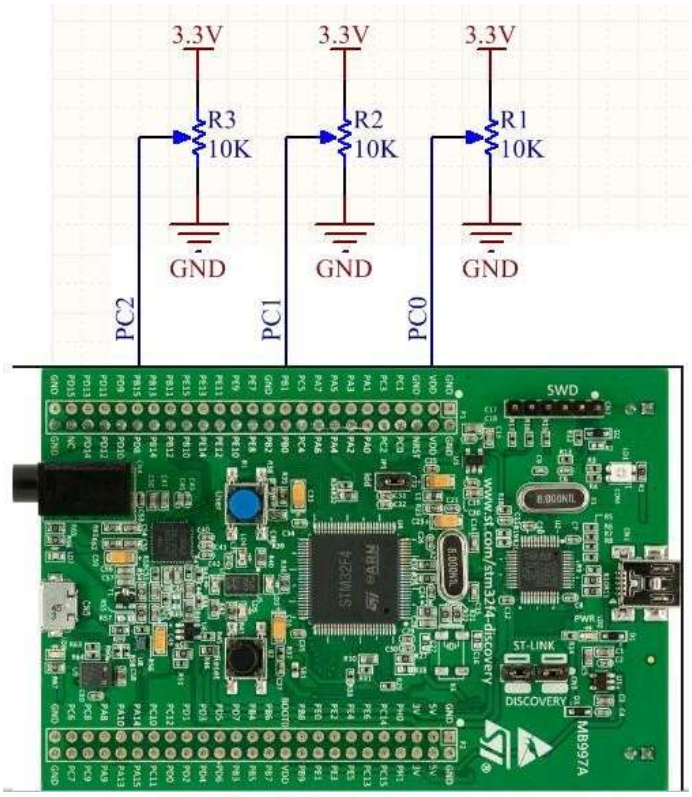
-DMA (Direct memory access) là một kỹ thuật chuyển dữ liệu từ bộ nhớ đến ngoại vi hoặc từ ngoại vi đến bộ nhớ mà không yêu cầu đến sự thực thi của CPU, có nghĩa là CPU sẽ hoàn toàn độc lập với ngoại vi được hỗ trợ DMA mà vẫn đảm bảo ngoại vi thực hiện công việc được giao, tùy vào từng loại ngoại vi mà DMA có sự hỗ trợ khác nhau.

6.2 Code ứng dụng đọc điện áp ngõ vào thành tín hiệu số

6.2.1 Ứng dụng đọc tín hiệu từ cảm biến khí gas MQ Sensor

Kết nối mạch

Kết nối chân
- PC0-AOUT
- 5V-VCC
- GND-GND



6.2.2 Code

Khởi tạo ADC-DMA

```
void ADC_Config(void)

{

  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

      // Khai báo     I/O
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN; /*Chon mode analog*/
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
GPIO_Init(GPIOC, &GPIO_InitStructure);
//Khai báo DMA
DMA_InitStructure.DMA_Channel = DMA_Channel_0; /*chanel duoc ho tro la chanel 0-do bang*/

  DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&ADCValue; /*ghep bien DMA_Memory0BaseAddr chua dia chi va cung kieu voi bien ADCValue*/

  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)(&(ADC1->DR)); /*gan dia chi thanh ghi chua gia tri chuyen doi ADC vao bien DMA_PeripheralBaseAddr cua DMA*/

  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;/*chon huong chuyen du lieu*/

  DMA_InitStructure.DMA_BufferSize = 3;/*chon kich thuoc mang du lieu*/

  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; /*moi lan chuyen du lieu, dia chi ngoai vi se ko tang dan*/

  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable; /*moi khi chuyen du lieu can tang dia chi bo nho*/
```

```
DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord; /* kich thuoc thanh ghi chua du lieu ngoai vi la
16bit*/

  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
/*kich thuoc mang du lieu ADCValue là 16bit*/

  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;/*chon mode DMA vong
tron, viec chuyen doi lien tuc lap lao*/

  DMA_InitStructure.DMA_Priority = DMA_Priority_High;/*thiet lap che do uu tien
cao*/

  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Enable;

  DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;

  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;

  DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;

  DMA_Init(DMA2_Stream0, &DMA_InitStructure);

/* DMA2_Stream0 enable */

  DMA_Cmd(DMA2_Stream0, ENABLE);
```

Khai báo ADC

```
/* ADC Common Init
**********************************************************/
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent; /*chon mode
cho ADC*/
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2; /*thiet lap bo
chia 2, cho ADC lay mau o tan so cao nhat*/
ADC_CommonInitStructure.ADC_DMAAccessMode =
ADC_DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_10Cycles; /*thoi gia tre giua 2 lan lay mau*/
ADC_CommonInit(&ADC_CommonInitStructure);

  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b; /*che do phan giai
ADC la 12 bit*/
```

```
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 3; /*so kenh ADC chuyen doi*/
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC1 regular channels configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1,
ADC_SampleTime_3Cycles);
ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 2,
ADC_SampleTime_3Cycles);
ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 3,
ADC_SampleTime_3Cycles);

  /* Enable ADC1 DMA */
  ADC_DMACmd(ADC1, ENABLE);

  /* Enable DMA request after last transfer (Single-ADC mode) */
  ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Start ADC1 Software Conversion */
  ADC_SoftwareStartConv(ADC1);
```

Trình bày sourcecode:

```
#include "stm32f4xx.h"
   DMA_InitTypeDef          DMA_InitStructure;
ADC_InitTypeDef          ADC_InitStructure;
ADC_CommonInitTypeDef     ADC_CommonInitStructure;
GPIO_InitTypeDef         GPIO_InitStructure;
   volatile uint16_t ADCValue[3]={0};
void ADC_Config(void);

int main(void)
{
```

```
  ADC_Config();
  while (1)
  {
  }
}
void ADC_Config(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN; /*Chon mode analog*/
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  DMA_InitStructure.DMA_Channel = DMA_Channel_0; /*chanel duoc ho tro la
chanel 0-do bang*/
  DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&ADCValue; /*ghep
bien DMA_Memory0BaseAddr chua dia chi va cung kieu voi bien ADCValue*/
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)(&(ADC1->DR)); /*gan
dia chi thanh ghi chua gia tri chuyen doi ADC vao bien DMA_PeripheralBaseAddr
cua DMA*/
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;   /*chon huong
chuyen du lieu*/
  DMA_InitStructure.DMA_BufferSize = 3; /*chon kich thuoc mang du lieu*/
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable; /*moi lan
chuyen du lieu, dia chi ngoai vi se ko tang dan*/
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable; /*moi khi
chuyen du lieu can tang dia chi bo nho*/
DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord; /* kich thuoc thanh ghi chua du lieu ngoai vi la
16bit*/
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
/*kich thuoc mang du lieu ADCValue là 16bit*/
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; /*chon mode DMA vong
tron, viec chuyen doi lien tuc lap lao*/
  DMA_InitStructure.DMA_Priority = DMA_Priority_High; /*thiet lap che do uu tien
cao*/
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Enable;
```

```
  DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
  DMA_Init(DMA2_Stream0, &DMA_InitStructure);

  /* DMA2_Stream0 enable */
  DMA_Cmd(DMA2_Stream0, ENABLE);
   /* ADC Common Init
**********************************************************/
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent; /*chon mode
cho ADC*/
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2; /*thiet lap bo
chia 2, cho ADC lay mau o tan so cao nhat*/
ADC_CommonInitStructure.ADC_DMAAccessMode =
ADC_DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_10Cycles; /*thoi gia tre giua 2 lan lay mau*/
ADC_CommonInit(&ADC_CommonInitStructure);

  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b; /*che do phan giai
ADC la 12 bit*/
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 3; /*so kenh ADC chuyen doi*/
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC1 regular channels configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1,
ADC_SampleTime_3Cycles);
ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 2,
ADC_SampleTime_3Cycles);
ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 3,
ADC_SampleTime_3Cycles);

  /* Enable ADC1 DMA */
  ADC_DMACmd(ADC1, ENABLE);
```

```
  /* Enable DMA request after last transfer (Single-ADC mode) */
  ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Start ADC1 Software Conversion */
  ADC_SoftwareStartConv(ADC1);
}

void Delay(__IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}

#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  while (1)
  {}
}
#endif
```

Phần mềm hỗ trợ

STMStudio dùng để đọc giá trị thu được từ chân PC0, PC1, PC2.

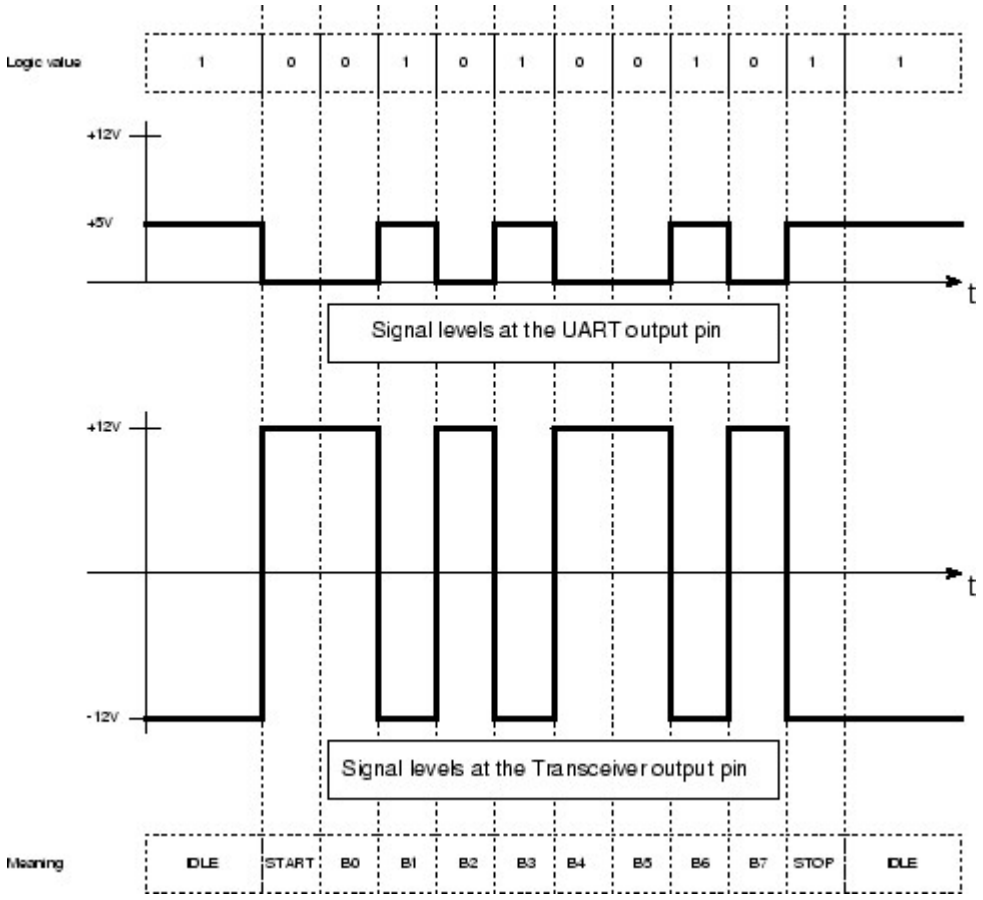
7. UART

7.1 Khái niệm cơ bản

USART (Universal synchronous asynchronous receiver transmitter) truyền nhận nối tiếp đồng bộ và không đồng bộ. Trong truyền thông nối tiếp dữ liệu truyền sẽ nối đuôi nhau trên một hay vài đường truyền. Ưu điểm của truyền thông nối tiếp là vi điều khiển có khả năng truyền-nhận nhiều dữ liệu, tiết kiệm đường đường IO, nhưng nhược điểm là không được nhanh như truyền song song và dễ bị mất, lỗi dữ liệu, phải có kế hoạch phòng ngừa các tình huống này.

Chuẩn truyền thông này tương thích với RS232 của PC nếu có một thiết bị chuyển đổi điện áp để giao tiếp. Nếu máy tính bạn không có cổng COM bạn có thể sử dụng thiết bị chuyển đổi "USB to COM" hoặc "USB to USART".

Dưới đây là khung truyền chuẩn của USART và RS232.



7.2 Sơ đồ phần cứng

Chú ý kết nối theo kiểu đấu chéo TX này vào RX kia.

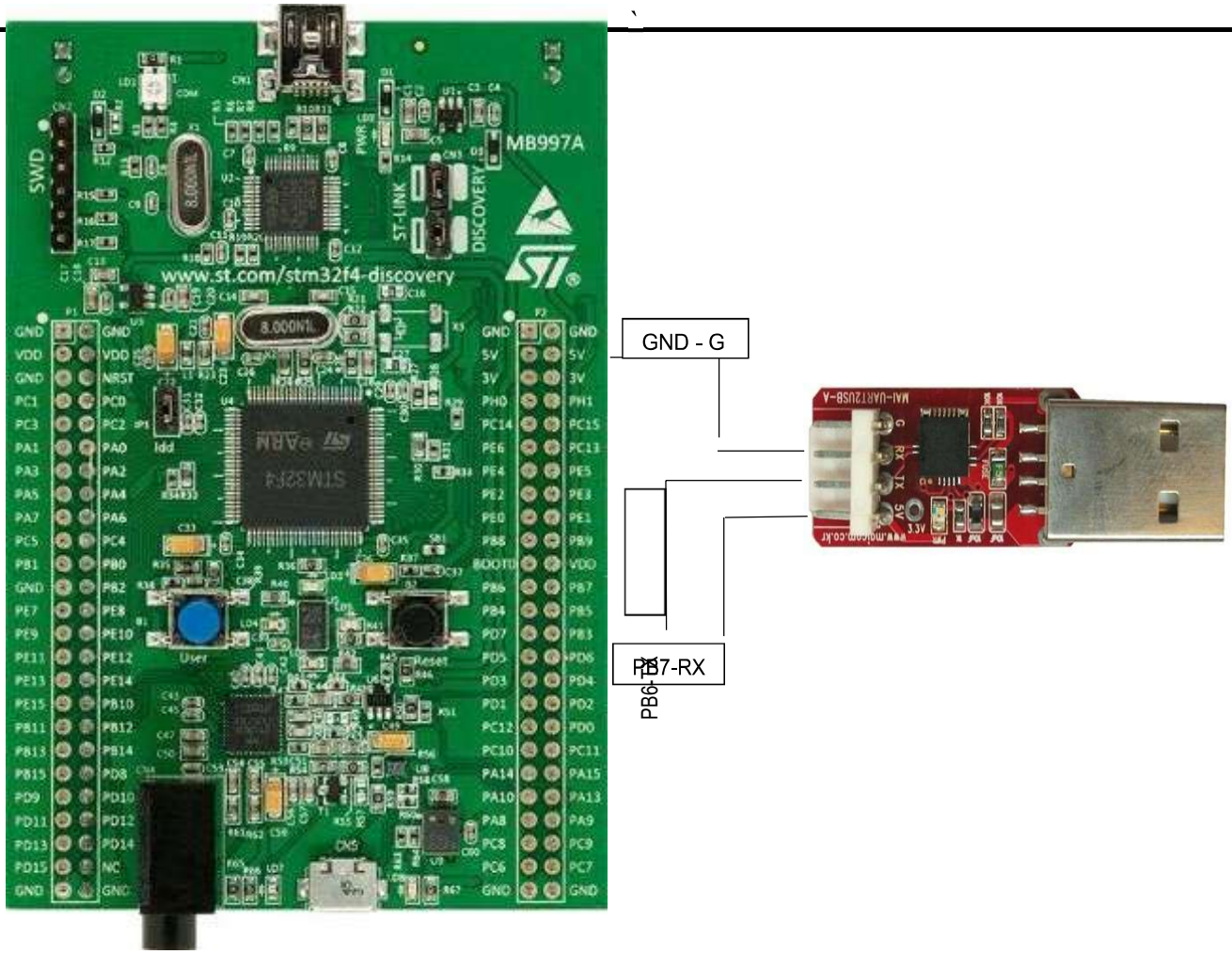
	Pins pack 1		Pins pack 2		Pins pack 3		
U(S)ARTx	TX	RX	TX	RX	TX	RX	APB
USART1	PA9	PA10	PB6	PB7			2
USART2	PA2	PA3	PD5	PD6			1
USART3	PB10	PB11	PC10	PC11	PD8	PD9	1
UART4	PA0	PA1	PC10	PC11			1
UART5	PC12	PD2					1
USART6	PC6	PC7	PG14	PG9			2
UART7	PE8	PE7	PF7	PF6			1
UART8	PE1	PE0					1

7.3 Lập trình Uart cơ bản

7.3.1 Truyền nhận chữ, số

7.3.1.1 Sơ đồ mạch

Ví dụ sử dung khối Uart 1



- PB6 nối với Rx
- PB7 nối với Tx
- GND nối với GND

7.3.1.2Khởi tạo các khối

Khởi tạo Uart

```
void USART_Configuration(unsigned int BaudRate)
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  /* Khởi tạo chân TX Uart*/
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  /* Khởi tạo chân RX Uart */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  USART_InitStructure.USART_BaudRate = BaudRate;/* BaudRate là đối số tôi truyền
từ khai báo ở đầu hàm USART_Configuration(), nó có nghĩa là tốc độ baud (số bit truyền
đi trong 1s). Chẳng hạn tôi cho đối số BaudRate là 9600 vậy trong 1s truyền đi 9600 bit
và chu kỳ mỗi bit là có thời gian là 1/9600 = 1.04^-4s. Đối số này càng cao thời gian
truyền dữ liệu càng nhanh nhưng độ an toàn lại càng giảm, ngoài ra bạn nên sử dụng tốc
độ baud với một quy chuẩn quốc tế (chẳng hạn 4800, 9600, 19200, 38400 ...)*/
USART_InitStructure.USART_WordLength = USART_WordLength_8b; //độ dài
khung truyền là 8bit
  USART_InitStructure.USART_StopBits = USART_StopBits_1; //1 stop bit
USART_InitStructure.USART_Parity = USART_Parity_No; //không kiểm tra chẵn lẽ
USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None; //chọn chế độ truyền và nhận
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; //kết
nối chân IO đến khối USART1
  USART_Init(USART1, &USART_InitStructure); //cho phép USART1 bắt đầu hoạt
động

  GPIO_PinAFConfig(GPIOB,GPIO_PinSource6,GPIO_AF_USART1);
  GPIO_PinAFConfig(GPIOB,GPIO_PinSource7,GPIO_AF_USART1);

  USART_Cmd(USART1, ENABLE);
}
```

Hàm gửi dữ liệu

```
void SendUSART(USART_TypeDef* USARTx,uint16_t ch)

{
```

```
  USART_SendData(USARTx, (uint8_t) ch);
  /* Vòng lặp sẽ kết thúc tới khi nào dừng truyền */
  while (USART_GetFlagStatus(USARTx, USART_IT_TXE) == RESET)
  {}
}
```

Hàm nhận dữ liệu

```
int GetUSART(USART_TypeDef* USARTx)
{
  while (USART_GetFlagStatus(USARTx, USART_IT_RXNE) == RESET)
  {}
  return USART_ReceiveData(USARTx);
}
```

7.3.1.3 Trình bày sourcecode

```
Main c:
#include "stm32f4xx.h"
#include <stdio.h>
USART_InitTypeDef USART_InitStructure;
GPIO_InitTypeDef GPIO_InitStructure;

void USART_Configuration(unsigned int BaudRate); // khai báo hàm uart
void SendUSART(USART_TypeDef* USARTx,uint16_t ch); // khai báo hàm gửi dữ
liệu
int GetUSART(USART_TypeDef* USARTx);//khai báo hàm nhận dữ liệu
/* Private function prototypes -----------------------------------------------*/
#ifdef_GNUC____
  /* With GCC/RAISONANCE, small printf (option LD Linker->Libraries->Small printf
     set to 'Yes') calls io_putchar() */
  #define PUTCHAR_PROTOTYPE int io_putchar(int ch)
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
  #define GETCHAR_PROTOTYPE int fgetc(FILE *f)
#endif /*__GNUC__*/
```

```
void Delay(_IO uint32_t nCount)
{
  while(nCount--)
  {
  }
}

int main(void)
{
  USART_Configuration(38400);
        printf(" Testboard STM32F4 \r"); // truyền dòng chữ lên máy tính
        while(1){
   int a; //khai báo biến a kiểu int
   scanf("%d",&a); //Nhập 1 ký tự và gán vào biến a
   printf("%d\r",a); //Gửi kí tự lại máy tính
  }
}
void USART_Configuration(unsigned int BaudRate)
{
//Khởi tạo xung clock cho các ngoại vi
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  /* Khởi tạo chân Tx cho uart */
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  /* Khởi tạo chân Rx cho uart */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  USART_InitStructure.USART_BaudRate = BaudRate; // tốc độ Baudrate
USART_InitStructure.USART_WordLength = USART_WordLength_8b; //Độ dài
khung truyền là 8b
  USART_InitStructure.USART_StopBits = USART_StopBits_1; //1 stop bit
```

```
  USART_InitStructure.USART_Parity = USART_Parity_No; //không kiểm tra chẵn lẻ
USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None; //vô hiệu hóa dòng điều khiển phần cứng
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; //chọn
chết độ truyền nhận
  USART_Init(USART1, &USART_InitStructure); //cho phép uart 1 hoạt động

  //khai báo sử dụng chân PB6,PB7
GPIO_PinAFConfig(GPIOB,GPIO_PinSource6,GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOB,GPIO_PinSource7,GPIO_AF_USART1);

  USART_Cmd(USART1, ENABLE);
}
void SendUSART(USART_TypeDef* USARTx,uint16_t ch) // hàm gửi dữ liệu
{
  USART_SendData(USARTx, (uint8_t) ch);
  /* Cho đến khi ngừng truyền dữ liệu */
  while (USART_GetFlagStatus(USARTx, USART_IT_TXE) == RESET)
  {}
}
int GetUSART(USART_TypeDef* USARTx) //hàm nhận dữ liệu
{
  while (USART_GetFlagStatus(USARTx, USART_IT_RXNE) == RESET)
  {}
  return USART_ReceiveData(USARTx);
}

/**
  * @brief Retargets the C library printf function to the USART.
  * @param  None
  * @retval None
  */
GETCHAR_PROTOTYPE
{
  return GetUSART(USART1);
}
```

```
PUTCHAR_PROTOTYPE // mỗi khi hàm printf chạy sẽ nhãy xuống câu lệnh này
{
 /* Place your implementation of fputc here */
 /* e.g. write a character to the USART */
 USART_SendData(USART1, (uint8_t) ch);
 /* Loop until the end of transmission */
 while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET)
 {}
 return ch;
}
```

7.3.1.4 Tạo giao diện C#

Link hướng dẫn : http://www.payitforward.edu.vn/wordpress/tutorials/lap-trinh-csharp-va-ung-dung-voi-vi-dieu-khien/

8. SERIAL PERIPHERAL INTERFACE SPI:

8.1 Giới thiệu SPI:

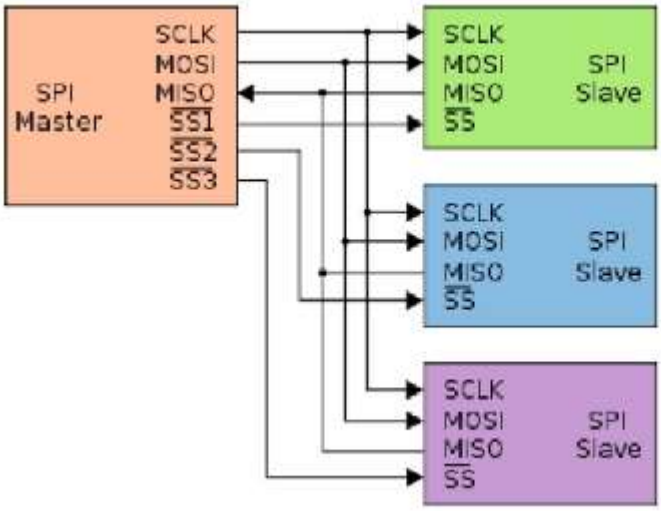
SPI (Serial Peripheral Interface, SPI bus – Giao diện Ngoại vi nối tiếp) là một chuẩn đồng bộ nối tiếp để truyền dữ liệu ở chế độ song công toàn phần full-duplex (hai chiều, hai phía), do công ty Motorola thiết kế để tạo ra một chuẩn giao tiếp giữa các vi điều khiển với nhau một cách đơn giản và hiệu quả. SPI là một giao thức truyền thông đồng bộ trong đó CLK của Master sẽ được sử dụng làm xung đồng bộ giữa Master và Slave.

Sơ đồ truyền thông SPI giữa 2 vi xử lý

Trong giao tiếp SPI 1 thiết bị sẽ được chọn làm Master và bộ còn lại là Slave.

Đôi lúc SPI còn được gọi là chuẩn giao tiếp 4 dây bởi vì chúng sử dụng 4 dây để thiết lập kết nối giữa các vi xử lý, các ngoại vi với nhau, 4 dây đó có các chức năng là:

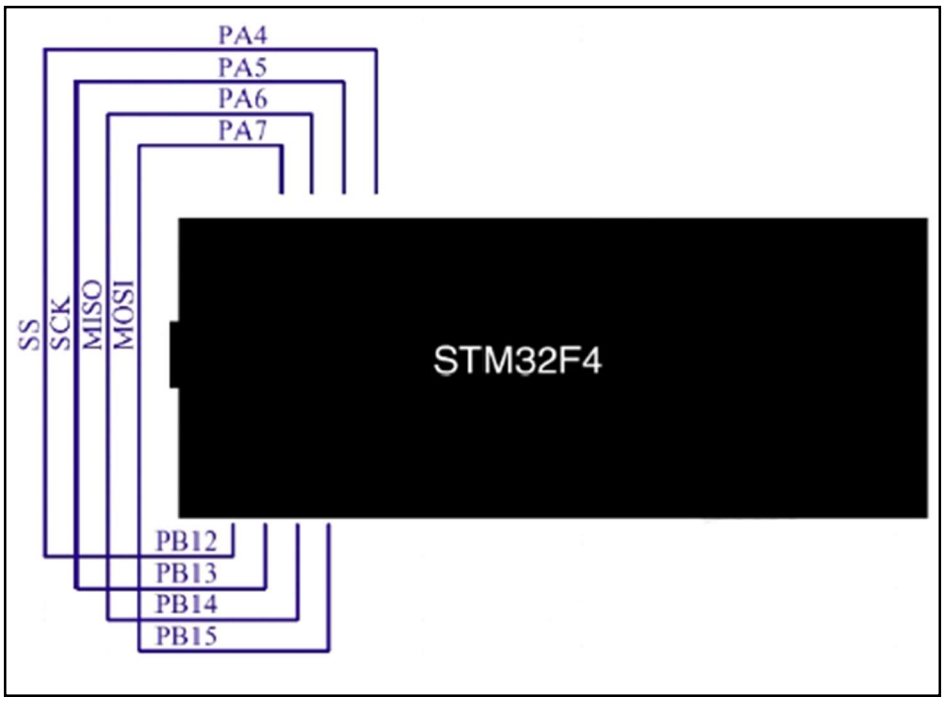
- MOSI (Master Out Slave In): Cổng này đóng vai trò ngõ ra của Master và là ngõ vào của bên Slave.
- MISO(Master In Slave out): Đóng vai trò là ngõ và của Master và ngõ ra của Slave.
- SCLK hay SCK: Đây là tín hiệu clock dùng cho việc đồng bộ truyền nhận giữa các thiết bị
- CS (Chip Select): được sử dụng để chọn Slave. Nếu như có kết nối với nhiều thiết bị khác nhau thì nó sẽ được sử dụng để chọn các chip Slave đẩy dữ liệu ra để tranh trường hợp gây lỗi dữ liệu



Cấu hình 1 Master kết nối với nhiều Slave

Trong cấu hình kết nối như hình trên các chân CS có thể được sử dụng các chân IO thông thường.

8.2 Ví dụ về SPI:



Chức năng của các chân theo bảng bên dưới:

	SLAVE	MASTER
CONTROL_SS	PB12	PA4
SCK	PB13 --- SPI2_SCK	PA5 --- SPI1_SCK
MISO	PB14 --- SPI2_MISO	PA6 --- SPI1_MISO
MOSI	PB15 --- SPI2_MOSI	PA7 --- SPI1_MOSI

Chúng ta cần phải lập trình cho cả hàm main.c và stm32f4xx_it.c. Dưới đây là cách lập trình hàm main.c

```
#include "stm32f4xx.h"

#define SS_DIS GPIO_ResetBits(GPIOA, GPIO_Pin_4);
#define SS_EN GPIO_SetBits(GPIOA,GPIO_Pin_4);
volatile uint8_t SPI_data_send,SPI_data_get;

GPIO_InitTypeDef        GPIO_InitStructure;
SPI_InitTypeDef         SPI_InitStructure;
```

```
NVIC_InitTypeDef          NVIC_InitStructure;

void Delay(__IO uint32_t nCount);
void SPI_Configuration(void);

int main(void)
{
        SPI_Configuration();
        while(1)
        {
                SS_DIS;
                SPI_I2S_SendData(SPI1, SPI_data_send);
                while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_TXE) == RESET);
                SS_EN;
                Delay(1000);
        }

}
```

Cấu hình SPI

```
void SPI_Configuration(void)
{
 /* SPI_MASTER configuration ------------------------------------------------*/
 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
 RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
 GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
 GPIO_Init(GPIOA, &GPIO_InitStructure);
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
 GPIO_Init(GPIOA, &GPIO_InitStructure);

 GPIO_PinAFConfig(GPIOA,GPIO_PinSource5, GPIO_AF_SPI1);
```

```
GPIO_PinAFConfig(GPIOA,GPIO_PinSource6,GPIO_AF_SPI1);
GPIO_PinAFConfig(GPIOA,GPIO_PinSource7,GPIO_AF_SPI1);

SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_Init(SPI1, &SPI_InitStructure);




/* SPI_SLAVE configuration ----------------------------------------------*/
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);

GPIO_PinAFConfig(GPIOB,GPIO_PinSource12,GPIO_AF_SPI2);   // only connect to
GPIO_PinAFConfig(GPIOB,GPIO_PinSource13,GPIO_AF_SPI2);   // only connect to
GPIO_PinAFConfig(GPIOB,GPIO_PinSource14,GPIO_AF_SPI2);   // only connect to
GPIO_PinAFConfig(GPIOB,GPIO_PinSource15,GPIO_AF_SPI2);   // only connect to

SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Hard;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
```

```
  SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
  SPI_InitStructure.SPI_CRCPolynomial = 7;
  SPI_Init(SPI2, &SPI_InitStructure);

  NVIC_InitStructure.NVIC_IRQChannel = SPI2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  SPI_I2S_ITConfig(SPI2, SPI_I2S_IT_RXNE, ENABLE);
  /* Enable SPI_SLAVE */
  SPI_Cmd(SPI2, ENABLE);
  /* Enable SPI_MASTER */
  SPI_Cmd(SPI1, ENABLE);
}

void Delay(__IO uint32_t nCount)
{
  while(nCount--) { }
}

#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
  while (1){ }
}
#endif
```

Sau đó chúng ta lập trình đối với hàm Stm32f4xx_it.c

```
#include "stm32f4xx_it.h"
#include "stm32f4xx.h"

extern uint8_t SPI_data_get;

void NMI_Handler(void){ }
void HardFault_Handler(void)
{
```

```
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1){ }
}
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1){ }
}
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1){ }
}


void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1){ }
}

void SVC_Handler(void){ }

void DebugMon_Handler(void){ }

void PendSV_Handler(void){ }

void SysTick_Handler(void){ }

void SPI2_IRQHandler(void)
{
  if (SPI_I2S_GetITStatus(SPI2, SPI_I2S_IT_RXNE) != RESET)
  {
        SPI_data_get = SPI_I2S_ReceiveData(SPI2);
  }
}
```

Trước tiên chúng ta tập trung vào đoạn code cấu hình SPI: **SPI_Configuration()** để thiết lập chế độ SPI bao gồm 1 Slave và 1 Master.

SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;

Đây là dòng khai báo chế độ truyền song công cho SPI. Có nhiều chế độ truyền thông cho SPI như song công, bán song công, chỉ truyền, chỉ nhận,… Trong ví dụ này cài đặt ở chế độ truyền song công vừa cho truyền và nhận dữ liệu.

SPI_InitStructure.SPI_Mode = SPI_Mode_Master;

Dòng lệnh này chọn chế độ truyền chủ động cho SPI1.

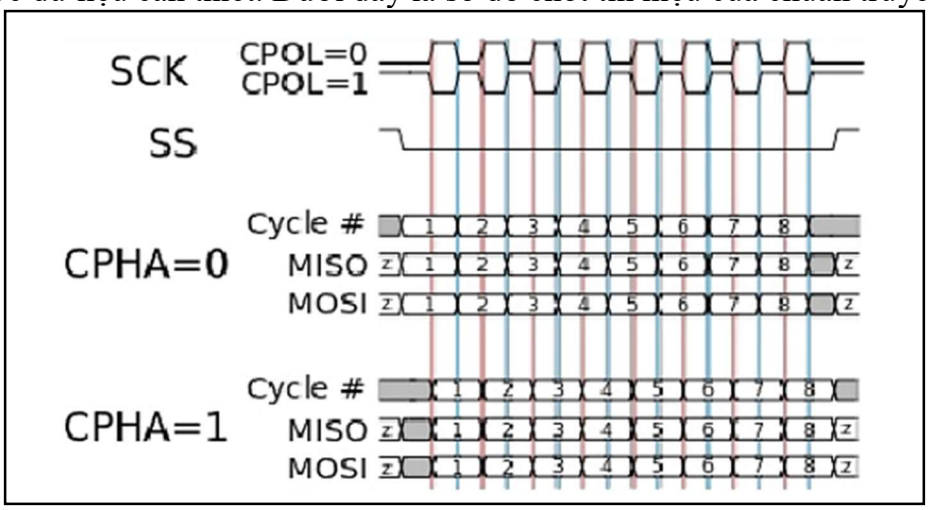
SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;

Dòng lệnh này chọn chế độ truyền chủ động cho SPI2.

SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;

Đây là dòng lệnh định khung truyền cho 1 frame dữ liệu SPI, ở đây là 8 bit.

Tiếp theo một vấn đề quan trong trong SPI là cách thức lấy mẫu tín hiệu. Bởi vì SPI là chuẩn truyền thông đồng bộ nên chúng sử dụng cùng 1 xung clock và phải cùng chốt 1 lúc ở cả master và slave để có thể lấy được dữ liệu cần thiết. Dưới đây là sơ đồ chốt tín hiệu của chuẩn truyền SPI.



Tóm lại, khi SS ở mức 0 thì dữ liệu của MISO và MOSI phải đồng bộ với nhau và sau khi thấy cạnh xuống của xung clock (SCK) thì chốt 1 bit dữ liệu, khung truyền kết thúc khi SS(CS-chip select) bật lên mức 1.

SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;

SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;

SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;

Đây là dòng lệnh cấu hình cho việc lấy xung clock của chuẩn truyền SPI và cấu hình chân SS bằng phần mềm. CPOL được cấu hình ở mức thấp có nghĩa là sẽ chốt bit truyền ở mức cao, CPHA 2 cạnh và NSS chốt bằng phần mềm chứ không phải bằng phần cứng.

SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;

Đây là dòng lệnh tạo xung nhịp trên chân SCK, chúng ta cấu hình bộ SPI thuộc APB2 nên có chu kỳ cao nhất là 84MHz, chúng ta chia 256 nên sẽ còn 328.125KHz cho clock của SPI.

SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;

Dòng này có nghĩa là chúng ta để bit MSB truyền đi trước tức là bit có trọng số cao nhất là bit thứ 8.

Các phần khai báo của Slave cũng tương tự không có nhiều thay đổi với Master.

Tiếp theo chúng ta quay trở lại hàm **main** và vòng lặp **while** để xem xét các hàm chức năng bên trong.

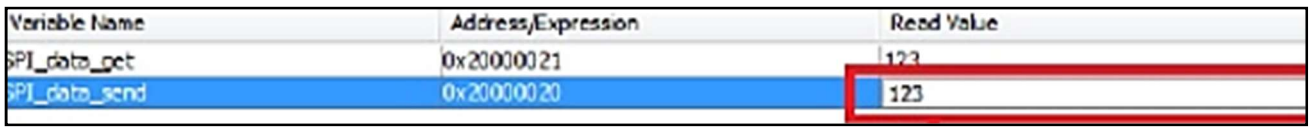
```
while(1)
        {
   SS_DIS;
                SPI_I2S_SendData(SPI1, SPI_data_send);
                while(SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_TXE)==RESET);
                SS_EN;
                Delay(1000);
        }
```

SS_DIS như ở phần khai báo là bật chân SS (dưa SS xuống mức 0).

SPI_I2S_SendData(SPI1, SPI_data_send); có nghĩa là chúng ta sẽ truyền một dữ liệu từ SPI1 có giá trị trong biến SPI_data_send.

Tiếp theo đó chúng ta đợi cờ của chuẩn truyền SPI bật về Reset.
SPI_I2S_GetFlagStatus(SPI1,SPI_I2S_FLAG_TXE)==RESET rồi ngắt chân SS (đưa SS lên 1). Sau đó Delay. Cách làm này là để tránh cho việc truyền nhận dữ liệu quá nhanh khi khung truyền cũ chưa truyền xong mà bắt đầu truyền đi khung truyền mới khiến cho frame truyền bị lỗi.

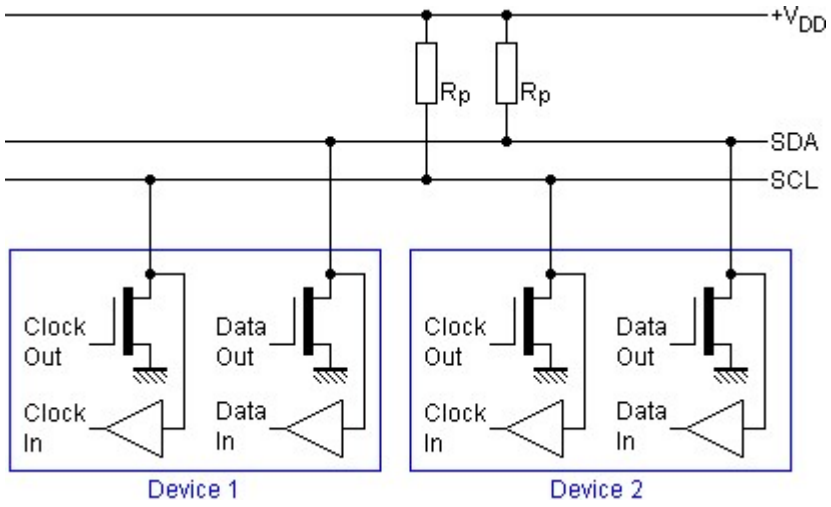
Tiếp theo chúng ta quan sát hàm **stm32f4xx_it.c** ở trong hàm này chúng ta khai báo biến SPI_data_get, khi có một tín hiệu ngắt trong hàm SPI_I2S_RecieveData() chúng ta sẽ lấy dữ liệu ra và gán vào biến SPI_data_get.
Bằng cách quan sát giá trị của biến này trên 2 chuẩn SPI 1 và SPI2 chúng ta có thể biết được giữ liệu bên trong thay ghi được truyền đi là gì . Cách quan sát các biến này có thể lấy trong cửa sổ Debuger để quan sát trực tiếp giá trị trong 2 biến này hoặc dùng phần mềm STM Studio như ví dụ bên dưới.



9. INTER INTEGRATED CIRCUIT – I2C:

9.1 Giới thiệu về I2C:

- **I²C** (*Inter-Integrated Circuit*) là một loại bus nối tiếp được phát triển bởi hãng sản xuất linh kiện điện tử Philips. Ban đầu, loại bus này chỉ được dùng trong các linh kiện điện tử của Philips. Sau đó, do tính ưu việt và đơn giản của nó, I²C đã được chuẩn hóa và được dùng rộng rãi trong các module truyền thông nối tiếp của vi mạch tích hợp ngày nay.
- Một giao tiếp I2C sử dụng 2 đường truyền tín hiệu:
  - Serial Data (SDA)
  - Serial Clock (SCL)
- SDA là đường truyền dữ liệu 2 hướng, còn SCL là đường truyền xung đồng hồ để đồng bộ và chỉ theo một hướng. Khi một thiết bị ngoại vi kết nối vào đường bus I2C thì chân SDA của nó sẽ nối với dây SDA của bus, chân SCL sẽ nối với dây SCL.

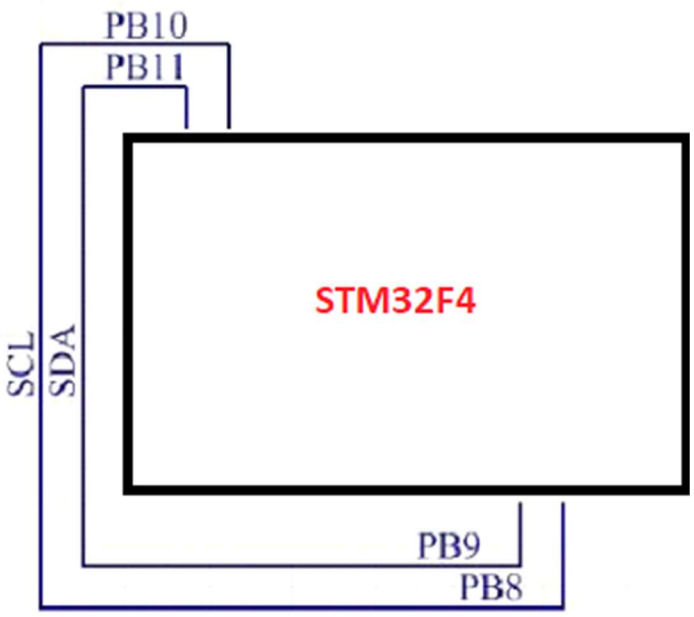


- Mỗi dây SDA hay SCL đều được nối với điện áp dương của nguồn cấp thông qua một điện trở kéo lên (pullup resistor). Sự cần thiết của các điện trở kéo này là vì chân giao tiếp I2C của các thiết bị ngoại vi thường là dạng cực máng hở (opendrain hay opencollector). Giá trị của các điện trở này khác nhau tùy vào từng thiết bị và chuẩn giao tiếp, thường dao động trong khoảng 1K đến 4.7k

- Có rất nhiều thiết bị (ICs) cùng được kết nối vào một bus I2C, tuy nhiên sẽ không xảy ra chuyện nhầm lẫn giữa các thiết bị, bởi mỗi thiết bị sẽ được nhận ra bởi một địa chỉ duy nhất với một quan hệ chủ/tớ tồn tại trong suốt thời gian kết nối. Mỗi thiết bị có thể hoạt động như là thiết bị nhận hoặc truyền dữ liệu hay có thể vừa truyền vừa nhận. Hoạt động truyền hay nhận còn tùy thuộc vào việc thiết bị đó là chủ (master) hãy tớ (slave).
- Một thiết bị hay một IC khi kết nối với bus I2C, ngoài một địa chỉ (duy nhất) để phân biệt, nó còn được cấu hình là thiết bị chủ hay tớ, vì trên một bus I2C thì quyền điều khiển thuộc về thiết bị chủ. Thiết bị chủ nắm vai trò tạo xung đồng hồ cho toàn hệ thống, khi giữa hai thiết bị chủ-tớ giao tiếp thì thiết bị chủ có nhiệm vụ tạo xung đồng hồ và quản lý địa chỉ của thiết bị tớ trong suốt quá trình giao tiếp. Thiết bị chủ giữ vai trò chủ động, còn thiết bị tớ giữ vai trò bị động trong việc giao tiếp.

9.2 Ví dụ về I2C:

Kết nối phần cứng như sau:



Trong bài viết sử dụng 2 bộ I2C, I2C1 vai trò là chip master, I2C2 là chip slave. Master truyền dữ liệu theo kiểu thông thường, slave thiết lập ngắt sự kiện.

Nội dung file main.c

```
#include "stm32f4xx.h"
#include "MY_I2C.h"

/*      MASTER                  SLAVE
  PB8 --- I2C1_SCL          PB10 --- I2C2_SCL
  PB9 --- I2C1_SDA          PB11 --- I2C2_SDA
**************************************************/

#define FAST_I2C_MODE
#define Slave_Address 0x68
#define BufferSIZE 3

volatile uint8_t MasterTxBuffer[BufferSIZE] = {1,2,3};
volatile uint8_t MasterRxBuffer[BufferSIZE];
volatile uint8_t SlaveTxBuffer[BufferSIZE] = {4,5,6};
volatile uint8_t SlaveRxBuffer[BufferSIZE];
int c;


GPIO_InitTypeDef  GPIO_InitStructure;
I2C_InitTypeDef   I2C_InitStructure;
NVIC_InitTypeDef  NVIC_InitStructure;
void I2C_Configuration(void);
void Delay(__IO uint32_t nCount);
```

```
int main(void)
{
 I2C_Configuration();
 while (1)
 {
  I2C_ByteWrite(I2C1, Slave_Address, (u8*)MasterTxBuffer,3);
  I2C_BufferRead(I2C1, Slave_Address, (u8*)MasterRxBuffer,3);
  c = SlaveRxBuffer[0];
  while(1);
 }
}

void I2C_Configuration(void)
{
 #ifdef FAST_I2C_MODE
 #define I2C_SPEED 400000
 #define I2C_DUTYCYCLE I2C_DutyCycle_16_9
 #else /* STANDARD_I2C_MODE*/
 #define I2C_SPEED 100000
 #define I2C_DUTYCYCLE I2C_DutyCycle_2
 #endif /* FAST_I2C_MODE*/

 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2, ENABLE);

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_10 | GPIO_Pin_3;
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
 GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;/*enable pullup resistors */
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
 GPIO_Init(GPIOB, &GPIO_InitStructure);

 GPIO_PinAFConfig(GPIOB,GPIO_PinSource6,GPIO_AF_I2C1);
 GPIO_PinAFConfig(GPIOB,GPIO_PinSource7,GPIO_AF_I2C1);
 GPIO_PinAFConfig(GPIOB,GPIO_PinSource3,GPIO_AF_I2C2);
 GPIO_PinAFConfig(GPIOB,GPIO_PinSource10,GPIO_AF_I2C2);

 /********************** Master **********************/
 /* I2C De-initialize */
 I2C_DeInit(I2C1);
 I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
 I2C_InitStructure.I2C_DutyCycle = I2C_DUTYCYCLE;
 I2C_InitStructure.I2C_OwnAddress1 = 0x01;
 I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
 I2C_InitStructure.I2C_ClockSpeed = I2C_SPEED;

 I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
 I2C_Init(I2C1, &I2C_InitStructure);
 /* I2C enable */
 I2C_Cmd(I2C1, ENABLE);
 /* Enable Interrupt */
```

```
  /****************** Slave ******************************/
  /* I2C De-initialize */
  I2C_DeInit(I2C2);
  I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle = I2C_DUTYCYCLE;
  I2C_InitStructure.I2C_OwnAddress1 = Slave_Address;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_ClockSpeed = I2C_SPEED;
  I2C_InitStructure.I2C_AcknowledgedAddress =  I2C_AcknowledgedAddress_7bit;
  I2C_Init(I2C2, &I2C_InitStructure);
  /* I2C ENABLE */
  I2C_Cmd(I2C2, ENABLE);

  /* Enable Interrupt */
  I2C_ITConfig(I2C2, (I2C_IT_ERR | I2C_IT_EVT | I2C_IT_BUF) , ENABLE);

  NVIC_InitStructure.NVIC_IRQChannel = I2C2_EV_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  NVIC_InitStructure.NVIC_IRQChannel = I2C2_ER_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
void Delay(__IO uint32_t nCount)
{
  while(nCount--){ }
}

#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  /* Infinite loop */
  while (1){ }
}
#endif
```

Nội dung file stm32f4xx_it.c

```
#include "stm32f4xx_it.h"

void NMI_Handler(void)
{
}
```

```
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {
  }
}

void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {
  }
}

void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {
  }
}

void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {
  }
}

void SVC_Handler(void)
{
}

void DebugMon_Handler(void)
{
}



void PendSV_Handler(void)
{
}

void SysTick_Handler(void)
{
}

void I2C2_ER_IRQHandler(void)
{
  /* Check on I2C2 AF flag and clear it */
```

```
  if (I2C_GetITStatus(I2C2, I2C_IT_AF))
  {
   I2C_ClearITPendingBit(I2C2, I2C_IT_AF);
  }
}

/**
  * @brief  This function handles I2Cx event interrupt request.
  * @param  None
  * @retval None
  */

volatile uint32_t Event = 0;
volatile uint8_t Tx_Idx;
volatile uint8_t Rx_Idx;
extern uint8_t SlaveTxBuffer[];
extern uint8_t SlaveRxBuffer[];


void I2C2_EV_IRQHandler(void)
{
    /* Get Last I2C Event */
    Event = I2C_GetLastEvent(I2C2);
    switch (Event)
    {

    /*******************************************************/
    /*         Slave Transmitter Events                    */
    /*                                        */
    /* *****************************************************/

    /* Check on EV1 */
    case I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED:
        Tx_Idx = 0;
        I2C_ITConfig(I2C2, I2C_IT_BUF , ENABLE);
        break;



   /* Check on EV3 */
     case I2C_EVENT_SLAVE_BYTE_TRANSMITTING:
     case I2C_EVENT_SLAVE_BYTE_TRANSMITTED:
        I2C_SendData(I2C2, SlaveTxBuffer[Tx_Idx++]);
        break;

    /*************************************************/
    /*             Slave Receiver Events         */
    /*                                      */
    /*************************************************/

    /* check on EV1*/
    case I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED:
        Rx_Idx = 0;
```

```
            break;

        /* Check on EV2*/
        case I2C_EVENT_SLAVE_BYTE_RECEIVED:
        case (I2C_EVENT_SLAVE_BYTE_RECEIVED | I2C_SR1_BTF):
            SlaveRxBuffer[Rx_Idx++] = I2C_ReceiveData(I2C2);
            break;

        /* Check on EV4 */
        case I2C_EVENT_SLAVE_STOP_DETECTED:
            I2C_GetFlagStatus(I2C2, I2C_FLAG_STOPF);
            I2C_Cmd(I2C2, ENABLE);
            break;

        default:
            break;
        }
    }
```

Nội dung file MY_I2C.h:

```
#ifndef __MY_I2C_H
#define __MY_I2C_H

#ifdef __cplusplus
 extern "C" {
#endif

#include "stm32f4xx.h"

void I2C_ByteWrite(I2C_TypeDef* I2Cx, u8 slaveAddr, u8* pBuffer, u16 NumByteToWrite);
void I2C_BufferRead(I2C_TypeDef* I2Cx, u8 slaveAddr, u8* pBuffer, u16 NumByteToRead);
void I2C_BufferRead_Addr(I2C_TypeDef* I2Cx, u8 slaveAddr, u8* pBuffer, u8 ReadAddr, u16
NumByteToRead);


void I2C_ByteWrite(I2C_TypeDef* I2Cx, u8 slaveAddr, u8* pBuffer, u16 NumByteToWrite)
{
        int i;
  /* Send START condition */
  I2C_GenerateSTART(I2Cx, ENABLE);
  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send slave address for write */
  I2C_Send7bitAddress(I2Cx, slaveAddr, I2C_Direction_Transmitter);

  /* Test on EV6 and clear it */
  while(!I2C_CheckEvent(I2Cx,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

  for(i=0;i<NumByteToWrite;i++)
```

```
  {
   /* Send the byte to be written */
   I2C_SendData(I2Cx, pBuffer[i]);

   /* Test on EV8 and clear it */
   while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
  }

  /* Send STOP condition */
  I2C_GenerateSTOP(I2Cx, ENABLE);
}



void I2C_BufferRead(I2C_TypeDef* I2Cx, u8 slaveAddr, u8* pBuffer, u16 NumByteToRead)
{
  /* While the bus is busy */
  while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

  /* Send START condition */
  I2C_GenerateSTART(I2Cx, ENABLE);

  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send slave address for write */
  I2C_Send7bitAddress(I2Cx, slaveAddr, I2C_Direction_Transmitter);

  /* Test on EV6 and clear it */
  while(!I2C_CheckEvent(I2Cx,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

  /* Clear EV6 by setting again the PE bit */

  I2C_Cmd(I2Cx, ENABLE);


  /* Send START condition a second time */
  I2C_GenerateSTART(I2Cx, ENABLE);

  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send slave address for read */
  I2C_Send7bitAddress(I2Cx, slaveAddr, I2C_Direction_Receiver);

  /* Test on EV6 and clear it */
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

  /* While there is data to be read */
  while(NumByteToRead)
  {
   if(NumByteToRead == 1)
```

```
    {
      /* Disable Acknowledgement */
      I2C_AcknowledgeConfig(I2Cx, DISABLE);

      /* Send STOP Condition */
      I2C_GenerateSTOP(I2Cx, ENABLE);
    }

    /* Test on EV7 and clear it */
    if(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED))
    {
      /* Read a byte from the slave */
      *pBuffer = I2C_ReceiveData(I2Cx);

      /* Point to the next location where the byte read will be saved */
      pBuffer++;

      /* Decrement the read bytes counter */
      NumByteToRead--;
    }
  }

  /* Enable Acknowledgement to be ready for another reception */
  I2C_AcknowledgeConfig(I2Cx, ENABLE);
}

void I2C_BufferRead_Addr(I2C_TypeDef* I2Cx, u8 slaveAddr, u8* pBuffer, u8 ReadAddr, u16
NumByteToRead)
{
  /* While the bus is busy */

 while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));

  /* Send START condition */
  I2C_GenerateSTART(I2Cx, ENABLE);

  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send slave address for write */
  I2C_Send7bitAddress(I2Cx, slaveAddr, I2C_Direction_Transmitter);

  /* Test on EV6 and clear it */
  while(!I2C_CheckEvent(I2Cx,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

  /* Clear EV6 by setting again the PE bit */
  I2C_Cmd(I2Cx, ENABLE);

  /* Send the slave internal address to write to */
  I2C_SendData(I2Cx, ReadAddr);

  /* Test on EV8 and clear it */
```

```
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

  /* Send START condition a second time */
  I2C_GenerateSTART(I2Cx, ENABLE);

  /* Test on EV5 and clear it */
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send slave address for read */
  I2C_Send7bitAddress(I2Cx, slaveAddr, I2C_Direction_Receiver);

  /* Test on EV6 and clear it */
  while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

  /* While there is data to be read */
  while(NumByteToRead)
  {
    if(NumByteToRead == 1)
    {
      /* Disable Acknowledgement */
      I2C_AcknowledgeConfig(I2Cx, DISABLE);

      /* Send STOP Condition */
      I2C_GenerateSTOP(I2Cx, ENABLE);
    }

    /* Test on EV7 and clear it */

  if(I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED))
    {
      /* Read a byte from the slave */
      *pBuffer = I2C_ReceiveData(I2Cx) ;

      /* Point to the next location where the byte read will be saved */
      pBuffer++;

      /* Decrement the read bytes counter */
      NumByteToRead--;
    }
  }

  /* Enable Acknowledgement to be ready for another reception */
  I2C_AcknowledgeConfig(I2Cx, ENABLE);
}


#ifdef __cplusplus
}
#endif

#endif /* __STM32F4xx_IT_H */

/****END OF FILE****/
```

Giải thích:

- Đây là các mảng chứa dữ liệu truyền đi và nhận về.

```
volatile uint8_t MasterTxBuffer[BufferSIZE] = {1,2,3};
volatile uint8_t MasterRxBuffer[BufferSIZE];
volatile uint8_t SlaveTxBuffer[BufferSIZE] = {4,5,6};
volatile uint8_t SlaveRxBuffer[BufferSIZE];
```

- Đây là các dòng tiền sử lý:
- Ở đây, đầu chương trình đã có dòng #define FAST_I2C_MODE nên tốc độ được sử dụng ở đây là 400000 bit/s.

```
#ifdef FAST_I2C_MODE
 #define I2C_SPEED 400000
 #define I2C_DUTYCYCLE I2C_DutyCycle_16_9
#else /* STANDARD_I2C_MODE*/
 #define I2C_SPEED 100000
 #define I2C_DUTYCYCLE I2C_DutyCycle_2
#endif /* FAST_I2C_MODE*/
```

- Đoạn code tiếp theo bật xung clock cho các khối IO và ngoại vi cần dùng:

```
 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
 RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2, ENABLE);
```

- Tiếp theo cấu hình các chân ngoại vi và kết nối nó đến ngoại vi I2C:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_10 |
GPIO_Pin_3;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOB, &GPIO_InitStructure);

GPIO_PinAFConfig(GPIOB,GPIO_PinSource6,GPIO_AF_I2C1);
GPIO_PinAFConfig(GPIOB,GPIO_PinSource7,GPIO_AF_I2C1);
GPIO_PinAFConfig(GPIOB,GPIO_PinSource3,GPIO_AF_I2C2);
GPIO_PinAFConfig(GPIOB,GPIO_PinSource10,GPIO_AF_I2C2);
```

- Phần code tiếp theo cấu hình cho I2C 1:

```
 /* I2C De-initialize */
 I2C_DeInit(I2C1);
 I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
 I2C_InitStructure.I2C_DutyCycle = I2C_DUTYCYCLE;
 I2C_InitStructure.I2C_OwnAddress1 = 0x01;
 I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
 I2C_InitStructure.I2C_ClockSpeed = I2C_SPEED;
 I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
 I2C_Init(I2C1, &I2C_InitStructure);
```

```
/* I2C ENABLE */
I2C_Cmd(I2C1, ENABLE);
```

Khối I2C 1 sử dụng mode I2C, duty là I2C_DutyCycle_16_9 như đã định nghĩ ở phần tiền xử lý. Địa chỉ module là 0x01 loại 7 bit, bật xác nhận ACK, tốc độ bit đã được nói trước đó là 400Kbs.
Cuối cùng cho phép I2C hoạt động bằng lệnh: I2C_Cmd(I2C1, ENABLE);

- Tiếp tục cấu hình cho I2C 2:

```
I2C_DeInit(I2C2);
I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
I2C_InitStructure.I2C_DutyCycle = I2C_DUTYCYCLE;
I2C_InitStructure.I2C_OwnAddress1 = Slave_Address;
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_InitStructure.I2C_ClockSpeed = I2C_SPEED;
I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_Init(I2C2, &I2C_InitStructure);
/* I2C ENABLE */
I2C_Cmd(I2C2, ENABLE);
/* Enable Interrupt */
I2C_ITConfig(I2C2, (I2C_IT_ERR | I2C_IT_EVT | I2C_IT_BUF) , ENABLE);
```

Khối I2C 2 sử dụng mode I2C, duty là I2C_DutyCycle_16_9. Địa chỉ module là Slave_Address loại 7 bit đã được định nghiã ở đầu đoạn code, bật xác nhận ACK, tốc độ bit đã được nói trước đó là 400Kbs.
Sau đó cho phép I2C hoạt động bằng lệnh: I2C_Cmd(I2C2, ENABLE);
Và cuối cùng cho phép các cờ ngắt của khối I2C 2 để thực hiện nhận dữ liệu: I2C_ITConfig(I2C2, (I2C_IT_ERR | I2C_IT_EVT | I2C_IT_BUF) , ENABLE);

- Sau cùng là cấu hình các ngắt và cho phép ngắt hoạt động.

```
NVIC_InitStructure.NVIC_IRQChannel = I2C2_EV_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

NVIC_InitStructure.NVIC_IRQChannel = I2C2_ER_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

- Hãy mở file stm32f4xx_it.c. Quá trình slave xử lý chủ yếu nằm trong đây.

Ở dòng **void I2C2_EV_IRQHandler(void)**: mỗi khi có một sự kiện truyền nhận dữ liệu, địa chỉ,… của slave sẽ xảy ra một ngắt và sẽ được trỏ đến hàm này.

Hoạt động truyền nhận sẽ diễn ra như sau:

Bước 1 : Khi phát hiện ra một sự kiện trên đường truyền đầu tiên nó sẽ nhảy vào ngắt I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED lợi dụng việc này gán Tx_Idx = 0 biến này là một biến đếm thứ tự dữ liệu gửi đi.

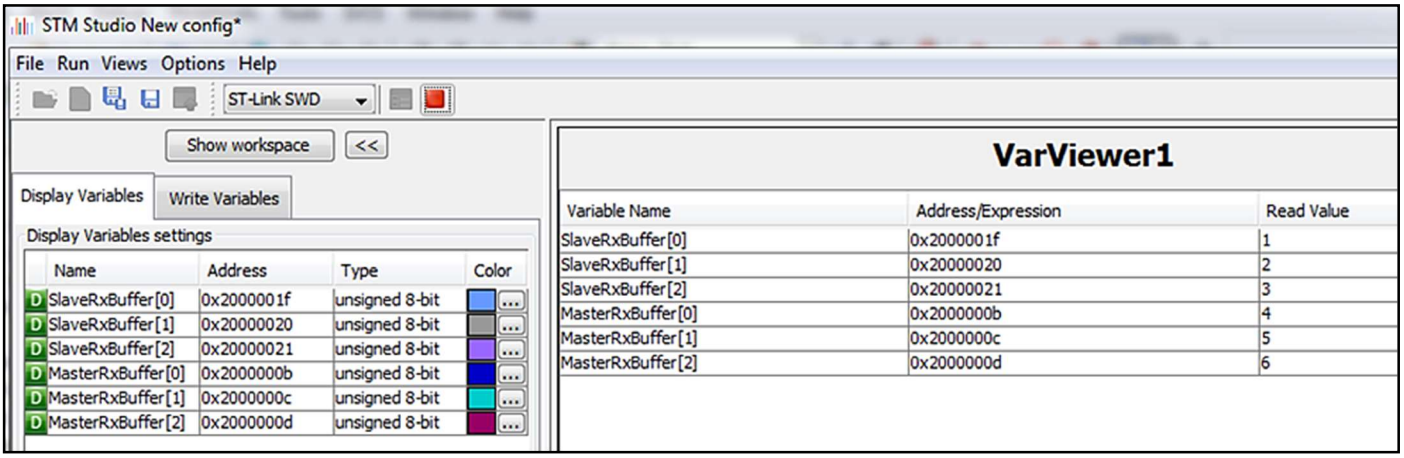
Bước 2 : Nếu master muốn slave gửi dữ liệu thì lần ngắt tiếp theo sẽ nhảy vào đây, ở ngắt này sẽ gửi dữ liệu trong mảng SlaveTxBuffer[] đến master. Ngắt này sẽ được thực hiện nhiều lần, đã thực hiện phép tăng biến Tx_Idx mỗi lần xảy ra ngắt. Quá trình xảy ra cho đến khi master không muốn nhận

dữ liệu nữa nó sẽ phát ra một tín hiệu kết thúc.

Bước 3 : Bước cuối trong quá trình truyền dữ liệu của slave, nó sẽ nhảy đến ngắt I2C_EVENT_SLAVE_STOP_DETECTED.

Quá trình nhận dữ liệu của slave cũng xảy ra tương tự như 3 bước trên nhưng các sự kiện có khác để phù hợp với quá trình nhận.

Để chắc chắn sự truyền nhận hoàn toàn thành công ta có thể sử dụng phần mềm STMStudio kiểm tra lại các biến nhận như hình bên dưới.



Tài liệu tham khảo:

1. http://icviet.vn/bai-hoc/vi-dieu-khien/stm32/
2. http://www.arm.vn/
3. Definitive Guide to ARM(r) Cortex(r)-M3 and Cortex(r)-M4 Processors, The - Yiu, Joseph
4. Discovering the STM32 microcontroller