**HƯỚNG DẪN SỬ DỤNG THƯ VIỆN STM32F103C8T6**

## Tổng quan

* Thư viện còn thiếu phần giải thích nội dung một số hàm.
* Các module cần được test lại do vẫn còn sửa chữa nội dung.
* Có vấn đề ở module analog.

## Tình trạng hoàn thành thư viện

### GPIO

+ pinMode, digitalToggle, digitalRead, digitalWrite.

### ADC

+ analogEnable, analogDisable, analogRead, ADC\_DMA.

### PWM

+ analogInit, analogWrite*.*

### Ngắt ngoài

+ attachInterrupt.

*\* lưu ý không thể dùng chung 2 ngắt trên 1 line vd A1 và B1*

*\* chân B3 không ngắt được, không tìm ra nguyên nhân, đã test trên board khác cũng ko đc.*

### Timer

+ millisInit, millis, delay. delayMicroseconds.

\* các timer 2,3,4 dùng riêng để chạy PWM

### USART(Serial)

+ begin, end, ngắt nhận USART, print, println, write, read.

+ hoàn thành cả 3 bộ USART cùng với ngắt nhận dữ liệu.

* Bỏ qua hàm Available vì không sử dụng bộ đệm để lưu giá trị,

\*

### I2C

+ begin, end , beginTransmission, endTransmission, requestFrom, write, available, read, Slave ngắt yêu cầu gởi đi(onrequest), Slave ngắt yêu cầu gởi đến

+ đã bổ sung I2C2.

### SPI

+ Begin, transfer, setDataMode, setClockDivider, read, attachInterrupt, detachInterrupt.

+ đã bổ sung SPI2

* Các hàm setBitOrder, setClockDivider không cần thiết.

## Hướng dẫn sử dụng thư viện

### GPIO

\* Thư viện GPIO chứa các hàm dùng để điều khiển I/O cho các chân vđk. Việc điều khiển bao gồm:

* PinMode: cấu hình chế độ vào ra cho các chân.
* digitalRead: Đọc trạng thái 1 chân digital.
* digitalWrite: xuất trạng thái 1 chân ( mức 1 hoặc 0).
* digitalToggle: đảo trạng thái 1 chân (từ 1 qua 0 hoặc ngược lại).

#### PinMode()

cấu hình chế độ vào ra cho các chân.

\* **Cú pháp**

**pinMode**(GPIO\_ PINx, GPIO\_MODE\_)

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: A0 – C15.

GPIO\_MODE\_: Chế độ hoạt động cần thiết lập.

Các chế độ của GPIO hiện tại:

* **INPUT**: chế độ nhận digital.
* **OUTPUT**: chế độ xuất digital.
* **INPUT\_PULLUP**: chế độ nhận digital kéo lên Vcc.
* **INPUT\_PULLDOWN**: chế độ nhận digital kéo xuống GND.
* **AF\_PP**: có thế nhập/ xuất nhưng ko có trở kéo lên/xuống nếu nhận digital, khi xuất thì theo kiểu Push/Pull.
* **AF\_OD**: có thế nhập/ xuất nhưng ko có trở kéo lên/xuống nếu nhận digital, khi xuất thì theo kiểu Open-Drain.

***\* Giải thích thêm về Push-Pull và Open Drain:***

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

\* **Trả về**

Không.

**\* Ví du**

pinMode(C13, INPUT);

pinMode(B13, AF\_OD);

#### digitalRead()

Hàm digitalRead() dùng để đọc trạng thái của chân GPIO.

\* **Cú pháp**

pinStatus = **pinMode**(GPIO\_PINx)

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: A0 – C15.

\* **Trả về**

0 hoặc LOW nếu pin ở mức thấp.

1 hoặc HIGH nếu pin ở mức cao

**\* Ví dụ**

int pinStatus = digitalRead(C13);

#### digitalToggle(GPIOx, PINx)

Hàm digitalToggle() dùng để đảo trạng thái của 1 chân GPIO.

\* **Cú pháp**

**digitalToggle**(GPIO\_PINx)

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: A0 – C15.

\* **Trả về**

Không.

**\* Ví du**

digitalToggle(C\_13);

#### digitalWrite (GPIO\_ PINx)

Hàm digitalWrite() dùng để thay đổi trạng thái của 1 chân GPIO là mức cao hay mức thấp.

\* **Cú pháp**

**digitalWrite**(GPIOx, PINx, mode)

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: A0 – C15.

mode: LOW/ HIGH hoặc 0/1

\* **Trả về**

Không.

**\* Ví du**

digitalWrite(C13, LOW);

digitalWrite(C13, 1);

### ADC

\* Thư viện ADC chứa các hàm dùng để điều khiển I/O cho các chân vi điều khiển. Việc điều khiển bao gồm các hàm sau:

analogEnable: cấu hình tất cả các chân analog.

analogDisable: kết thúc việc dùng chân analog.

analogRead: Đọc trạng thái 1 chân analog.

ADC\_DMA\_Init: cài đặt chế độ đọc analog sử dụng DMA.

\* Ban đầu nếu chân analog chưa cài đặt các chức năng khác thì khi bật 1 chân analog sẽ có thể dùng được các chân đó. Còn nếu chân nào đã cài đặt ở chế độ khác thì phải đặt lại bằng analogEnable mới có thể dùng chức năng analog được.

#### **analogEnable**()

Hàm analogEnable() dùng để cấu hình các chân A0-A7, B0-B1 là chân đọc analog.

\* **Cú pháp**

**analogEnable**();

\* **Tham số**

Không.

\* **Trả về**

Không.

**\* Ví du**

analogEnable();

#### **analogDisable**()

Hàm analogDisable() dùng để kết thúc việc sử dụng các chân analog.

\* **Cú pháp**

analogDisable();

\* **Tham số**

Không.

\* **Trả về**

Không.

**\* Ví du**

analogEnable();

#### **analogRead**()

Hàm analogRead() dùng để đọc giá trị analog trên 1 chân A0-A7.

\* **Cú pháp**

uint16\_tADCvalue **= analogRead**(GPIO\_PINx);

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: A0 – C15.

\* **Trả về**

ADCvalue: giá trị analog tại chân GPIO

**\* Ví du**

uint16\_t adc1 = analogRead();

#### ADC\_DMA\_Init()

Hàm ADC\_DMA\_Init() dùng để thiết lập chế độ đọc ADC bằng cơ chế DMA trên các chân A0-A7, B0-B1.

Việc đọc ADC qua DMA sẽ được thực hiện ngầm mà không cần phải viết chương trình xử lí hay nhận giá trị.

Không thể cài đặt để đọc một số chân theo ý muốn mà phải đọc từ kênh 0 đến kên 9(A0-A7, B0-B1).

\* **Cú pháp**

ADC\_DMA\_Init (uint16\_t ADC\_values[], uint8\_t len);

\* **Tham số**

ADC\_values[]: mảng chứ giá trị của các chân analog cần đọc.

len: số kênh cần đọc.

\* **Trả về**

Không.

**\* Ví du**

uint16\_t adcArray[8];

ADC\_DMA\_Init (adcArray, 8);

#### analogRead: Đọc trạng thái 1 chân analog.

**\* Cú pháp:** uint16\_t **analogRead**(uint8\_t GPIO\_PINx)

* GPIO\_PINx: 1 chân GPIO: A0-A7, B0-B1.
* Giá trị trả về kiểu uint16\_t: 0-1024.

### PWM

#### **analogInit**()

Hàm analogInit() dùng để cấu hình việc xuất PWM cho từng chân có khả năng xuất PWM.

Tần số xung: 18KHz.

Giá trị PWM cao nhất (độ phân giải): 40000.

\* **Cú pháp**

analogInit (GPIO\_PINx);

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: tùy theo danh sách bên dưới.

**\* Danh sách các chân có thể cấu hình là chân PWM:**

**PA0** Timer2\_Chanel 1 **PA6** Timer3\_Chanel 1 **PB6** Timer4\_Chanel 1

**PA1** Timer2\_Chanel 2 **PA7** Timer3\_Chanel 2 **PB7** Timer4\_Chanel 2

**PA2** Timer2\_Chanel 3 **PB0** Timer3\_Chanel 3 **PB8** Timer4\_Chanel 3

**PA3** Timer2\_Chanel 4 **PB1** Timer3\_Chanel 4 PB9 Timer4\_Chanel 4

\* **Trả về:**

Không.

**\* Ví du:**

analogInit (A0);

#### **analogWrite**()

Hàm analogWrite() Xuất PWM trên 1 chân.

\* **Cú pháp**

**analogWrite**(GPIO\_PINx, val);

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: tùy theo danh sách bên dưới.

\* GPIO và PIN phải phù hợp với danh sách chân có thể xuất PWM.

val: giá trị xung PWM, 0 < val < 40000. Nếu val > 40000 thì PWM sẽ xuất ra giá trị là 40000 (100% mức cao);

\* **Trả về**

Không.

**\* Ví du**

analogWrite(B0, 20000);

### Ngắt ngoài

Phần thư viện ngắt ngoài dùng để cấu hình ngắt trên các chân GPIO, sử dụng để đọc nút nhấn, đọc xung…

\* trước khi sử dụng cần thêm các dong sau vào cuối file **“stm32f1xx\_it.h”** như hình dưới:

// External Interrupt

**void** **EXTI15\_10\_IRQHandler**(**void**);

**void** **EXTI9\_5\_IRQHandler**(**void**);

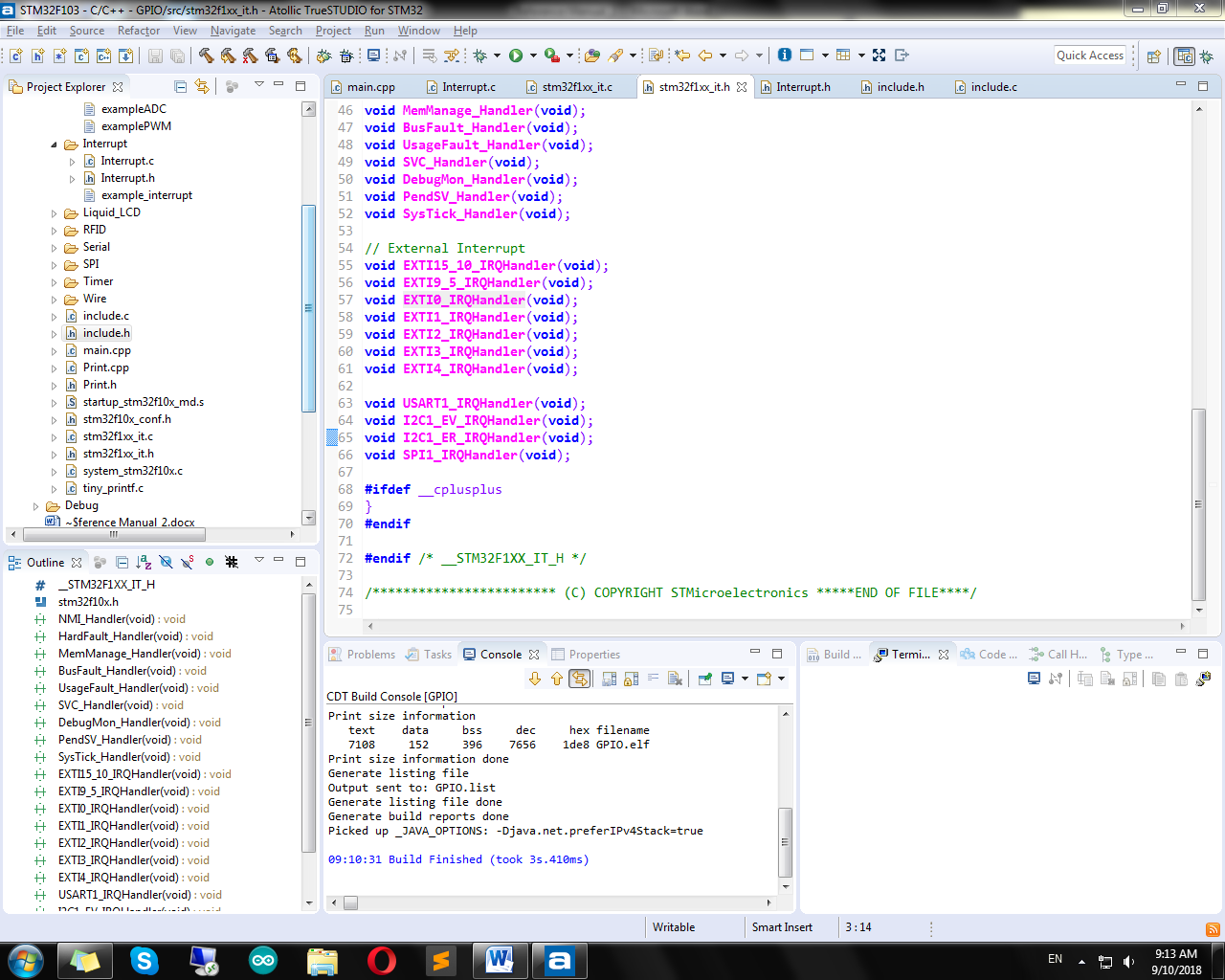
**void** **EXTI0\_IRQHandler**(**void**);

**void** **EXTI1\_IRQHandler**(**void**);

**void** **EXTI2\_IRQHandler**(**void**);

**void** **EXTI3\_IRQHandler**(**void**);

**void** **EXTI4\_IRQHandler**(**void**);



#### attachInterrupt()

Hàm attachInterrupt() dùng để thiết lập ngắt ngoài cho 1 chân GPIO.

\* **Cú pháp**

attachInterrupt(GPIO\_PINx, mode, handler);

\* **Tham số**

GPIO\_PINx: 1 chân GPIO: tùy theo danh sách bên dưới.

mode:

+ RISING: ngắt sườn lên.

+ FALLING: ngắt sườn xuống.

+ FALLING\_RISING: ngắt cả sườn lên và sườn xuống.

handler: hàm sẽ thực hiện khi ngắt trên GPIO tương ứng xảy ra.

\* lưu ý hàm này phải được khởi tạo trước khi gọi hàm attachInterrupt.

\* **Trả về**

Không.

**\* Ví du**

void INT\_1\_FUNC()

{

}

int main(void)

{

attachInterrupt(A1, RISING, INT\_1\_FUNC);

while(1)

{

}

}

### Timer

Ngắt timer được sử dụng để làm bộ định thời cho vi điều khiển bằng việc sử dụng hàm **millis().**

#### millisInit()

cấu hình systemtick timer để sử dụng hàm millis: sau mỗi 1 mili giây sẽ xảy ra ngắt 1 lần.

\* **Cú pháp**

millisInit();

\* **Tham số**

Không.

\* **Trả về**

Không.

**\* Ví du**

millisInit();

#### millis()

có nhiệm vụ trả về một số - là thời gian (tính theo mili giây) kể từ lúc mạch STM32f1 bắt đầu chương trình của bạn. Nó sẽ tràn số và quay số 0 (sau đó tiếp tục tăng).

\* **Cú pháp**

uint32\_t currentTime;

currentTimer = millis();

\* **Tham số**

Không.

\* **Trả về**

currentTimer: giá trị timer hiện tại tính từ lúc khởi động chương trình.

**\* Ví du**

currentTimer = millis();// giá trị hiện tại

delay(1000);

currentTimer = millis() – currentTimer;// giá trị timer sau 1s currentTimer = 1000

### USART(Serial)

Thư viện Serial được dùng trong việc giao tiếp giữa các board mạch với nhau (hoặc board mạch với máy tính hoặc với các thiết bị khác). Tất cả các mạch STM32f1 đều có ít nhất 1 cổng **Serial** (hay còn được gọi là UART hoặc USART). Giao tiếp Serial được thực hiện qua 2 cổng digital RX và TX hoặc qua cổng USB tới máy tính. Vì vậy, nếu bạn đang sử dụng các hàm của thư viện Serial này, bạn không thể sử dụng các chân digital RX và digital TX để làm việc khác được!

Để sử dụng thư viện, trước hết ta cần add thư viện Serial vào bằng lệnh

**#include** <Serial/Serial.h>

\* Lưu ý với các USART2 và 3 trước khi sử dụng cần khởi tạo bằng cú pháp:

\* **Cú pháp**

HardwareSerial SerialName = HardwareSerial(USART);

\* **Tham số**

USART: số thứ tự của USART: = 2 với USART2.

= 3 với USART3.

\* USART1 (SerialName là Serial) mặc định có USART = 1.

\* **Trả về**

Không.

**\* Ví dụ**

HardwareSerial Serial2 = HardwareSerial(2);

HardwareSerial Serial3 = HardwareSerial(3);

\*\*\* Các chân chức năng:

USART1\_PIN\_TX A9

USART1\_PIN\_RX A10

USART2\_PIN\_TX A2

USART2\_PIN\_RX A3

USART3\_PIN\_TX B10

USART3\_PIN\_RX B11

#### begin

Khởi động một cổng Serial với một baudrate cho trước có trên STM32f1. Để giao tiếp với máy tính, bạn phải dùng một trong các mức baudrate sau: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, hoặc 115200. Ngoài ra, bạn có thể thay thế mức baudrate khác những mức trên trong trường hợp giao tiếp với một mạch nào đó có sẵn mức baudrate xác định và không thay đổi được.

\* **Cú pháp**

Serial.begin(baud)

#### end

Đóng cổng Serial, để dùng các chân digital 0, 1 hoặc các chân tương tự cho công tác pinMode. Để bật lại, bạn dùng Serial.begin();

\* **Cú pháp**

Serial.end()

\* **Tham số**

Không.

\* **trả về**

Không.

\* **Ví dụ**

#### Write

Gửi dữ liệu tới cộng Serial. Dữ liệu được gửi bằng kiểu **uint8\_t** hoặc một dãy các **uint8\_t** Cũng như hàm Serial.print(). Các **uint8\_t** được mã hóa thành các ký tự kiểu char.

\* **Cú pháp**

Serial.write(val)

Serial.write(str)

Serial.write(buf, len)

\* **Tham số**

val: Một giá trị kiểu uint8\_t.

str: Một giá trị kiểu [string](http://arduino.vn/reference/string).

buf: Kiểu byte[] (mảng uint8\_t), len: độ dài các phần tử sẽ được xuất ra Serial, sẽ được tính từ phần tử đầu tiên đến phần tử len - 1

\* **Trả về**

số lượng byte được gửi đi.

**\* Ví dụ**

Serial.write(45); // gửi một kiểu byte có giá trị 45

int bytesSent = Serial.write("hello"); // gửi chuỗi "hello" và trả về số byte đã gửi

#### Read

Dùng để đọc từng ký tự nhận về từ serial, đây là cách không hiệu quả vì không có thể trôi dữ liệu.

\* **Cú pháp**

Serial.read()

\* **Tham số**

Không.

\* **Trả về**

Byte vừa nhận được qua cổng serial.

**\* Ví dụ**

#### print

Hàm này sẽ xuất dữ liệu ra cổng Serial dưới dạng chuỗi con người có thể đọc được. Hàm này có thể được sử dụng dưới nhiều dạng khác nhau. Các chữ số của một số (nguyên hoặc thực) được chuyển thành chuỗi và xuất ra màn hình[.](http://arduino.vn/reference/library/serial/1/huong-dan-ham/print) Ví dụ:

Serial.print(78) cho ta "78"

Serial.print(1.23456) cho ta "1.23"

Serial.print('N') cho ta "N"

Serial.print("Hello world.") cho ta "Hello world."

Tham số thứ 2 (có thể có hoặc không) sẽ giúp hệ thống in dữ liệu dưới dạng mà bạn muốn (thường là dùng để debug). Các giá trị hợp lệ là:

BIN: in dữ liệu dưới dạng hệ nhị phân (hệ cơ số 2)

DEC: in dữ liệu dưới dạng hệ thập phân (hệ cơ số 10)

OCT: in dữ liệu dưới dạng hệ bát phân (hệ cơ số 8)

HEX: in dữ liệu dưới dạng hệ thập lục phân  (hệ cơ số 16)

Còn đối với số thực, thì giá trị nhập vào là một số nguyên bất kỳ (âm hoặc dương), hệ thống sẽ dùng giá trị này để làm tròn số thực của bạn. Xem ví dụ để rõ hơn.

Serial.print(78, BIN) cho ta "1001110"

Serial.print(78, OCT) cho ta "116"

Serial.print(78, DEC) cho ta "78"

Serial.print(78, HEX) cho ta "4E"

Serial.println(1.23456, 0) cho ta "1"

Serial.println(1.23456, 2) cho ta "1.23"

Serial.println(1.23456, 4) cho ta "1.2346"

\* **Cú pháp**

Serial.print(val)

Serial.print(val, digit)

\* **Tham số**

val: bất kỳ giá trị ở bất kỳ kiểu dữ liệu nào

digit: DEC, HEX, BIN

\* **Trả về**

size\_t: [int](http://arduino.vn/reference/int) - Số byte (sau khi đã chuyển thành chuỗi) được gửi vào cổng Serial. Nó sẽ được trả về trước khi có bất kỳ giá trị nào được gửi đi vào Serial.

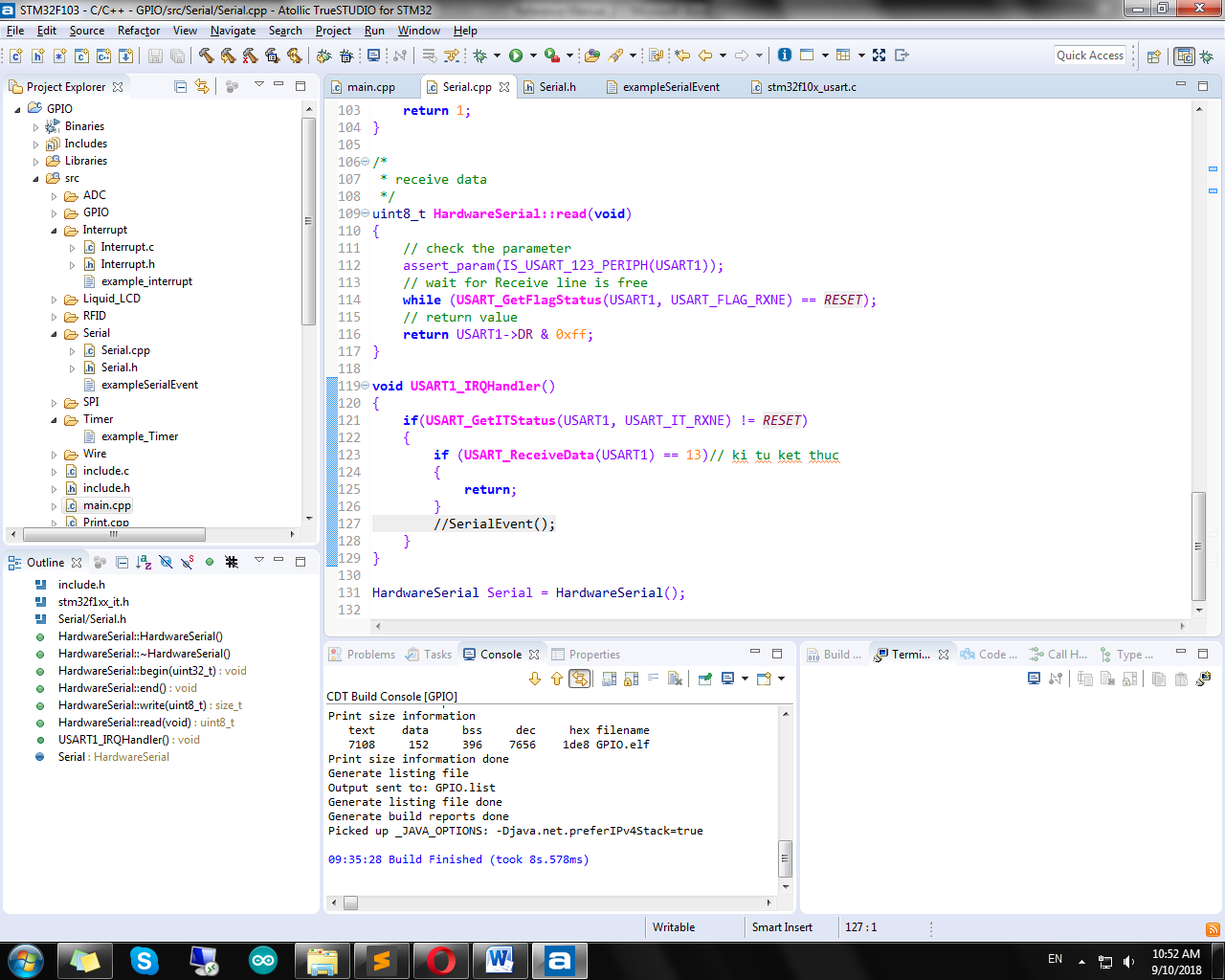
**\* Ví dụ**

#### print

Giống hệt hàm [Serial.print()](http://arduino.vn/reference/library/serial/1/huong-dan-ham/print), nhưng nó sẽ gửi thêm một dấu xuống dòng sau khi gửi những gì bạn yêu cầu.

#### Ngắt UART

Hàm serialEvent() sẽ được gọi khi nào có tín hiệu từ cổng Serial. Và sau đó bạn sẽ dùng hàm [Serial.read()](http://arduino.vn/reference/library/serial/1/huong-dan-ham/read) hoặc tương tự để đọc dữ liệu.



\* **Cú pháp**

void serialEvent()

{

ttt = Serial.read(USART1);// hàm trả về giá trị vừa nhận được từ Serial

}

### I2C

\* Ta cần phải include thư viện I2C để sử dụng:

#include <Wire/Wire.h>

* **Sơ đồ nối chân:**

I2C1\_PIN\_SCL B6

I2C1\_PIN\_SDA B7

#### begin

Khởi tạo việc sử dụng thư viện I2C1 ở chế độ master hoặc slave.

\* **Cú pháp**

Wire.begin()

Wire.begin(address)

\* **Tham số**

address: địa chỉ của Slave, nếu không nhập address thì mặc định là 0x00.

\* **Trả về**

Không.

**\* Ví dụ**

#### end

Kết thúc việc sử dụng module I2C1.

\* **Cú pháp**

Wire.end()

\* **Tham số**

Không.

\* **Trả về**

Không.

**\* Ví dụ**

#### requestFrom

Hàm này được sử dụng bởi master để yêu cầu 1 slave nào đó gởi 1 số byte theo yêu cầu. Dữ liệu có thể được lấy về bằng hàm available() và read().

Nếu việc yêu cầu hoàn tất, hàm sẽ gởi đi tín hiệu stop đồng thời thả đường truyền I2C cho các giao tiếp tiếp theo.

Nếu việc yêu cầu thất bại, hàm sẽ trả về tín hiệu reset, đường truyền I2C sẽ không được thả ra. Do đó, không thể thực hiện các giao tiếp khác từ các thiết bị khác. Điều này cho phép master có thể gởi nhiều yêu cầu trong khi đang điều khiển.

\* **Cú pháp**

Wire.requestFrom(address,quantity)

\* **Tham số**

address: địa chỉ của Slave cần yêu cầu gởi dữ liệu.

quantity: số byte dữ liệu yêu cầu.

\* **Trả về**

Uint8\_t: số byte nhận được từ slave.

**\* Ví du**

Wire.requestFrom(0x08, 6); // request 6 bytes from slave device #8

while (Wire.available()) // slave may send less than requested

{

char c = Wire.read(); // receive a byte as character

Serial.print(c); // print the character

}

delay(500);

#### beginTransmission

Hàm sử dụng để bắt đầu trao đổi với 1 thiết bị I2C slave theo địa chỉ được truyền vào. Dữ liệu sẽ được truyền thông qua hàm write() và kết thúc bởi hàm endTransmission().

\* **Cú pháp**

Wire.beginTransmission(address)

\* **Tham số**

address: địa chỉ của Slave cần giao tiếp.

\* **Trả về**

Không.

**\* Ví dụ**

Wire.beginTransmission(0x08); // giao tiếp đến thiết bị có địa chỉ 0x08

Wire.write("x is "); // gởi 5 byte dữ liệu.

Wire.write(x); // gởi 1 byte dữ liệu

Wire.endTransmission(); // kết thúc việc trao đổi.

#### endTransmission

Hàm sử dụng để kết thúc 1 quá trình trao đổi đã được tạo bởi hàm beginTranmission().

Nếu việc yêu cầu hoàn tất, hàm sẽ gởi đi tín hiệu stop đồng thời thả đường truyền I2C cho các giao tiếp tiếp theo.

Nếu việc yêu cầu thất bại, hàm sẽ trả về tín hiệu reset, đường truyền I2C sẽ không được thả ra. Do đó, không thể thực hiện các giao tiếp khác từ các thiết bị khác. Điều này cho phép master có thể gởi nhiều yêu cầu trong khi đang điều khiển.

\* **Cú pháp**

Wire.endTransmission()

\* **Tham số**

Không.

\* **Trả về**

Trả về 0 nếu quá trình hoàn tất.

**\* Ví dụ**

#### write

Gởi dữ liệu từ master đến slave giữa 2 hàm beginTransmission() và endTransmission(), hoặc gởi dữ liệu từ slave khi có yêu cầu bởi hàm requestFrom từ master.

\* **Cú pháp**

Wire.write(value)   
Wire.write(string)   
Wire.write(data, length)

\* **Tham số**

Không.

\* **Trả về**

Trả về 0 nếu quá trình hoàn tất.

**\* Ví dụ**

#### available

Trả về số byte có sẵn để có thể đọc được bằng hàm read(). Hàm này nên được gọi từ master sau khi đã sử dụng hàm requestFrom() hoặc sử dụng trên ngắt onReceive() của slave.

\* **Cú pháp**

Wire.available()

\* **Tham số**

Không.

\* **Trả về**

Số byte dữ liệu có sẵn để đọc.

**\* Ví du**

void receiveEvent()

{

while (1 < Wire.available())

{

char c = Wire.read(); // nhận 1 byte theo dạng kí tự

Serial.print(c); // in kí tự

}

int x = Wire.read(); // nhận byte theo kiểu int

Serial.println(x); // in giá trị x

}

#### read

Đọc 1 byte dữ liệu được gởi từ slave đến master sau khi master yêu cầu gởi bằng hàm requestFrom() hoặc đọc 1 byte dữ liệu được master gởi đến slave thông qua ngắt nhận I2C (onReceive()).

\* **Cú pháp**

Wire.read()

\* **Tham số**

Không.

\* **Trả về**

Byte tiếp theo đọc được.

**\* Ví dụ**

Wire.requestFrom(0x08, 6); // request 6 bytes from slave device #8

while (Wire.available()) // slave may send less than requested

{

char c = Wire.read(); // receive a byte as character

Serial.print(c); // print the character

}

delay(500);

#### onReceive

“Đăng kí” 1 hàm mà hàm này được thược hiện khi thiết bị slave nhận được một yêu cầu gởi dữ liệu đến từ master.

\* **Cú pháp**

Wire. onReceive(handler)

\* **Tham số**

Handler: có cú pháp: void myHandler(void).

\* **Trả về**

Không trả về.

**\* Ví dụ**

#### onRequest

“Đăng kí” 1 hàm mà hàm này được thược hiện khi thiết bị slave nhận được một yêu cầu gởi dữ liệu cho master.

\* **Cú pháp**

Wire. onRequest(handler)

\* **Tham số**

Handler: có cú pháp: void myHandler(void).

\* **Trả về**

Không trả về.

**\* Ví dụ**

### SPI

* **Sơ đồ nối chân**

\* Ta cần include thư viện SPI để sử dụng:

#include <SPI/SPIClass.h>

// SPI1 define PIN/GPIO

SPI1\_PIN\_NSS A4

SPI1\_PIN\_SCK A5

SPI1\_PIN\_MISO A6

SPI1\_PIN\_MOSI A7

// SPI2 define PIN/GPIO

SPI2\_PIN\_NSS B12

SPI2\_PIN\_SCK B13

SPI2\_PIN\_MISO B14

SPI2\_PIN\_MOSI B15

Tương tự USART, muốn sử dụng SPI2, ta cần khởi tạo ngay từ đầu:

\* **Cú pháp**

SPIClass SPIName = SPIClass (USART);

\* **Tham số**

SPI: số thứ tự của SPI: = 2 với SPI2.

\* SPI1 (SPIName là SPI) mặc định có SPI = 1.

\* **Trả về**

Không.

**\* Ví dụ**

SPIClass SPI2 = SPIClass(2);

#### begin

Khởi tạo việc sử dụng thư viện SPI.

\* **Cú pháp**

SPI.begin(mode)

\* **Tham số**

address: - SPI\_Mode\_Slave: thiết lập SPI1 ở chế độ slave.

- SPI\_Mode\_Master: thiết lập SPI1 ở chế độ master.

\* **Trả về**

Không.

**\* Ví dụ**

#### end

Kết thúc việc sử dụng module I2C1.

\* **Cú pháp**

SPI.end()

\* **Tham số**

Không.

\* **Trả về**

Không.

**\* Ví dụ**

#### transfer

Hàm transfer được sử dụng bởi master để gởi 1 byte đến slave và nhận về 1 byte từ slave đó. Hàm này là sự kết hợp giữa hàm write() và hàm read().

Trước khi gọi hàm này, ta cần xác định slave sẽ trao đổi dữ liệu bằng các kéo chân SS chủa slave đó xuống mức thấp, và để kết thúc quá trình cần thả chân SS về lại mức cao.

\* **Cú pháp**

returnVal = SPI.transfer(val)

\* **Tham số**

val: giá trị cần gởi đến slave kiểu uint8\_t.

\* **Trả về**

returnVal: giá trị trả về kiểu uint8\_t.

**\* Ví dụ**

digitalWrite(SPI1\_GPIO, SPI1\_PIN\_NSS, LOW);

a = SPI.transfer(0);

delayMicroseconds (50);

digitalWrite(SPI1\_GPIO, SPI1\_PIN\_NSS, HIGH);

delay(1000);

Serial.println(a);

#### write & read

Hàm read() dùng để slave đọc 1 byte dữ liệu từ master. Nó được sử dụng trong ngắt của slave để nhận dữ liệu từ master.

Hàm write() dùng để gởi 1 byte dữ liệu từ master đến slave mà không nhận lại dữ liệu từ slave. Nó được sử dụng trong ngắt của slave để phản hồi dữ liệu cho master.

Sự kết hợp giữa hàm read() và write của SPI giúp cho slave có thể giao tiếp với master khi master sử dụng hàm transfer() một các hiệu quả nhất.

\* **Cú pháp:**

SPI.write(val)

returnVal = SPI.read()

\* **Tham số**

val: giá trị cần gởi.

\* **Trả về**

returnVal: giá trị nhận về.

**\* Ví dụ**

void spiEvent()

{

uint8\_t data = SPI.read();

Serial.println(data);

SPI.write(45);

}

#### attachInterrupt

“Đăng kí” 1 hàm mà hàm này được thược hiện khi thiết bị slave nhận được tín hiệu trao đổi.

\* **Cú pháp**

SPI. attachInterrupt (handler)

\* **Tham số**

Handler: có cú pháp: void myHandler(void)

\* **Trả về**

Không.

**\* Ví dụ**

void spiEvent()

{

uint8\_t data = SPI.read();

Serial.println(data);

SPI.write(45);

}

int main(void)

{

SPI.attachInterrupt(spiEvent);

while(1)

}

#### detachInterrupt

“Hủy đăng kí” việc ngắt nhận dữ liệu trên slave.

\* **Cú pháp**

SPI. detachInterrupt ()

\* **Tham số**

Không.

\* **Trả về**

Không.

**\* Ví dụ**