**Giao thức I2C trong PIC16F877A**

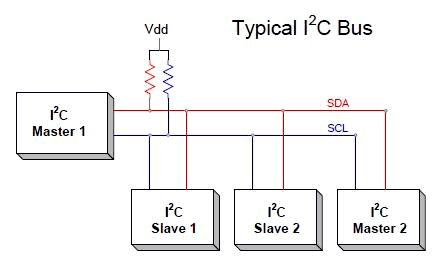
**I, Sơ lược về giao thức I2C**

Thuật ngữ I2C là viết tắt của “**Inter Integrated Circuits**”. Ngoài “I2C”, giao thức này còn được ký hiệu là “I2C”, hoặc “TWI” (Two Wires Interface).

I2C là 1 giao thức truyền thông đồng bộ, có nghĩa là cả 2 thiết bị đang chia sẻ thông tin phải sử dụng chung 1 clock. Giao thức này chỉ có 2 dây để chia sẻ thông tin, trong đó 1 dây dành cho clock, 1 dây dành cho data.

**II, Giao thức I2C hoạt động như thế nào?**

Như đã trình bày ở trên, giao thức này sẽ có 2 dây, 2 dây được kết nối giữa ít nhất 2 thiết bị. Ở đây 1 thiết bị gọi là “Master” và 1 thiết bị gọi là “Slave”. Ưu điểm của giao thức I2C là có thể kết nối nhiều Slave với 1 Master.



Giao thức sẽ diễn ra hoàn chỉnh thông qua 2 dây:

* **Serial Clock (SCL):** chia sẻ xung clock được tạo ra bởi Master cho Clock
* **Serial Data (SDA):** gửi dữ liệu đến và đi giữa Master và Slave

Tại bất kỳ thời điểm nào, chỉ có Master mới có thể bắt đầu giao tiếp. Vì có nhiều Slave trong bus I2C, Master phải tham chiếu đến từng Slave bằng cách sử dụng các địa chỉ khác nhau của các Slave.

**III, Giao thức I2C trong PIC16F877A**

Trong PIC16F877A, module **Master Synchronous Serial Port (MSSP)** là 1 modue giao diện nối tiếp, hữu ích để giao tiếp với thiết bị ngoại vi hoặc vi điều khiển khác. Module này có thể thực hiện được 2 giao thức:

* Serial Peripheral Interface (SPI)
* Inter – Integrated Circuit (I2C): Với giao thức I2C, 3 chế độ được hỗ trợ

+, Chế độ Master

+, Chế độ Slave

+, Chế độ nhiều Master

Ở bài viết này, chúng ta sẽ chỉ làm quen với giao thức I2C chế độ Master. 2 chân sẽ được sử dụng là:

* **Serial clock (SCL): RC3/SCK/SCL**
* **Serial data (SDA): RC4/SDI/SDA**

**1, Các thanh ghi trong chế độ I2C**

Module MSSP có 6 thanh ghi cho chế độ I2C:

1. MSSP Control Register (**SSPCON**)
2. MSSP Control Register 2 (**SSPCON2**)
3. MSSP Status Register (**SSPSTAT**)
4. Serial Receive/Transmit Buffer Register (**SSPBUF**)
5. MSSP Shift Register (**SSPSR**) – không thể truy cập trực tiếp thanh ghi này
6. MSSP Address Register (**SSPADD**)

**SSPCON** và **SSPCON2** là 2 thanh ghi điều khiển hoạt động trong chế độ I2C. Hai thanh ghi này đều có thể đọc và ghi dữ liệu được.

**SSPSTAT** là thanh ghi trạng thái, 6bits thấp của thanh ghi này là các bit read-only, 2bits cao đều có thể đọc/ghi.

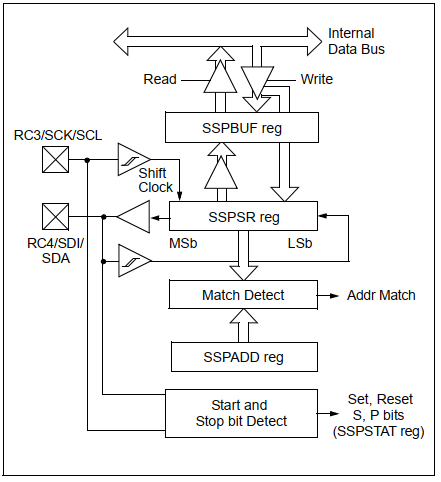
**SSPSR** là thanh ghi dịch, được sử dụng để dịch chuyển dữ liệu vào hoặc ra. **SSPBUF** là thanh ghi buffer, nơi data byte được viết lên hoặc đọc vào.

Trong quá trình nhận dữ liệu, **SSPSR** và **SSPBUF** sẽ tạo thành 1 cặp buffer để nhận dữ liệu. Khi **SSPSR** hoàn thành việc nhận dữ liệu, nó sẽ truyền dữ liệu đến thành ghi **SSPBUF** và cờ ngắt **SSPIF** sẽ được set lên mức High.

Trong quá trình truyền, **SSPBUF** và **SSPSR** không phải 1 cặp buffer. Mỗi khi viết dữ liệu lên **SSPBUF** sẽ viết lên cả 2 thanh ghi **SSPBUF** và **SSPSR**.

Thanh ghi **SSPADD** lưu địa chỉ của MCU trong chế độ Slave. Trong chế độ Master, thanh ghi này dùng để thiết lập độc tộ baud.

Hình ảnh dưới đây thể hiện tổng quan Module MSSP trong chế độ I2C:



**2, Các hàm cần thiết để sử dụng giao thức I2C trong PIC16F877A**

**Hàm khởi tạo chế độ I2C**

void I2C\_Init**(**const unsigned long freq\_KHz**)** //freq\_KHz = 100KHz ~ 1MHz

**{**

int SSPADD\_value **=** **(**8000000 **/** **(**4**\***freq\_KHz**))** **-** 1**;**

TRISC**.**B3 **=** 1**;** //set SDA and SCL pins as input

TRISC**.**B4 **=** 1**;**

/\*

SSPCON = 0010 1000

Enables the serial port

Configures the SDA and SCL pins as the serial port pins

Select I2C Master mode

\*/

SSPCON **=** 0x28**;**

SSPCON2 **=** 0x00**;**

SSPADD **=** SSPADD\_value**;**

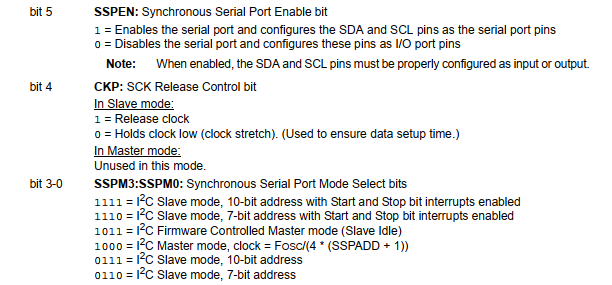
SSPSTAT **=** 0x00**;** //clear SSPSTAT register

**}**

Hàm này được sử dụng để khởi tạo chế độ I2C. Tham số truyền vào sẽ là mức xung clock mà chúng ta muốn sử dụng.

Điều đầu tiên cần làm là khai báo chân **PC3** và **PC4** ở chế độ Input.

Chúng ta sẽ setting các bit của thanh ghi **SSPCON** và **SSPCON2**. Tạm thời, chỉ cần quan tâm đến bit **SSPEN** và bit <3:0> (**SSPM3 – SSPM0**). Set bit SSPEN lên mức 1 để enable chân SDA và SCL thành serial port pin. Cấu hình cho 4bits cuối thành ‘1000’ để chọn chế độ I2C Master.



Chính vì chọn chế độ I2C master, nên ta phải thiết lập giá trị cho thanh ghi **SSPADD.** Dựa vào công thức trong datasheet **clock = Fosc / (4\*(SSPADD + 1))**, chúng ta sẽ thiết lập được giá trị để truyền vào thanh ghi **SSPADD**:

int SSPADD\_value **=** **(**8000000 **/** **(**4**\***freq\_KHz**))** **-** 1**;**

**Hàm đợi I2C\_Hold()**

void I2C\_Hold**(**void**)**

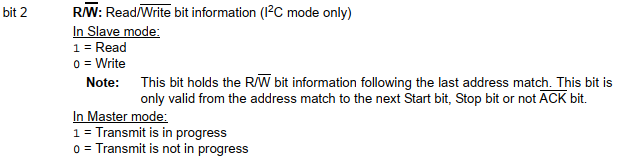
**{**

**while(** **(**SSPCON2 **&** 0x1F**)** **||** **(**SSPSTAT **&** 0x04**)** **);** //check I2C is not in progress

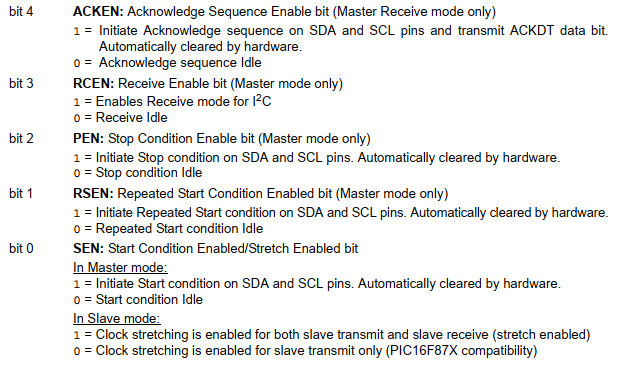
**}**

Đây là 1 hàm quan trọng, hàm này được sửa dụng để giúp cho thiết bị thực thi nhiệm vụ đến khi 1 quá trình giao tiếp I2C hoàn thành. Chúng ta sẽ phải kiểm tra hoạt động của I2C trước khi muốn thực hiện bất kỳ 1 hành động mới nào.

Ở đây, chúng ta sử dụng phép ‘AND’ và ‘OR’ để kiểm tra, mục đích là chắc chắn rằng bit **SSPSTAT<2>** và 5bits **SSPCON2<0:4>** đều đang ở mức 0. Hai hình ảnh dưới đây sẽ mô tả chi tiết về nhiệm vụ của các bit này:



**Bit SSPSTAT<2>**



**SSPCON2<0:4>**

**Hàm bắt đầu và kết thúc 1 quá trình giao tiếp I2C**

Mỗi lần đọc hoặc ghi dữ liệu trên bus I2C, chúng ta đều nên bắt đầu và kết thúc 1 quá trình kết nối. Để bắt đầu 1 giao tiếp I2C, bit **SEN** (**SSPCON2<0>)** cần được set lên mức cao. Tương tự như vậy, nếu muốn kết thúc 1 giao tiếp, chúng ta phải set bit **PEN** (**SSPCON2<2>)** lên mức cao.

Lưu ý, trước khi thực hiện bất kỳ thay đổi nào trên các bit này, đều phải kiểm tra xem bus I2C có đang bận không bằng hàm **I2C\_Hold()**

void I2C\_Start**(**void**)**

**{**

I2C\_Hold**();**

SEN\_bit **=** 1**;** //Initiate Start condition on SDA and SCL pins

**}**

void I2C\_End**(**void**)**

**{**

I2C\_Hold**();**

PEN\_bit **=** 1**;** //Initiate Stop condition on SDA and SCL pins

**}**

**Hàm gửi dữ liệu**

void I2C\_Write**(**unsigned char dataSend**)**

**{**

I2C\_Hold**();**

SSPBUF **=** dataSend**;**

**}**

Hàm này sẽ gửi 1byte data từ MCU đến thiết bị Slave, nó phải được sử dụng sau hàm **I2C\_Start()** và trước hàm **I2C\_End()**. Data truyền vào hàm sẽ được load đến thanh ghi **SSPBUF** và gửi đến I2C bus.

Thông thường trước khi truyền đi 1 data bất kỳ, chúng ta sẽ phải gửi đi 1 địa chỉ để tìm đến Slave. Vì thế, hàm này sẽ được sử dụng ít nhất 2 lần trong 1 quá trình giao tiếp, trong đó lần đầu tiên sử dụng để thiếp lập địa chỉ, các lần sau để gửi data.

**Hàm nhận dữ liệu**

/\*

ack = 0 => Not Acknowledge

ack = 1 => Acknowledge

\*/

unsigned char I2C\_Read**(**unsigned char ack**)**

**{**

unsigned char inComing**;**

I2C\_Hold**();**

RCEN\_bit **=** 1**;** //enable receive mode for I2C

I2C\_Hold**();**

inComing **=** SSPBUF**;** //load data saved in SSPBUF to inComing

I2C\_Hold**();**

ACKDT\_bit **=** **(**ack**)** **?** 0 **:** 1**;** //check if ack bit received

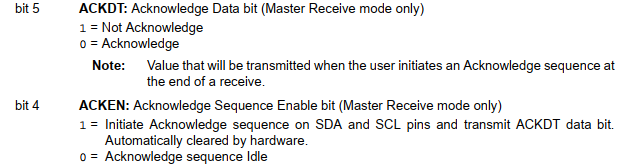
ACKEN\_bit **=** 1**;** //Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit

**return** inComing**;**

**}**

Hàm này được sử dụng để đọc dữ liệu trên bus I2C. Giá trị nhận về sẽ được lưu trong thành ghi **SSPBUF**, chúng ta sẽ chuyển giá trị đó về biến ‘inComing’ để return. Tương tự như hàm **I2C\_Write()**, hàm này cũng cần phải đặt giữa 2 hàm **I2C\_Start()** và **I2C\_End().**

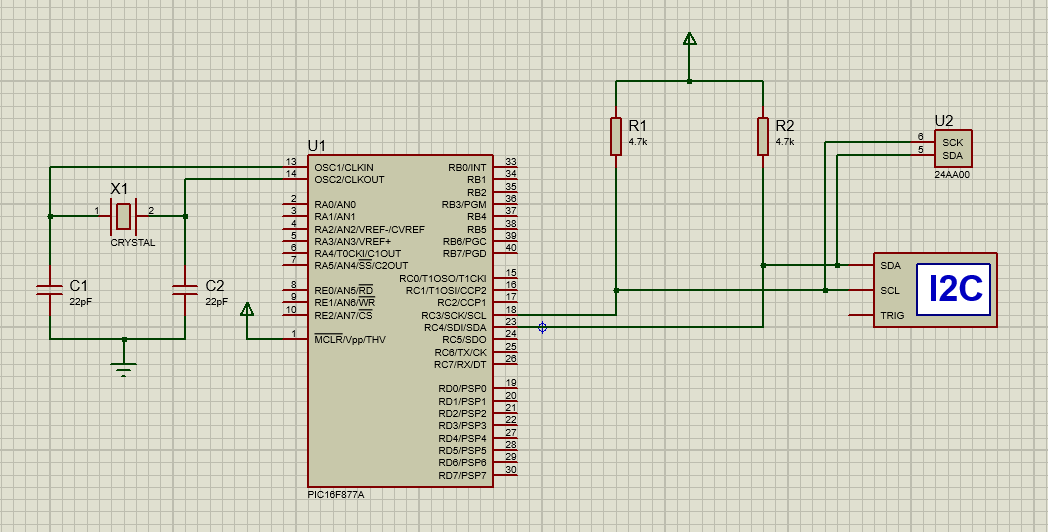
Trong quá trình giao tiếp I2C, Slave sau khi gửi dữ liệu xong sẽ gửi 1bit khác là bit Ack, bit này cũng cần được Master kiểm tra để đảm bảo giao tiếp thành công. Sau khi kiểm tra bit **ACKDT** (**SSPCON2<5>**), nó sẽ được kích hoạt bằng cách set bit **ACKEN** (**SSPCON2<4>**) lên mức High. Chi tiết sẽ được mô tả trong hình dưới đây:



**3, Thực hành với mô phỏng Proteus**

Như vậy, chúng ta đã tìm hiểu xong các hàm cơ bản cho giao thức I2C trong PIC16F877A. Sau đây, mình sẽ thực hành 1 ví dụ là truyền và nhận dữ liệu thông qua giao thức I2C giữa PIC16F877A và IC 24AA00.

**Sơ đồ nguyên lý:**



Mình sẽ sử dụng I2C Debuger trong Proteus để theo dõi quá trình truyền, nhận giữa MCU và 24AA00.

**Code hàm main()**

void main()

{

unsigned char data\_Recv = 0;

I2C\_Init(100000); //Init clock 100KHz

while(1)

{

I2C\_Start();

I2C\_Write(0xA0); //using 24AA00 be slave

I2C\_Write(0x01); //write into register

I2C\_Write('y'); //send 'y' to Slave

I2C\_End();

Delay\_ms(1000);

I2C\_Start();

I2C\_Write(0xA1);

data\_Recv = I2C\_Read(0);

I2C\_End();

Delay\_ms(3000);

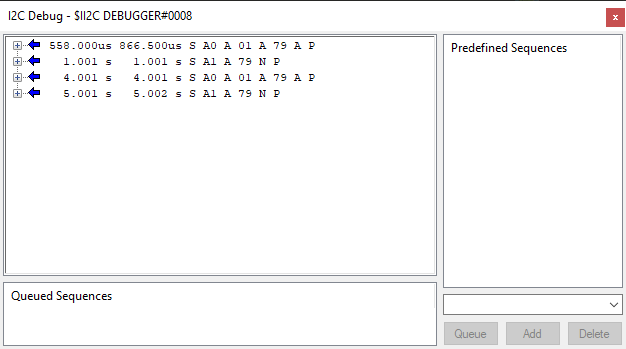
}

}

Mình sẽ khởi tạo xung clock là 100KHz. ‘0xA0’ chính là địa chỉ Slave của IC 24AA00, câu lệnh I2C\_Write(0xA0); dùng để gọi đến địa chỉ của IC này, sau đó chúng ta cần truyền vào thanh ghi của IC này 1 lệnh ‘0x01’ để IC tiến hành đọc dữ liệu. Và dữ liệu truyền đi sẽ là chữ ‘y’.

Tiếp theo, mình sẽ tiến hàm đọc dữ liệu từ IC 24AA00, câu lệnh I2C\_Write(0xA1); dùng để gọi đến địa chỉ của IC này (0xA1) nếu Master muốn nhận dữ liệu từ nó.

**Kết quả mô phỏng:**



Khi tiến hành chạy mô phỏng, cửa sổ I2C Debug hiện lên và cho ta thấy kết quả như trên. Ta có thể nhận thấy ‘**A0’** và ‘**A1’** là địa chỉ mà Master kết nối được, ‘**01’** và ‘**79’** (mã hexa của chữ ‘y’) là data mà ta gửi đến địa chỉ ‘**0xA0’**. Và ‘**79’** cũng là data Master nhận về từ Slave với địa chỉ ‘**0xA1’**.

Giải thích về các chữ ký hiệu trên I2C Debug:

‘S’: Start

‘A’: Acknowledge

‘N’: Not Acknowledge

‘P’: Stop

Hy vọng qua bài viết này, các bạn có thể tự xây dựng cho mình 1 thư viện I2C với PIC cũng như các vi điều khiển khác, qua đó vận dụng để thực hành với các dự án thực tế khác.

Chúc các bạn thành công!!!