

TRƯỜNG SĨ QUAN CHKT THÔNG TIN

NGUYEN VĂN TÌNH

KHOÁ DH12

HỆ ĐÀO TẠO DÀI HẠN ĐẠI HỌC

TÀI LIỆU VI ĐIỀU KHIỂN

PIC 16F877A

NĂM 2008

SI QUAN CHI HUY KI THUẬT THÔNG TIN

LỜI NÓI ĐẦU

Chào các đồng chí. Mình tên là Nguyễn Văn Tình là học viên tiểu đoàn 18 khóa ĐH12 .Sau nhiều thời gian nghiên cứu về vi điều khiển, và nhận thấy là chúng ta học ĐIỆN TỬ VIỄN THÔNG nhưng chúng ta chưa được các thầy dạy nhiều về vi điều khiển .Chính vì điều đó nên hôm nay mình quyết định cho xuất bản quyển sách về một loại vi điều khiển mà nó có những tính năng vượt trội so với nhiều dòng vi điều khiển trước như: 8051, AVR, P89v51... Đó là dòng vi điều khiển của hãng MICROCHIP .Vi điều khiển hiện nay cũng như tương lai là có rất nhiều ứng dụng trong cuộc sống, không chỉ cho những đồng chí thích nghiên cứu về tự động hóa hay về robocon ...v.v..Vi hiện tại là dòng vi điều khiển này cũng chưa được đưa vào dạy trong các trường đại học mà đa số các trường chỉ dạy về 8051 và cũng chưa có tài liệu chính thống nào. Nay mình cho ra đời quyển sách này với mục đích giúp chúng ta tiếp cận công nghệ mới chứ không để lạc hậu so với những sinh viên bên ngoài. Vì nhiều lý do và khả năng có hạn nên quá trình biên soạn có nhiều sai sót. mong các đồng chí đọc và cho ý kiến để chúng ta cùng sửa chữa. Mọi thắc mắc xin liên hệ với mình NGUYỄN VĂN TÌNH c3/d18/dh12h. Hay qua email henlagka@yahoo.com. DT:0583743625

Nha Trang ,ngay 25 tháng 12 năm 2008

TaiLieuTuoit.com

MỤC LỤC

CHƯƠNG 1 TỔNG QUAN VỀ VI ĐIỀU KHIỂN PIC

- 1.1 PIC LÀ GÌ ??
- 1.2 TẠI SAO LÀ PIC MÀ KHÔNG LÀ CÁC HỌ VI ĐIỀU KHIỂN KHÁC??
- 1.3 KIẾN TRÚC PIC
- 1.4 RISC VÀ CISC
- 1.5 PIPELINING
- 1.6 CÁC DÒNG PIC VÀ CÁCH LỰA CHỌN VI ĐIỀU KHIỂN PIC
- 1.7 NGÔN NGỮ LẬP TRÌNH CHO PIC
- 1.8 MẠCH NẠP PIC
- 1.9 BOOTLOADER VÀ ICP (In Circuit Programming)

CHƯƠNG 2 VI ĐIỀU KHIỂN PIC16F877A

- 2.1 SƠ ĐỒ CHÂN VI ĐIỀU KHIỂN PIC16F877A
- 2.2 MỘT VÀI THÔNG SỐ VỀ VI ĐIỀU KHIỂN PIC16F877A
- 2.3 SƠ ĐỒ KHỐI VI ĐIỀU KHIỂN PIC16F877A
- 2.4 TỔ CHỨC BỘ NHỚ
 - 2.4.1 BỘ NHỚ CHƯƠNG TRÌNH
 - 2.4.2 BỘ NHỚ DỮ LIỆU
 - 2.4.2.1 THANH GHI CHÚC NĂNG ĐẶC BIỆT SFR
 - 2.4.2.2 THANH GHI MỤC ĐÍCH CHUNG GPR
 - 2.4.3 STACK
- 2.5 CÁC CỔNG XUẤT NHẬP CỦA PIC16F877A
 - 2.5.1 PORTA
 - 2.5.2 PORTB
 - 2.5.3 PORTC
 - 2.5.4 PORTD
 - 2.5.5 PORTE
- 2.6 TIMER 0
- 2.7 TIMER1
- 2.8 TIMER2
- 2.9 ADC
- 2.10 COMPARATOR
 - 2.10.1 BỘ TẠO ĐIỆN ÁP SO SÁNH
- 2.11 CCP
- 2.12 GIAO TIẾP NỐI TIẾP

- 1.12.1 USART
 - 2.12.1.1 USART BẤT ĐỒNG BỘ
 - 2.12.1.1.1 TRUYỀN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART BẤT ĐỒNG BỘ
 - 2.12.1.1.2 NHẬN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART BẤT ĐỒNG BỘ
 - 2.12.1.1.2 USART ĐỒNG BỘ
 - 2.12.1.2.1 TRUYỀN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ MASTER MODE
 - 2.12.1.2.2 NHẬN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ MASTER MODE
 - 2.12.1.2.3 TRUYỀN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ SLAVE MODE
 - 2.12.1.2.4 NHẬN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ SLAVE MODE
- 2.12.2 MSSP
 - 2.12.2.1 SPI
 - 2.12.2.1.1 SPI MASTER MODE
 - 2.12.2.1.2 SPI SLAVE MODE
 - 2.12.2.2 I2C
 - 2.12.2.2.1 I2C SLAVE MODE
 - 2.12.2.2.2 I2C MASTER MODE
- 2.13 CỔNG GIAO TIẾP SONG SONG PSP (PARALLEL SLAVE PORT)
- 2.14 TỔNG QUAN VỀ MỘT SỐ ĐẶC TÍNH CỦA CPU.
 - 2.14.1 CONFIGURATION BIT
 - 2.14.2 CÁC ĐẶC TÍNH CỦA OSCILLATOR
 - 2.14.3 CÁC CHẾ ĐỘ RESET
 - 2.14.4 NGẮT (INTERRUPT)
 - 2.14.4.1 NGẮT INT
 - 2.14.4.2 NGẮT DO SỰ THAY ĐỔI TRẠNG THÁI CÁC PIN TRONG PORTB
 - 2.14.5 WATCHDOG TIMER (WDT)
 - 2.14.6 CHẾ ĐỘ SLEEP
 - 2.14.6.1 “ĐÁNH THỨC” VI ĐIỀU KHIỂN

CHƯƠNG 3 TẬP LỆNH CỦA VI ĐIỀU KHIỂN PIC

- 3.1 VÀI NÉT SƠ LƯỢC VỀ TẬP LỆNH CỦA VI ĐIỀU KHIỂN PIC
- 3.2 TẬP LỆNH CỦA VI ĐIỀU KHIỂN PIC
- 3.3 CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH ASSEMBLY VIẾT CHO VI ĐIỀU KHIỂN PIC

CHƯƠNG 4 MỘT SỐ ỨNG DỤNG CỤ THỂ CỦA PIC16F877A

4.1 ĐIỀU KHIỂN CÁC PORT I/O

4.1.1 CHƯƠNG TRÌNH DELAY

4.1.2 MỘT SỐ ỨNG DỤNG VỀ ĐẶC TÍNH I/O CỦA CÁC PORT ĐIỀU KHIỂN

4.2 VI ĐIỀU KHIỂN PIC16F877A VÀ IC GHI DỊCH 74HC595

4.3 PIC16F877A VÀ LED 7 ĐOẠN

4.4 NGẮT VÀ CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH NGẮT

4.5 TIMER VÀ ỨNG DỤNG

4.5.1 TIMER VÀ HOẠT ĐỘNG ĐỊNH THỜI

PHỤ LỤC 1 SƠ ĐỒ KHỐI CÁC PORT CỦA VI ĐIỀU KHIỂN PIC16F877A

PHỤ LỤC 2 THANH GHI SFR (SPECIAL FUNCTION REGISTER)

CHƯƠNG 1 TỔNG QUAN VỀ VI ĐIỀU KHIỂN PIC

1.1 PIC LÀ GÌ ??

PIC là viết tắt của “Programable Intelligent Computer”, có thể tạm dịch là “máy tính thông minh khả trình” do hãng Genenral Instrument đặt tên cho vi điều khiển đầu tiên của họ: PIC1650 được thiết kế để dùng làm các thiết bị ngoại vi cho vi điều khiển CP1600. Vi điều khiển này sau đó được nghiên cứu phát triển thêm và từ đó hình thành nên dòng vi điều khiển PIC ngày nay.

1.2 TẠI SAO LÀ PIC MÀ KHÔNG LÀ CÁC HỌ VI ĐIỀU KHIỂN KHÁC??

Hiện nay trên thị trường có rất nhiều họ vi điều khiển như 8051, Motorola 68HC, AVR, ARM,... Ngoài họ 8051 được hướng dẫn một cách căn bản ở môi trường đại học, bản thân người viết đã chọn họ vi điều khiển PIC để mở rộng vốn kiến thức và phát triển các ứng dụng trên công cụ này vì các nguyên nhân sau:

Họ vi điều khiển này có thể tìm mua dễ dàng tại thị trường Việt Nam.

Giá thành không quá đắt.

Có đầy đủ các tính năng của một vi điều khiển khi hoạt động độc lập.

Là một sự bổ sung rất tốt về kiến thức cũng như về ứng dụng cho họ vi điều khiển mang tính truyền thống: họ vi điều khiển 8051.

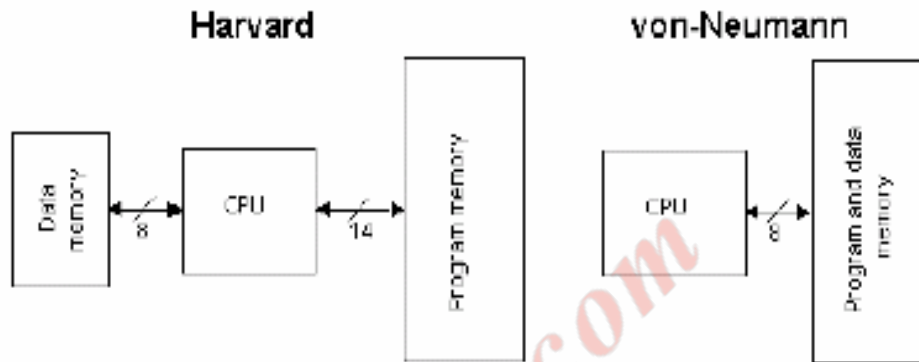
Số lượng người sử dụng họ vi điều khiển PIC. Hiện nay tại Việt Nam cũng như trên thế giới, họ vi điều khiển này được sử dụng khá rộng rãi. Điều này tạo nhiều thuận lợi trong quá trình tìm hiểu và phát triển các ứng dụng như: số lượng tài liệu, số lượng các ứng dụng mở đã được phát triển thành công, dễ dàng trao đổi, học tập, dễ dàng tìm được sự chỉ dẫn khi gặp khó khăn,...

Sự hỗ trợ của nhà sản xuất về trình biên dịch, các công cụ lập trình, nạp chương trình từ đơn giản đến phức tạp,...

Các tính năng đa dạng của vi điều khiển PIC, và các tính năng này không ngừng được phát triển.

1.3 KIẾN TRÚC PIC

Cấu trúc phần cứng của một vi điều khiển được thiết kế theo hai dạng kiến trúc: kiến trúc Von Neuman và kiến trúc Havard.



Hình 1.1: Kiến trúc Havard và kiến trúc Von-Neuman

Tổ chức phần cứng của PIC được thiết kế theo kiến trúc Havard. Điểm khác biệt giữa kiến trúc Havard và kiến trúc Von-Neuman là cấu trúc bộ nhớ dữ liệu và bộ nhớ chương trình.

Đối với kiến trúc Von-Neuman, bộ nhớ dữ liệu và bộ nhớ chương trình nằm chung trong một bộ nhớ, do đó ta có thể tổ chức, cân đối một cách linh hoạt bộ nhớ chương trình và bộ nhớ dữ liệu. Tuy nhiên điều này chỉ có ý nghĩa khi tốc độ xử lý của CPU phải rất cao, vì với cấu trúc đó, trong cùng một thời điểm CPU chỉ có thể tương tác với bộ nhớ dữ liệu hoặc bộ nhớ chương trình. Như vậy có thể nói kiến trúc Von-Neuman không thích hợp với cấu trúc của một vi điều khiển.

Đối với kiến trúc Havard, bộ nhớ dữ liệu và bộ nhớ chương trình tách ra thành hai bộ nhớ riêng biệt. Do đó trong cùng một thời điểm CPU có thể tương tác với cả hai bộ nhớ, như vậy tốc độ xử lý của vi điều khiển được cải thiện đáng kể.

Một điểm cần chú ý nữa là tập lệnh trong kiến trúc Havard có thể được tối ưu tùy theo yêu cầu kiến trúc của vi điều khiển mà không phụ thuộc vào cấu trúc dữ liệu. Ví dụ, đối với vi điều khiển dòng 16F, độ dài lệnh luôn là 14 bit (trong khi dữ liệu được tổ chức thành từng byte), còn đối với kiến trúc Von-Neuman, độ dài lệnh luôn là bội số của 1 byte (do dữ liệu được tổ chức thành từng byte). Đặc điểm này được minh họa cụ thể trong hình 1.1.

1.4 RISC và CISC

Như đã trình bày ở trên, kiến trúc Havard là khái niệm mới hơn so với kiến trúc Von-Neuman. Khái niệm này được hình thành nhằm cải tiến tốc độ thực thi của một vi điều khiển. Qua việc tách rời bộ nhớ chương trình và bộ nhớ dữ liệu, bus chương trình và bus dữ liệu, CPU có thể cùng một lúc truy xuất cả bộ nhớ chương trình và bộ nhớ dữ liệu, giúp tăng tốc độ xử lý của vi điều khiển lên gấp đôi. Đồng thời cấu trúc lệnh không còn phụ thuộc vào cấu trúc dữ liệu nữa mà có thể linh động điều chỉnh tùy theo khả năng và tốc độ của từng vi điều

khẩn. Và để tiếp tục cải tiến tốc độ thực thi lệnh, tập lệnh của họ vi điều khiển PIC được thiết kế sao cho chiều dài mã lệnh luôn cố định (ví dụ đối với họ 16Fxxxx chiều dài mã lệnh luôn là 14 bit) và cho phép thực thi lệnh trong một chu kỳ của xung clock (ngoại trừ một số trường hợp đặc biệt như lệnh nhảy, lệnh gọi chương trình con ... cần hai chu kỳ xung đồng hồ). Điều này có nghĩa tập lệnh của vi điều khiển thuộc cấu trúc Harvard sẽ ít lệnh hơn, ngắn hơn, đơn giản hơn để đáp ứng yêu cầu mã hóa lệnh bằng một số lượng bit nhất định.

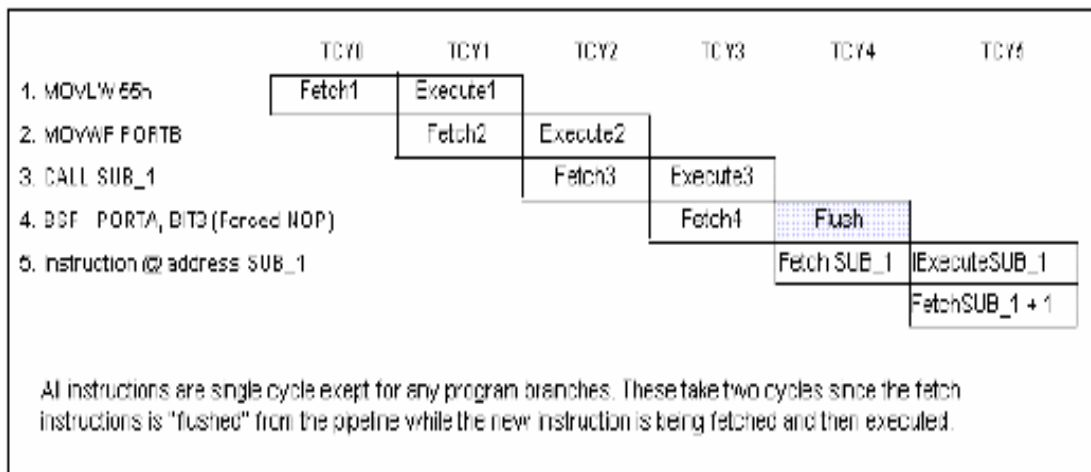
Vi điều khiển được tổ chức theo kiến trúc Harvard còn được gọi là vi điều khiển RISC (Reduced Instruction Set Computer) hay vi điều khiển có tập lệnh rút gọn. Vi điều khiển được thiết kế theo kiến trúc Von-Neuman còn được gọi là vi điều khiển CISC (Complex Instruction Set Computer) hay vi điều khiển có tập lệnh phức tạp vì mã lệnh của nó không phải là một số cố định mà luôn là bội số của 8 bit (1 byte).

1.5 PIPELINING

Đây chính là cơ chế xử lý lệnh của các vi điều khiển PIC. Một chu kỳ lệnh của vi điều khiển sẽ bao gồm 4 xung clock. Ví dụ ta sử dụng oscillator có tần số 4 MHz, thì xung lệnh sẽ có tần số 1 MHz (chu kỳ lệnh sẽ là 1 us). Giả sử ta có một đoạn chương trình như sau:

1. MOVLW 55h
2. MOVWF PORTB
3. CALL SUB_1
4. BSF PORTA, BIT3 (forced NOP)
5. instruction @ address SUB_1

Ở đây ta chỉ bàn đến qui trình vi điều khiển xử lý đoạn chương trình trên thông qua từng chu kỳ lệnh. Quá trình trên sẽ được thực thi như sau:



Hình 1.2: Cơ chế pipelining

TCY0: đọc lệnh 1

TCY1: thực thi lệnh 1, đọc lệnh 2

TCY2: thực thi lệnh 2, đọc lệnh 3

TCY3: thực thi lệnh 3, đọc lệnh 4.

TCY4: vì lệnh 4 không phải là lệnh sẽ được thực thi theo qui trình thực thi của chương trình (lệnh tiếp theo được thực thi phải là lệnh đầu tiên tại label SUB_1) nên chu kỳ thực thi lệnh này chỉ được dùng để đọc lệnh đầu tiên tại label SUB_1. Như vậy có thể xem lệnh 3 cần 2 chu kỳ xung clock để thực thi.

TCY5: thực thi lệnh đầu tiên của SUB_1 và đọc lệnh tiếp theo của SUB_1.

Quá trình này được thực hiện tương tự cho các lệnh tiếp theo của chương trình.

Thông thường, để thực thi một lệnh, ta cần một chu kỳ lệnh để gọi lệnh đó, và một chu kỳ xung clock nữa để giải mã và thực thi lệnh. Với cơ chế pipelining được trình bày ở trên, mỗi lệnh xem như chỉ được thực thi trong một chu kỳ lệnh. Đối với các lệnh mà quá trình thực thi nó làm thay đổi giá trị thanh ghi PC (Program Counter) cần hai chu kỳ lệnh để thực thi vì phải thực hiện việc gọi lệnh ở địa chỉ thanh ghi PC chỉ tới. Sau khi đã xác định đúng vị trí lệnh trong thanh ghi PC, mỗi lệnh chỉ cần một chu kỳ lệnh để thực thi xong.

1.6 CÁC DÒNG PIC VÀ CÁCH LỰA CHỌN VI ĐIỀU KHIỂN PIC

Các kí hiệu của vi điều khiển PIC:

PIC12xxx: độ dài lệnh 12 bit

PIC16xxx: độ dài lệnh 14 bit

PIC18xxx: độ dài lệnh 16 bit

C: PIC có bộ nhớ EPROM (chỉ có 16C84 là EEPROM)

F: PIC có bộ nhớ flash

LF: PIC có bộ nhớ flash hoạt động ở điện áp thấp

LV: tương tự như LF, đây là kí hiệu cũ

Bên cạnh đó một số vi điều khiển có kí hiệu xxFxxx là EEPROM, nếu có thêm chữ A ở cuối là flash (ví dụ PIC16F877 là EEPROM, còn PIC16F877A là flash).

Ngoài ra còn có thêm một dòng vi điều khiển PIC mới là dsPIC.

Ở Việt Nam phổ biến nhất là các họ vi điều khiển PIC do hãng Microchip sản xuất.

Cách lựa chọn một vi điều khiển PIC phù hợp:

Trước hết cần chú ý đến số chân của vi điều khiển cần thiết cho ứng dụng. Có nhiều vi điều khiển PIC với số lượng chân khác nhau, thậm chí có vi điều khiển chỉ có 8 chân, ngoài ra còn có các vi điều khiển 28, 40, 44, ... chân.

Cần chọn vi điều khiển PIC có bộ nhớ flash để có thể nạp xóa chương trình được nhiều lần hơn.

Tiếp theo cần chú ý đến các khối chức năng được tích hợp sẵn trong vi điều khiển, các chuẩn giao tiếp bên trong.

Sau cùng cần chú ý đến bộ nhớ chương trình mà vi điều khiển cho phép.

Ngoài ra mọi thông tin về cách lựa chọn vi điều khiển PIC có thể được tìm thấy trong cuốn sách “Select PIC guide” do nhà sản xuất Microchip cung cấp.

1.7 NGÔN NGỮ LẬP TRÌNH CHO PIC

Ngôn ngữ lập trình cho PIC rất đa dạng. Ngôn ngữ lập trình cấp thấp có MPLAB (được cung cấp miễn phí bởi nhà sản xuất Microchip), các ngôn ngữ lập trình cấp cao hơn bao gồm C, Basic, Pascal, ... Ngoài ra còn có một số ngôn ngữ lập trình được phát triển dành riêng cho PIC như PICBasic, MikroBasic,...

1.8 MẠCH NẠP PIC

Đây cũng là một dòng sản phẩm rất đa dạng dành cho vi điều khiển PIC. Có thể sử dụng các mạch nạp được cung cấp bởi nhà sản xuất là hãng Microchip như: PICSTART plus, MPLAB ICD 2, MPLAB PM 3, PRO MATE II. Có thể dùng các sản phẩm này để nạp cho vi điều khiển khác thông qua chương trình MPLAB. Dòng sản phẩm chính thống này có ưu thế là nạp được cho tất cả các vi điều khiển PIC, tuy nhiên giá thành rất cao và thường gặp rất nhiều khó khăn trong quá trình mua sản phẩm.

Ngoài ra do tính năng cho phép nhiều chế độ nạp khác nhau, còn có rất nhiều mạch nạp được thiết kế dành cho vi điều khiển PIC. Có thể sơ lược một số mạch nạp cho PIC như sau:

JDM programmer: mạch nạp này dùng chương trình nạp Icprog cho phép nạp các vi điều khiển PIC có hỗ trợ tính năng nạp chương trình điện áp thấp ICSP (In Circuit Serial Programming). Hầu hết các mạch nạp đều hỗ trợ tính năng nạp chương trình này.

WARP-13A và MCP-USB: hai mạch nạp này giống với mạch nạp PICSTART PLUS do nhà sản xuất Microchip cung cấp, tương thích với trình biên dịch MPLAB, nghĩa là ta có thể trực tiếp dùng chương trình MPLAB để nạp cho vi điều khiển PIC mà không cần sử dụng một chương trình nạp khác, chẳng hạn như ICprog.

P16PRO40: mạch nạp này do Nigel thiết kế và cũng khá nổi tiếng. Ông còn thiết kế cả chương trình nạp, tuy nhiên ta cũng có thể sử dụng chương trình nạp Icprog.

Mạch nạp Universal của Williem: đây không phải là mạch nạp chuyên dụng dành cho PIC như P16PRO40.

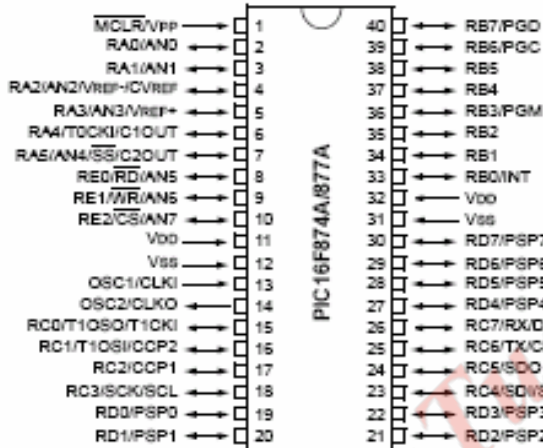
Các mạch nạp kể trên có ưu điểm rất lớn là đơn giản, rẻ tiền, hoàn toàn có thể tự lắp ráp một cách dễ dàng, và mọi thông tin về sơ đồ mạch nạp, cách thiết kế, thi công, kiểm tra và chương trình nạp đều dễ dàng tìm được và download miễn phí thông qua mạng Internet. Tuy nhiên các mạch nạp trên có nhược điểm là hạn chế về số vi điều khiển được hỗ trợ, bên cạnh đó mỗi mạch nạp cần được sử dụng với một chương trình nạp thích hợp.

1.9 BOOTLOADER VÀ ICP (In Circuit Programming)

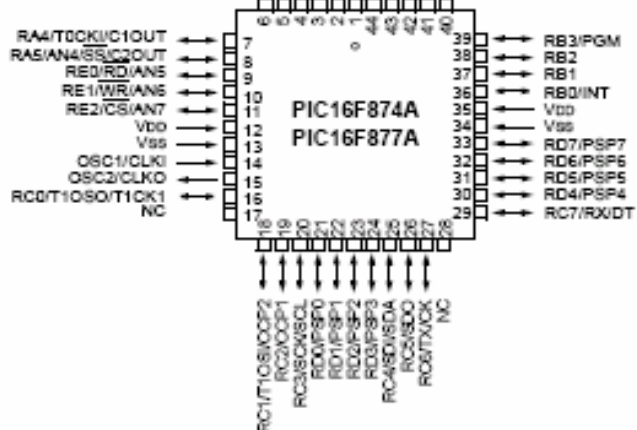
CHƯƠNG 2 VI ĐIỀU KHIỂN PIC16F877A

2.1 SƠ ĐỒ CHÂN VI ĐIỀU KHIỂN PIC16F877A

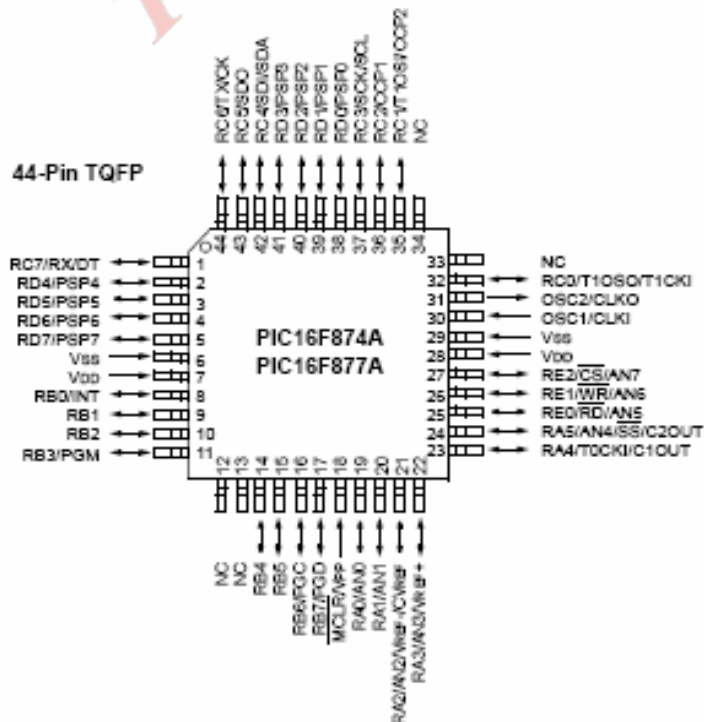
40-Pin PDIP



44-Pin PLCC



44-Pin TQFP



Hình 2.1 Vi điều khiển PIC16F877A/PIC16F874A và các dạng sơ đồ chân

2.2 MỘT VÀI THÔNG SỐ VỀ VI ĐIỀU KHIỂN PIC16F877A

Đây là vi điều khiển thuộc họ PIC16Fxxx với tập lệnh gồm 35 lệnh có độ dài 14 bit. Mỗi lệnh đều được thực thi trong một chu kỳ xung clock. Tốc độ hoạt động tối đa cho phép là 20 MHz với một chu kỳ lệnh là 200ns. Bộ nhớ chương trình 8Kx14 bit, bộ nhớ dữ liệu 368x8 byte RAM và bộ nhớ dữ liệu EEPROM với dung lượng 256x8 byte. Số PORT I/O là 5 với 33 pin I/O.

Các đặc tính ngoại vi bao gồm các khối chức năng sau:

Timer0: bộ đếm 8 bit với bộ chia tần số 8 bit.

Timer1: bộ đếm 16 bit với bộ chia tần số, có thể thực hiện chức năng đếm dựa vào xung clock ngoại vi ngay khi vi điều khiển hoạt động ở chế độ sleep.

Timer2: bộ đếm 8 bit với bộ chia tần số, bộ postcaler.

Hai bộ Capture/so sánh/điều chế độ rộng xung.

Các chuẩn giao tiếp nối tiếp SSP (Synchronous Serial Port), SPI và I2C.

Chuẩn giao tiếp nối tiếp USART với 9 bit địa chỉ.

Cổng giao tiếp song song PSP (Parallel Slave Port) với các chân điều khiển RD, WR, CS ở bên ngoài.

Các đặc tính Analog:

8 kênh chuyển đổi ADC 10 bit.

Hai bộ so sánh.

Bên cạnh đó là một vài đặc tính khác của vi điều khiển như:

Bộ nhớ flash với khả năng ghi xóa được 100.000 lần.

Bộ nhớ EEPROM với khả năng ghi xóa được 1.000.000 lần.

Dữ liệu bộ nhớ EEPROM có thể lưu trữ trên 40 năm.

Khả năng tự nạp chương trình với sự điều khiển của phần mềm.

Nạp được chương trình ngay trên mạch điện ICSP (In Circuit Serial Programming) thông qua 2 chân.

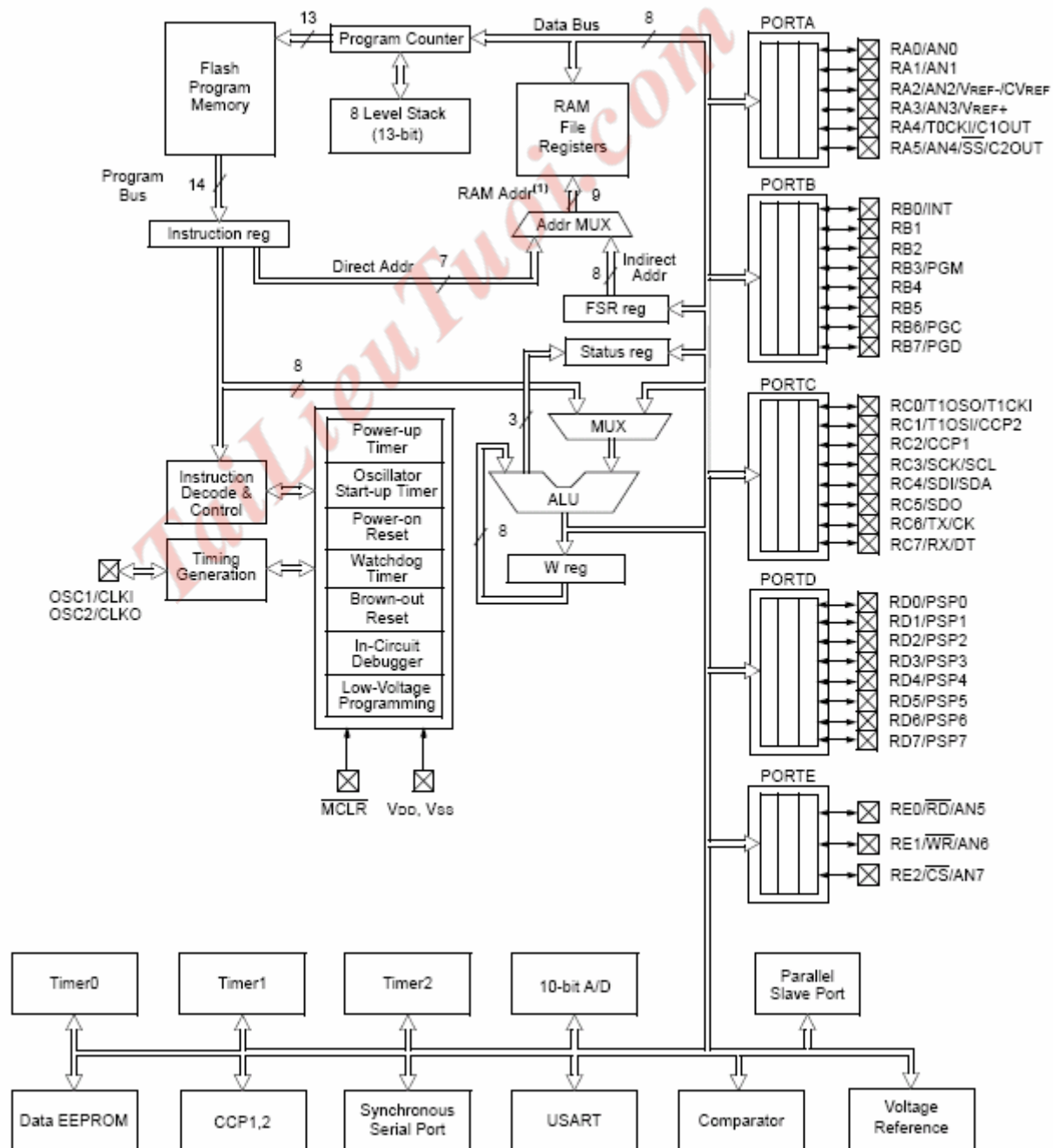
Watchdog Timer với bộ dao động trong.

Chức năng bảo mật mã chương trình.

Chế độ Sleep.

Có thể hoạt động với nhiều dạng Oscillator khác nhau.

2.3 SƠ ĐỒ KHỐI VI ĐIỀU KHIỂN PIC16F877A



Hình 2.2 Sơ đồ khối vi điều khiển PIC16F877A.

2.4 TỔ CHỨC BỘ NHỚ

Cấu trúc bộ nhớ của vi điều khiển PIC16F877A bao gồm bộ nhớ chương trình (Program memory) và bộ nhớ dữ liệu (Data Memory).

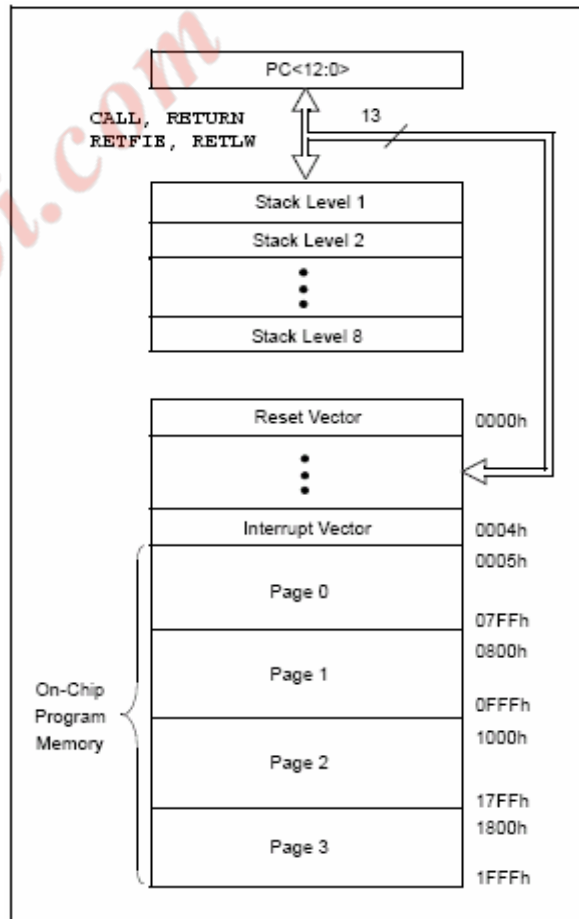
2.4.1 BỘ NHỚ CHƯƠNG TRÌNH

Bộ nhớ chương trình của vi điều khiển PIC16F877A là bộ nhớ flash, dung lượng bộ nhớ 8K word (1 word = 14 bit) và được phân thành nhiều trang (từ page0 đến page 3). Như vậy bộ nhớ chương trình có khả năng chứa được $8 \times 1024 = 8192$ lệnh (vì một lệnh sau khi mã hóa sẽ có dung lượng 1 word (14 bit)).

Để mã hóa được địa chỉ của 8K word bộ nhớ chương trình, bộ đếm chương trình có dung lượng 13 bit (PC<12:0>).

Khi vi điều khiển được reset, bộ đếm chương trình sẽ chỉ đến địa chỉ 0000h (Reset vector). Khi có ngắt xảy ra, bộ đếm chương trình sẽ chỉ đến địa chỉ 0004h (Interrupt vector).

Bộ nhớ chương trình không bao gồm bộ nhớ stack và không được địa chỉ hóa bởi bộ đếm chương trình. Bộ nhớ stack sẽ được đề cập cụ thể trong phần sau.



Hình 2.3 Bộ nhớ chương trình PIC16F877A

2.4.2 BỘ NHỚ DỮ LIỆU

Bộ nhớ dữ liệu của PIC là bộ nhớ EEPROM được chia ra làm nhiều bank. Đối với PIC16F877A bộ nhớ dữ liệu được chia ra làm 4 bank. Mỗi bank có dung lượng 128 byte, bao gồm các thanh ghi có chức năng đặc biệt SFG (Special Function Register) nằm ở các vùng địa chỉ thấp và các thanh ghi mục đích chung GPR (General Purpose Register) nằm ở vùng địa chỉ còn lại trong bank. Các thanh ghi SFR thường xuyên được sử dụng (ví dụ như thanh ghi STATUS) sẽ được đặt ở tất cả các bank của bộ nhớ dữ liệu giúp thuận tiện trong quá trình truy xuất và làm giảm bớt lệnh của chương trình. Sơ đồ cụ thể của bộ nhớ dữ liệu PIC16F877A như sau:

File Address	File Address	File Address	File Address
Indirect addr. ^(*) 00h	Indirect addr. ^(*) 80h	Indirect addr. ^(*) 100h	Indirect addr. ^(*) 180h
TMR0 01h	OPTION_REG 81h	TMR0 101h	OPTION_REG 181h
PCL 02h	PCL 82h	PCL 102h	PCL 182h
STATUS 03h	STATUS 83h	STATUS 103h	STATUS 183h
FSR 04h	FSR 84h	FSR 104h	FSR 184h
PORTA 05h	TRISA 85h		
PORTB 06h	TRISB 86h	PORTB 106h	TRISB 186h
PORTC 07h	TRISC 87h		
PORTD ⁽¹⁾ 08h	TRISD ⁽¹⁾ 88h		
PORTE ⁽¹⁾ 09h	TRISE ⁽¹⁾ 89h		
PCLATH 0Ah	PCLATH 8Ah	PCLATH 10Ah	PCLATH 18Ah
INTCON 0Bh	INTCON 8Bh	INTCON 10Bh	INTCON 18Bh
PIR1 0Ch	PIE1 8Ch	EEDATA 10Ch	EECON1 18Ch
PIR2 0Dh	PIE2 8Dh	EEADR 10Dh	EECON2 18Dh
TMR1L 0Eh	PCON 8Eh	EEDATH 10Eh	Reserved ⁽²⁾ 18Eh
TMR1H 0Fh		EEADRH 10Fh	Reserved ⁽²⁾ 18Fh
T1CON 10h			
TMR2 11h	SSPCON2 91h		
T2CON 12h	PR2 92h		
SSPBUF 13h	SSPADDD 93h		
SSPCON 14h	SSPSTAT 94h		
CCPR1L 15h			
CCPR1H 16h			
CCP1CON 17h			
RCSTA 18h	TXSTA 98h	General Purpose Register 16 Bytes	General Purpose Register 16 Bytes
TXREG 19h	SPBRG 99h		
RCREG 1Ah			
CCPR2L 1Bh			
CCPR2H 1Ch	CMCON 9Ch		
CCP2CON 1Dh	CVRCON 9Dh		
ADRESH 1Eh	ADRESL 9Eh		
ADCON0 1Fh	ADCON1 9Fh		
General Purpose Register 96 Bytes	General Purpose Register 80 Bytes	General Purpose Register 80 Bytes	General Purpose Register 80 Bytes
	accesses 70h-7Fh	accesses 70h-7Fh	accesses 70h - 7Fh
Bank 0 7Fh	Bank 1 FFh	Bank 2 17Fh	Bank 3 1FFh

■ Unimplemented data memory locations, read as '0'.
 * Not a physical register.

Note 1: These registers are not implemented on the PIC16F876A.
Note 2: These registers are reserved; maintain these registers clear.

Hình 2.4 Sơ đồ bộ nhớ dữ liệu PIC16F877A

2.4.2.1 THANH GHI CHỨC NĂNG ĐẶC BIỆT SFR

Đây là các thanh ghi được sử dụng bởi CPU hoặc được dùng để thiết lập và điều khiển các khối chức năng được tích hợp bên trong vi điều khiển. Có thể phân thanh ghi SFR làm hai loại: thanh ghi SFR liên quan đến các chức năng bên trong (CPU) và thanh ghi SFR dùng để thiết lập và điều khiển các khối chức năng bên ngoài (ví dụ như ADC, PWM, ...). Phần này sẽ đề cập đến các thanh ghi liên quan đến các chức năng bên trong. Các thanh ghi dùng để thiết lập và điều khiển các khối chức năng sẽ được nhắc đến khi ta đề cập đến các khối chức năng đó. Chi tiết về các thanh ghi SFR sẽ được liệt kê cụ thể trong bảng phụ lục 2.

Thanh ghi STATUS (03h, 83h, 103h, 183h): thanh ghi chứa kết quả thực hiện phép toán của khối ALU, trạng thái reset và các bit chọn bank cần truy xuất trong bộ nhớ dữ liệu.

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C
bit 7							bit 0

Thanh ghi OPTION_REG (81h, 181h): thanh ghi này cho phép đọc và ghi, cho phép điều khiển chức năng pull-up của các chân trong PORTB, xác lập các tham số về xung tác động, cạnh tác động của ngắt ngoại vi và bộ đếm Timer0.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBP1	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Thanh ghi INTCON (0Bh, 8Bh, 10Bh, 18Bh): thanh ghi cho phép đọc và ghi, chứa các bit điều khiển và các bit cờ hiệu khi timer0 bị tràn, ngắt ngoại vi RB0/INT và ngắt interrupt-on-change tại các chân của PORTB.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

Thanh ghi PIE1 (8Ch): chứa các bit điều khiển chi tiết các ngắt của các khối chức năng ngoại vi.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Thanh ghi PIR1 (0Ch) chứa cờ ngắt của các khối chức năng ngoại vi, các ngắt này được cho phép bởi các bit điều khiển chứa trong thanh ghi PIE1.

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Thanh ghi PIE2 (8Dh): chứa các bit điều khiển các ngắt của các khối chức năng CCP2, SSP bus, ngắt của bộ so sánh và ngắt ghi vào bộ nhớ EEPROM.

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

Thanh ghi PIR2 (0Dh): chứa các cờ ngắt của các khối chức năng ngoại vi, các ngắt này được cho phép bởi các bit điều khiển chứa trong thanh ghi PIE2.

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

Thanh ghi PCON (8Eh): chứa các cờ hiệu cho biết trạng thái các chế độ reset của vi điều khiển.

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-1
—	—	—	—	—	—	POR	BOR
bit 7							bit 0

2.4.2.2 THANH GHI MỤC ĐÍCH CHUNG GPR

Các thanh ghi này có thể được truy xuất trực tiếp hoặc gián tiếp thông qua thanh ghi FSG (File Select Register). Đây là các thanh ghi dữ liệu thông thường, người sử dụng có thể tùy theo mục đích chương trình mà có thể dùng các thanh ghi này để chứa các biến số, hằng số, kết quả hoặc các tham số phục vụ cho chương trình.

2.4.3 STACK

Stack không nằm trong bộ nhớ chương trình hay bộ nhớ dữ liệu mà là một vùng nhớ đặc biệt không cho phép đọc hay ghi. Khi lệnh CALL được thực hiện hay khi một ngắt xảy ra làm chương trình bị rẽ nhánh, giá trị của bộ đếm chương trình PC tự động được vi điều khiển cất vào trong stack. Khi một trong các lệnh RETURN, RETLW hay RETFIE được thực thi, giá trị PC sẽ tự động được lấy ra từ trong stack, vi điều khiển sẽ thực hiện tiếp chương trình theo đúng qui trình định trước.

Bộ nhớ Stack trong vi điều khiển PIC họ 16F87xA có khả năng chứa được 8 địa chỉ và hoạt động theo cơ chế xoay vòng. Nghĩa là giá trị cất vào bộ nhớ Stack lần thứ 9 sẽ ghi đè lên giá trị cất vào Stack lần đầu tiên và giá trị cất vào bộ nhớ Stack lần thứ 10 sẽ ghi đè lên giá trị cất vào Stack lần thứ 2.

Cần chú ý là không có cờ hiệu nào cho biết trạng thái stack, do đó ta không biết được khi nào stack tràn. Bên cạnh đó tập lệnh của vi điều khiển dòng PIC cũng không có lệnh POP hay PUSH, các thao tác với bộ nhớ stack sẽ hoàn toàn được điều khiển bởi CPU.

2.5 CÁC CỔNG XUẤT NHẬP CỦA PIC16F877A

Cổng xuất nhập (I/O port) chính là phương tiện mà vi điều khiển dùng để tương tác với thế giới bên ngoài. Sự tương tác này rất đa dạng và thông qua quá trình tương tác đó, chức năng của vi điều khiển được thể hiện một cách rõ ràng.

Một cổng xuất nhập của vi điều khiển bao gồm nhiều chân (I/O pin), tùy theo cách bố trí và chức năng của vi điều khiển mà số lượng cổng xuất nhập và số lượng chân trong mỗi cổng có thể khác nhau. Bên cạnh đó, do vi điều khiển được tích hợp sẵn bên trong các đặc tính giao tiếp ngoại vi nên bên cạnh chức năng là cổng xuất nhập thông thường, một số chân xuất nhập còn có thêm các chức năng khác để thể hiện sự tác động của các đặc tính ngoại vi nêu trên đối với thế giới bên ngoài. Chức năng của từng chân xuất nhập trong mỗi cổng hoàn toàn có thể được xác lập và điều khiển được thông qua các thanh ghi SFR liên quan đến chân xuất nhập đó.

Vi điều khiển PIC16F877A có 5 cổng xuất nhập, bao gồm PORTA, PORTB, PORTC, PORTD và PORTE. Cấu trúc và chức năng của từng cổng xuất nhập sẽ được đề cập cụ thể trong phần sau.

2.5.1 PORTA

PORTA (RPA) bao gồm 6 I/O pin. Đây là các chân “hai chiều” (bidirectional pin), nghĩa là có thể xuất và nhập được. Chức năng I/O này được điều khiển bởi thanh ghi TRISA (địa chỉ 85h). Muốn xác lập chức năng của một chân trong PORTA là input, ta “set” bit điều khiển tương ứng với chân đó trong thanh ghi TRISA và ngược lại, muốn xác lập chức năng của một chân trong PORTA là output, ta “clear” bit điều khiển tương ứng với chân đó trong thanh ghi TRISA. Thao tác này hoàn toàn tương tự đối với các PORT và các thanh ghi điều khiển tương ứng TRIS (đối với PORTA là TRISA, đối với PORTB là TRISB, đối với PORTC là TRISC, đối với PORTD là TRISD và đối với PORTE là TRISE). Bên cạnh đó PORTA còn là ngõ ra của bộ ADC, bộ so sánh, ngõ vào analog ngõ vào xung clock của Timer0 và ngõ vào của bộ giao tiếp MSSP (Master Synchronous Serial Port). Đặc tính này sẽ được trình bày cụ thể trong phần sau.

Cấu trúc bên trong và chức năng cụ thể của từng chân trong PORTA sẽ được trình bày cụ thể trong Phụ lục 1.

Các thanh ghi SFR liên quan đến PORTA bao gồm:

PORTA (địa chỉ 05h)	: chứa giá trị các pin trong PORTA.
TRISA (địa chỉ 85h)	: điều khiển xuất nhập.
CMCON (địa chỉ 9Ch)	: thanh ghi điều khiển bộ so sánh.
CVRCON (địa chỉ 9Dh)	: thanh ghi điều khiển bộ so sánh điện áp.

ADCON1 (địa chỉ 9Fh) : thanh ghi điều khiển bộ ADC.
Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.5.2 PORTB

PORTB (RPB) gồm 8 pin I/O. Thanh ghi điều khiển xuất nhập tương ứng là TRISB. Bên cạnh đó một số chân của PORTB còn được sử dụng trong quá trình nạp chương trình cho vi điều khiển với các chế độ nạp khác nhau. PORTB còn liên quan đến ngắt ngoại vi và bộ Timer0. PORTB còn được tích hợp chức năng điện trở kéo lên được điều khiển bởi chương trình.

Cấu trúc bên trong và chức năng cụ thể của từng chân trong PORTB sẽ được trình bày cụ thể trong Phụ lục 1.

Các thanh ghi SFR liên quan đến PORTB bao gồm:

- PORTB (địa chỉ 06h,106h) : chứa giá trị các pin trong PORTB
- TRISB (địa chỉ 86h,186h) : điều khiển xuất nhập
- OPTION_REG (địa chỉ 81h,181h) : điều khiển ngắt ngoại vi và bộ Timer0.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.5.3 PORTC

PORTC (RPC) gồm 8 pin I/O. Thanh ghi điều khiển xuất nhập tương ứng là TRISC. Bên cạnh đó PORTC còn chứa các chân chức năng của bộ so sánh, bộ Timer1, bộ PWM và các chuẩn giao tiếp nối tiếp I2C, SPI, SSP, USART.

Cấu trúc bên trong và chức năng cụ thể của từng chân trong PORTC sẽ được trình bày cụ thể trong Phụ lục 1.

Các thanh ghi điều khiển liên quan đến PORTC:

- PORTC (địa chỉ 07h) : chứa giá trị các pin trong PORTC
- TRISC (địa chỉ 87h) : điều khiển xuất nhập.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.5.4 PORTD

PORTD (RPD) gồm 8 chân I/O, thanh ghi điều khiển xuất nhập tương ứng là TRISD. PORTD còn là cổng xuất dữ liệu của chuẩn giao tiếp PSP (Parallel Slave Port).

Cấu trúc bên trong và chức năng cụ thể của từng chân trong PORTD sẽ được trình bày cụ thể trong Phụ lục 1.

Các thanh ghi liên quan đến PORTD bao gồm:

- Thanh ghi PORTD : chứa giá trị các pin trong PORTD.
- Thanh ghi TRISD : điều khiển xuất nhập.
- Thanh ghi TRISE : điều khiển xuất nhập PORTE và chuẩn giao tiếp PSP.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.5.5 PORTE

PORTE (RPE) gồm 3 chân I/O. Thanh ghi điều khiển xuất nhập tương ứng là TRISE. Các chân của PORTE có ngõ vào analog. Bên cạnh đó PORTE còn là các chân điều khiển của chuẩn giao tiếp PSP.

Cấu trúc bên trong và chức năng cụ thể của từng chân trong PORTE sẽ được trình bày cụ thể trong Phụ lục 1.

Các thanh ghi liên quan đến PORTE bao gồm:

PORTE : chứa giá trị các chân trong PORTE.

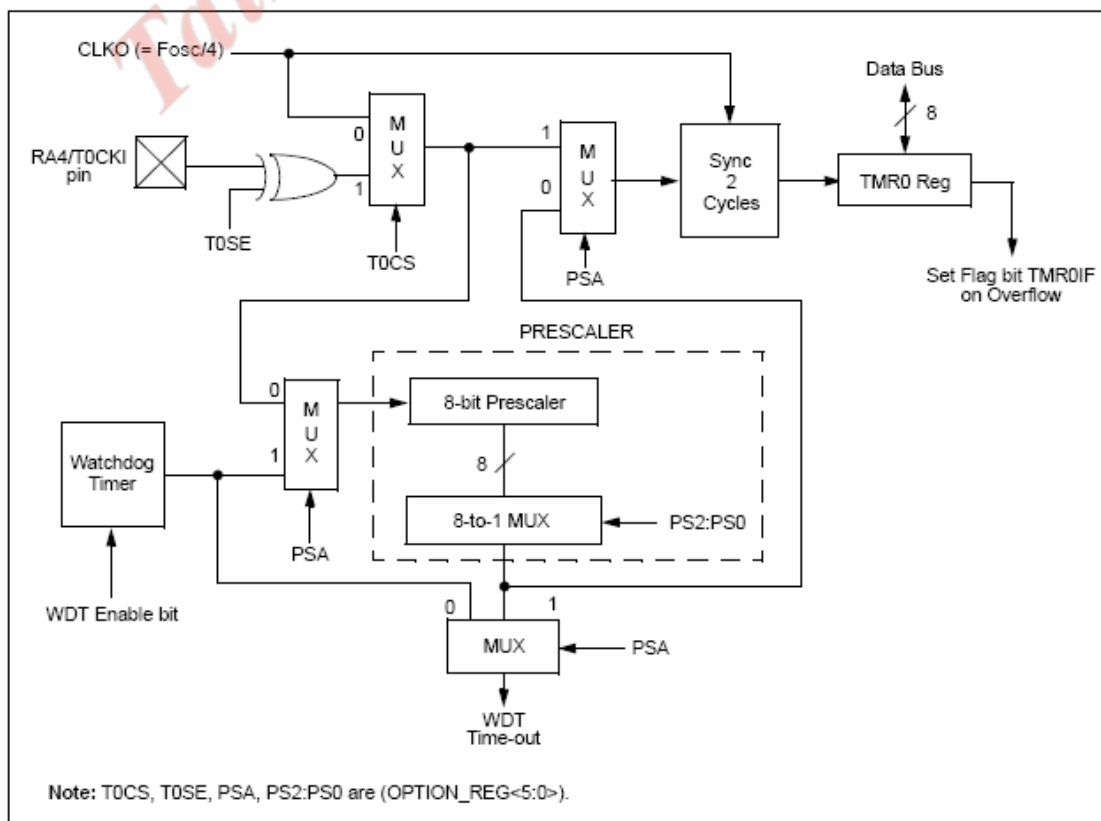
TRISE : điều khiển xuất nhập và xác lập các thông số cho chuẩn giao tiếp PSP.

ADCON1 : thanh ghi điều khiển khối ADC.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.6 TIMER 0

Đây là một trong ba bộ đếm hoặc bộ định thời của vi điều khiển PIC16F877A. Timer0 là bộ đếm 8 bit được kết nối với bộ chia tần số (prescaler) 8 bit. Cấu trúc của Timer0 cho phép ta lựa chọn xung clock tác động và cạnh tích cực của xung clock. Ngắt Timer0 sẽ xuất hiện khi Timer0 bị tràn. Bit TMR0IE (INTCON<5>) là bit điều khiển của Timer0. TMR0IE=1 cho phép ngắt Timer0 tác động, TMR0IF= 0 không cho phép ngắt Timer0 tác động. Sơ đồ khối của Timer0 như sau:



Hình 2.5 Sơ đồ khối của Timer0.

Muốn Timer0 hoạt động ở chế độ Timer ta clear bit TOSC (OPTION_REG<5>), khi đó giá trị thanh ghi TMR0 sẽ tăng theo từng chu kỳ xung đồng hồ (tần số vào Timer0 bằng ¼ tần số oscillator). Khi giá trị thanh ghi TMR0 từ FFh trở về 00h, ngắt Timer0 sẽ xuất hiện. Thanh ghi TMR0 cho phép ghi và xóa được giúp ta ấn định thời điểm ngắt Timer0 xuất hiện một cách linh động.

Muốn Timer0 hoạt động ở chế độ counter ta set bit TOSC (OPTION_REG<5>). Khi đó xung tác động lên bộ đếm được lấy từ chân RA4/TOCK1. Bit TOSE (OPTION_REG<4>) cho phép lựa chọn cạnh tác động vào bộ đếm. Cạnh tác động sẽ là cạnh lên nếu TOSE=0 và cạnh tác động sẽ là cạnh xuống nếu TOSE=1.

Khi thanh ghi TMR0 bị tràn, bit TMR0IF (INTCON<2>) sẽ được set. Đây chính là cờ ngắt của Timer0. Cờ ngắt này phải được xóa bằng chương trình trước khi bộ đếm bắt đầu thực hiện lại quá trình đếm. Ngắt Timer0 không thể “đánh thức” vi điều khiển từ chế độ sleep.

Bộ chia tần số (prescaler) được chia sẻ giữa Timer0 và WDT (Watchdog Timer). Điều đó có nghĩa là nếu prescaler được sử dụng cho Timer0 thì WDT sẽ không có được hỗ trợ của prescaler và ngược lại. Prescaler được điều khiển bởi thanh ghi OPTION_REG. Bit PSA (OPTION_REG<3>) xác định đối tượng tác động của prescaler. Các bit PS2:PS0 (OPTION_REG<2:0>) xác định tỉ số chia tần số của prescaler. Xem lại thanh ghi OPTION_REG để xác định lại một cách chi tiết về các bit điều khiển trên.

Các lệnh tác động lên giá trị thanh ghi TMR0 sẽ xóa chế độ hoạt động của prescaler. Khi đối tượng tác động là Timer0, tác động lên giá trị thanh ghi TMR0 sẽ xóa prescaler nhưng không làm thay đổi đối tượng tác động của prescaler. Khi đối tượng tác động là WDT, lệnh CLRWDWT sẽ xóa prescaler, đồng thời prescaler sẽ ngưng tác vụ hỗ trợ cho WDT.

Các thanh ghi điều khiển liên quan đến Timer0 bao gồm:

TMR0 (địa chỉ 01h, 101h) : chứa giá trị đếm của Timer0.

INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): cho phép ngắt hoạt động (GIE và PEIE).

OPTION_REG (địa chỉ 81h, 181h): điều khiển prescaler.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.7 TIMER1

Timer1 là bộ định thời 16 bit, giá trị của Timer1 sẽ được lưu trong hai thanh ghi (TMR1H:TMR1L). Cờ ngắt của Timer1 là bit TMR1IF (PIR1<0>). Bit điều khiển của Timer1 sẽ là TMR1IE (PIE<0>).

Tương tự như Timer0, Timer1 cũng có hai chế độ hoạt động: chế độ định thời (timer) với xung kích là xung clock của oscillator (tần số của timer bằng ¼ tần số của oscillator) và chế độ đếm (counter) với xung kích là xung phản ánh các sự kiện cần đếm lấy từ bên ngoài

[illegible]

Ngoài ra Timer1 còn có chức năng reset input bên trong được điều khiển bởi một trong hai khối CCP (Capture/Compare/PWM).

Timer1 có hai chế độ đếm là đồng bộ (Synchronous) và bất đồng bộ (Asynchronous). Chế độ đếm được quyết định bởi bit điều khiển $\overline{T1SYNC}$ (T1CON<2>).

Khi $\overline{T1SYNC}=0$ xung đếm vào Timer1 sẽ được đồng bộ hóa với xung clock bên trong. Ở chế độ này Timer1 sẽ không hoạt động khi vi điều khiển đang ở chế độ sleep.

INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): cho phép ngắt hoạt động (GIE và PEIE).

PIR1 (địa chỉ 0Ch): chứa cờ ngắt Timer1 (TMR1IF).

PIE1(địa chỉ 8Ch): cho phép ngắt Timer1 (TMR1IE).

TMR1L (địa chỉ 0Eh): chứa giá trị 8 bit thấp của bộ đếm Timer1.

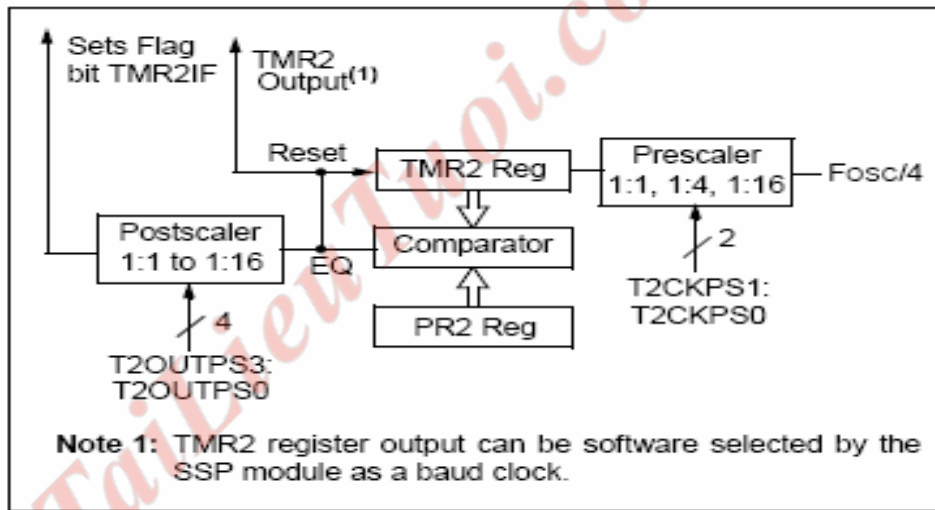
TMR1H (địa chỉ 0Eh): chứa giá trị 8 bit cao của bộ đếm Timer1.

T1CON (địa chỉ 10h): xác lập các thông số cho Timer1.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.8 TIMER2

Timer2 là bộ định thời 8 bit và được hỗ trợ bởi hai bộ chia tần số prescaler và postscaler. Thanh ghi chứa giá trị đếm của Timer2 là TMR2. Bit cho phép ngắt Timer2 tác động là TMR2ON (T2CON<2>). Cờ ngắt của Timer2 là bit TMR2IF (PIR1<1>). Xung ngõ vào (tần số bằng $\frac{1}{4}$ tần số oscillator) được đưa qua bộ chia tần số prescaler 4 bit (với các tỉ số chia tần số là 1:1, 1:4 hoặc 1:16 và được điều khiển bởi các bit T2CKPS1:T2CKPS0 (T2CON<1:0>)).



Hình 2.7 Sơ đồ khối Timer2.

Timer2 còn được hỗ trợ bởi thanh ghi PR2. Giá trị đếm trong thanh ghi TMR2 sẽ tăng từ 00h đến giá trị chứa trong thanh ghi PR2, sau đó được reset về 00h. Khi reset thanh ghi PR2 được nhận giá trị mặc định FFh.

Ngõ ra của Timer2 được đưa qua bộ chia tần số postscaler với các mức chia từ 1:1 đến 1:16. Postscaler được điều khiển bởi 4 bit T2OUTPS3:T2OUTPS0. Ngõ ra của postscaler đóng vai trò quyết định trong việc điều khiển cờ ngắt.

Ngoài ra ngõ ra của Timer2 còn được kết nối với khối SSP, do đó Timer2 còn đóng vai trò tạo ra xung clock đồng bộ cho khối giao tiếp SSP.

Các thanh ghi liên quan đến Timer2 bao gồm:

- INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): cho phép toàn bộ các ngắt (GIE và PEIE).
- PIR1 (địa chỉ 0Ch): chứa cờ ngắt Timer2 (TMR2IF).
- PIE1 (địa chỉ 8Ch): chứa bit điều khiển Timer2 (TMR2IE).
- TMR2 (địa chỉ 11h): chứa giá trị đếm của Timer2.
- T2CON (địa chỉ 12h): xác lập các thông số cho Timer2.

PR2 (địa chỉ 92h): thanh ghi hỗ trợ cho Timer2.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

Ta có một vài nhận xét về Timer0, Timer1 và Timer2 như sau:

Timer0 và Timer2 là bộ đếm 8 bit (giá trị đếm tối đa là FFh), trong khi Timer1 là bộ đếm 16 bit (giá trị đếm tối đa là FFFFh).

Timer0, Timer1 và Timer2 đều có hai chế độ hoạt động là timer và counter. Xung clock có tần số bằng $\frac{1}{4}$ tần số của oscillator.

Xung tác động lên Timer0 được hỗ trợ bởi prescaler và có thể được thiết lập ở nhiều chế độ khác nhau (tần số tác động, cạnh tác động) trong khi các thông số của xung tác động lên Timer1 là cố định. Timer2 được hỗ trợ bởi hai bộ chia tần số prescaler và postcaler độc lập, tuy nhiên cạnh tác động vẫn được cố định là cạnh lên.

Timer1 có quan hệ với khối CCP, trong khi Timer2 được kết nối với khối SSP.

Một vài so sánh sẽ giúp ta dễ dàng lựa chọn được Timer thích hợp cho ứng dụng.

2.9 ADC

ADC (Analog to Digital Converter) là bộ chuyển đổi tín hiệu giữa hai dạng tương tự và số. PIC16F877A có 8 ngõ vào analog (RA4:RA0 và RE2:RE0). Hiệu điện thế chuẩn V_{REF} có thể được lựa chọn là V_{DD} , V_{SS} hay hiệu điện thế chuẩn được xác lập trên hai chân RA2 và RA3. Kết quả chuyển đổi từ tín hiệu tương tự sang tín hiệu số là 10 bit số tương ứng và được lưu trong hai thanh ghi ADRESH:ADRESL. Khi không sử dụng bộ chuyển đổi ADC, các thanh ghi này có thể được sử dụng như các thanh ghi thông thường khác. Khi quá trình chuyển đổi hoàn tất, kết quả sẽ được lưu vào hai thanh ghi ADRESH:ADRESL, bit $\overline{GO/DONE}$ (ADCON0<2>) được xóa về 0 và cờ ngắt ADIF được set.

Qui trình chuyển đổi từ tương tự sang số bao gồm các bước sau:

1. Thiết lập các thông số cho bộ chuyển đổi ADC:

Chọn ngõ vào analog, chọn điện áp mẫu (dựa trên các thông số của thanh ghi ADCON1)

Chọn kênh chuyển đổi AD (thanh ghi ADCON0).

Chọn xung clock cho kênh chuyển đổi AD (thanh ghi ADCON0).

Cho phép bộ chuyển đổi AD hoạt động (thanh ghi ADCON0).

2. Thiết lập các cờ ngắt cho bộ AD

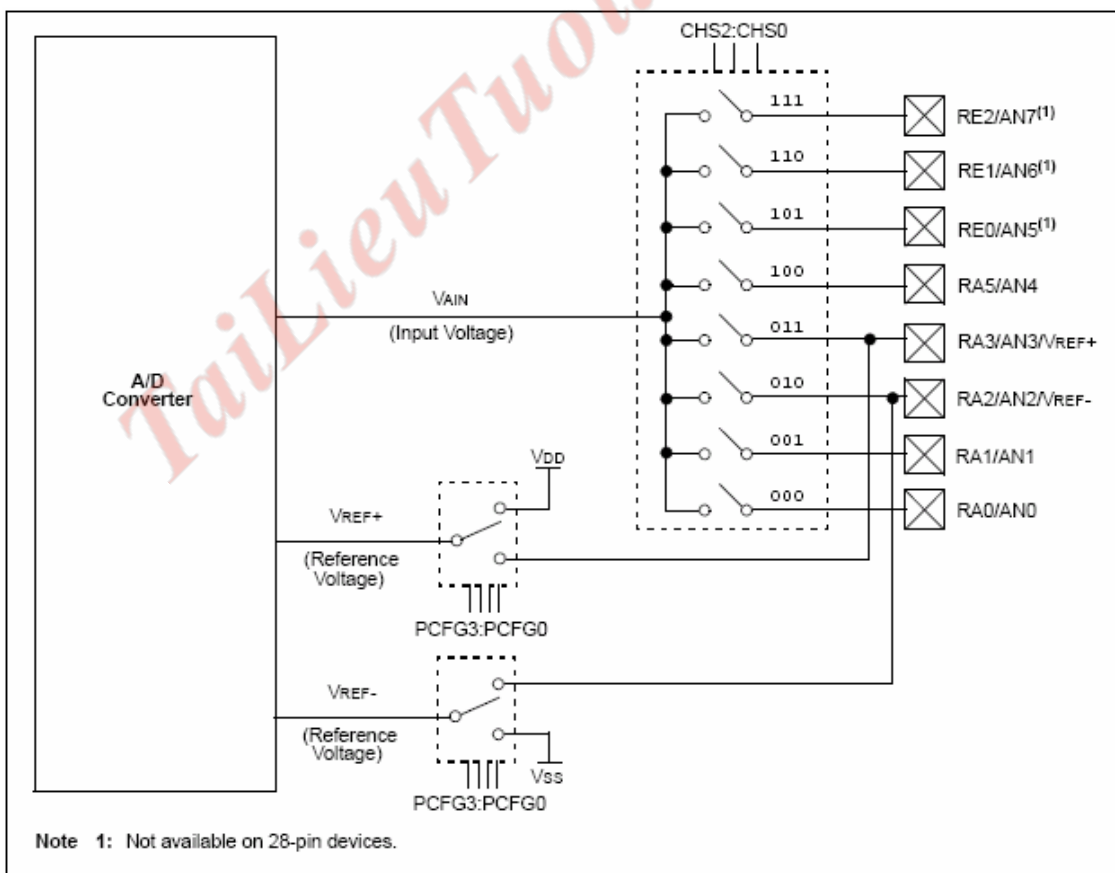
Clear bit ADIF.

Set bit ADIE.

Set bit PEIE.

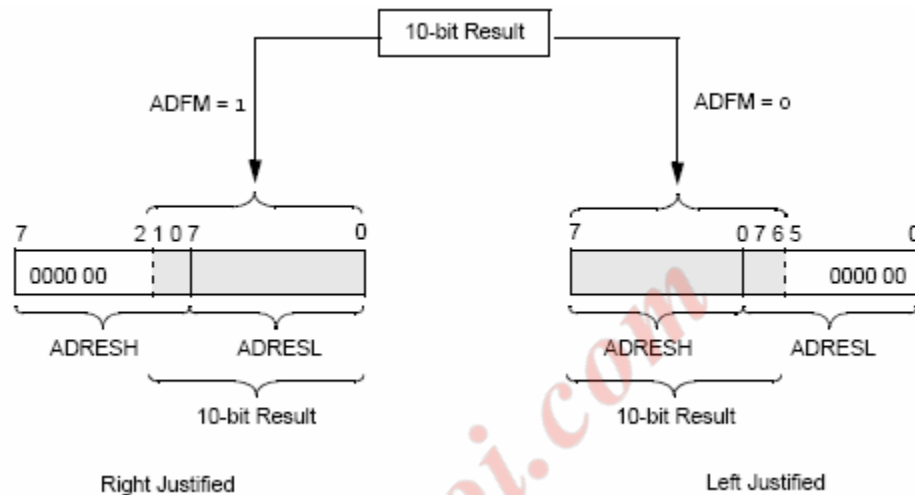
Set bit GIE.

3. Đợi cho tới khi quá trình lấy mẫu hoàn tất.
4. Bắt đầu quá trình chuyển đổi (set bit $\overline{GO/DONE}$).
5. Đợi cho tới khi quá trình chuyển đổi hoàn tất bằng cách:
Kiểm tra bit $\overline{GO/DONE}$. Nếu $\overline{GO/DONE} = 0$, quá trình chuyển đổi đã hoàn tất.
Kiểm tra cờ ngắt.
6. Đọc kết quả chuyển đổi và xóa cờ ngắt, set bit $\overline{GO/DONE}$ (nếu cần tiếp tục chuyển đổi).
7. Tiếp tục thực hiện các bước 1 và 2 cho quá trình chuyển đổi tiếp theo.



Hình 2.8 Sơ đồ khối bộ chuyển đổi ADC.

Cần chú ý là có hai cách lưu kết quả chuyển đổi AD, việc lựa chọn cách lưu được điều khiển bởi bit ADFM và được minh họa cụ thể trong hình sau:



Hình 2.9 Các cách lưu kết quả chuyển đổi AD.

Các thanh ghi liên quan đến bộ chuyển đổi ADC bao gồm:

INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): cho phép các ngắt (các bit GIE, PEIE).

PIR1 (địa chỉ 0Ch): chứa cờ ngắt AD (bit ADIF).

PIE1 (địa chỉ 8Ch): chứa bit điều khiển AD (ADIE).

ADRESH (địa chỉ 1Eh) và ADRESL (địa chỉ 9Eh): các thanh ghi chứa kết quả chuyển đổi AD.

ADCON0 (địa chỉ 1Fh) và ADCON1 (địa chỉ 9Fh): xác lập các thông số cho bộ chuyển đổi AD.

PORTA (địa chỉ 05h) và TRISA (địa chỉ 85h): liên quan đến các ngõ vào analog ở PORTA.

PORTE (địa chỉ 09h) và TRISE (địa chỉ 89h): liên quan đến các ngõ vào analog ở PORTE.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.10 COMPARATOR

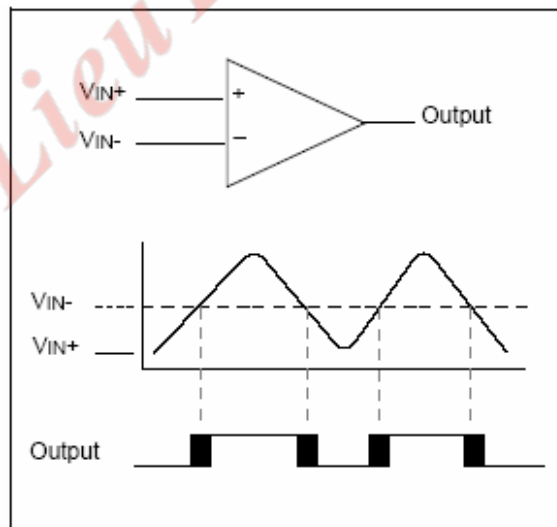
Bộ so sánh bao gồm hai bộ so sánh tín hiệu analog và được đặt ở PORTA. Ngõ vào bộ so sánh là các chân RA3:RA0, ngõ ra là hai chân RA4 và RA5. Thanh ghi điều khiển bộ so sánh là CMCON. Các bit CM2:CM0 trong thanh ghi CMCON đóng vai trò chọn lựa các chế độ hoạt động cho bộ Comparator (hình 2.10).

Cơ chế hoạt động của bộ Comparator như sau:

Tín hiệu analog ở chân V_{IN+} sẽ được so sánh với điện áp chuẩn ở chân V_{IN-} và tín hiệu ở ngõ ra bộ so sánh sẽ thay đổi tương ứng như hình vẽ. Khi điện áp ở chân V_{IN+} lớn hơn điện áp ở chân V_{IN-} ngõ ra sẽ ở mức 1 và ngược lại.

Dựa vào hình vẽ ta thấy đáp ứng tại ngõ ra không phải là tức thời so với thay đổi tại ngõ vào mà cần có một khoảng thời gian nhất định để ngõ ra thay đổi trạng thái (tối đa là 10 us). Cần chú ý đến khoảng thời gian đáp ứng này khi sử dụng bộ so sánh.

Cực tính của các bộ so sánh có thể thay đổi dựa vào các giá trị đặt vào các bit C2INV và C1INV (CMCON<4:5>).



Hình 2.10 Nguyên lý hoạt động của một bộ so sánh đơn giản.

ghì PIR2 (địa chỉ 0Dh): chứa cờ ngắt của bộ so
ghì PIE2 (địa chỉ 8Dh): chứa bit cho phép bộ so
ghì PORTA (địa chỉ 05h) và TRISA (địa chỉ
PORTA.

ác thanh ghi sẽ được trình bày cụ thể trong phụ l

ẠO ĐIỆN ÁP SO SÁNH

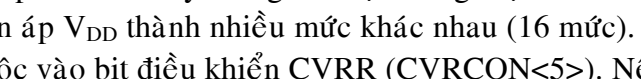
ày chỉ hoạt động khi bộ Comparator được định
in RA0/AN0 và RA1/AN1 (khi CIS = 0) hoặc
à ngõ vào analog của điện áp cần so sánh đưa v
chi tiết ở hình 2.10). Trong khi đó điện áp đưa v
áp so sánh. Sơ đồ khối của bộ tạo điện áp so sá

N2A/BSF-1C/BSF			
----------------	--	--	--

←			
---	--	--	--

CVROE

Hình 2.12 Sơ đồ khối bộ tạo điện áp
áp so sánh này bao gồm một thang điện trở 16



có tác dụng như một thành phần của câu phân

không đi qua điện trở 8R), khi đó 1 mức điện áp có giá trị $V_{DD}/24$. Ngược lại khi CVRR ở mức logic 0, dòng điện sẽ qua điện trở 8R và 1 mức điện áp có giá trị $V_{DD}/32$. Các mức điện áp này được đưa qua bộ MUX cho phép ta chọn được điện áp đưa ra pin RA2/AN2/ V_{REF-}/CV_{REF} để đưa vào ngõ V_{IN+} của bộ so sánh bằng cách đưa các giá trị thích hợp vào các bit CVR3:CVR0.

Bộ tạo điện áp so sánh này có thể xem như một bộ chuyển đổi D/A đơn giản. Giá trị điện áp cần so sánh ở ngõ vào Analog sẽ được so sánh với các mức điện áp do bộ tạo điện áp tạo ra cho tới khi hai điện áp này đạt được giá trị xấp xỉ bằng nhau. Khi đó kết quả chuyển đổi xem như được chứa trong các bit CVR3:CVR0.

Các thanh ghi liên quan đến bộ tạo điện áp so sánh này bao gồm:

Thanh ghi CVRCON (địa chỉ 9Dh): thanh ghi trực tiếp điều khiển bộ so sánh điện áp.

Thanh ghi CMCON (địa chỉ 9Ch): thanh ghi điều khiển bộ Comparator.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.11 CCP

CCP (Capture/Compare/PWM) bao gồm các thao tác trên các xung đếm cung cấp bởi các bộ đếm Timer1 và Timer2. PIC16F877A được tích hợp sẵn hai khối CCP : CCP1 và CCP2. Mỗi CCP có một thanh ghi 16 bit (CCPR1H:CCPR1L và CCPR2H:CCPR2L), pin điều khiển dùng cho khối CCPx là RC2/CCP1 và RC1/T1OSI/CCP2. Các chức năng của CCP bao gồm:

Capture.

So sánh (Compare).

Điều chế độ rộng xung PWM (Pulse Width Modulation).

Cả CCP1 và CCP2 về nguyên tắc hoạt động đều giống nhau và chức năng của từng khối là khá độc lập. Tuy nhiên trong một số trường hợp ngoại lệ CCP1 và CCP2 có khả năng phối hợp với nhau để để tạo ra các hiện tượng đặc biệt (Special event trigger) hoặc các tác động lên Timer1 và Timer2. Các trường hợp này được liệt kê trong bảng sau:

CCPx	CCPy	Tác động
Capture	Capture	Dùng chung nguồn xung clock từ TMR1
Capture	Compare	Tạo ra hiện tượng đặc biệt làm xóa TMR1
Compare	Compare	Tạo ra hiện tượng đặc biệt làm xóa TMR1
PWM	PWM	Dùng chung tần số xung clock và cùng chịu tác động của ngắt TMR2.
PWM	Capture	Hoạt động độc lập
PWM	Compare	Hoạt động độc lập

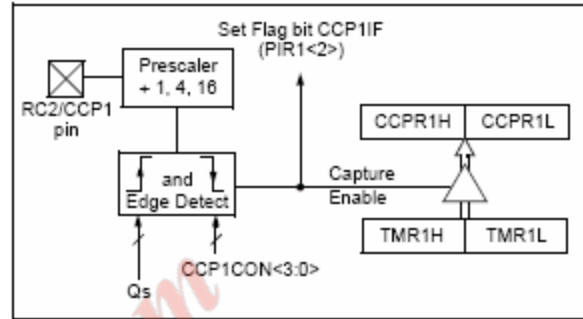
Khi hoạt động ở chế độ Capture thì khi có một “hiện tượng” xảy ra tại pin RC2/CCP1 (hoặc RC1/T1OSI/CCP2), giá trị của thanh ghi TMR1 sẽ được đưa vào thanh ghi CCPR1 (CCPR2). Các “hiện tượng” được định nghĩa bởi các bit CCPxM3:CCPxM0 (CCPxCON<3:0>) và có thể là một trong các hiện tượng sau:

Mỗi khi có cạnh xuống tại các pin CCP.

Mỗi khi có cạnh lên.

Mỗi cạnh lên thứ 4.

Mỗi cạnh lên thứ 16.



Hình 2.13 Sơ đồ khối CCP (Capture mode).

Sau khi giá trị của thanh ghi TMR1 được đưa vào thanh ghi CCPRx, cờ ngắt CCPIF được set và phải được xóa bằng chương trình. Nếu hiện tượng tiếp theo xảy ra mà giá trị trong thanh ghi CCPRx chưa được xử lý, giá trị tiếp theo nhận được sẽ tự động được ghi đè lên giá trị cũ.

Một số điểm cần chú ý khi sử dụng CCP như sau:

Các pin dùng cho khối CCP phải được ấn định là input (set các bit tương ứng trong thanh ghi TRISC). Khi ấn định các pin dùng cho khối CCP là output, việc đưa giá trị vào PORTC cũng có thể gây ra các “hiện tượng” tác động lên khối CCP do trạng thái của pin thay đổi.

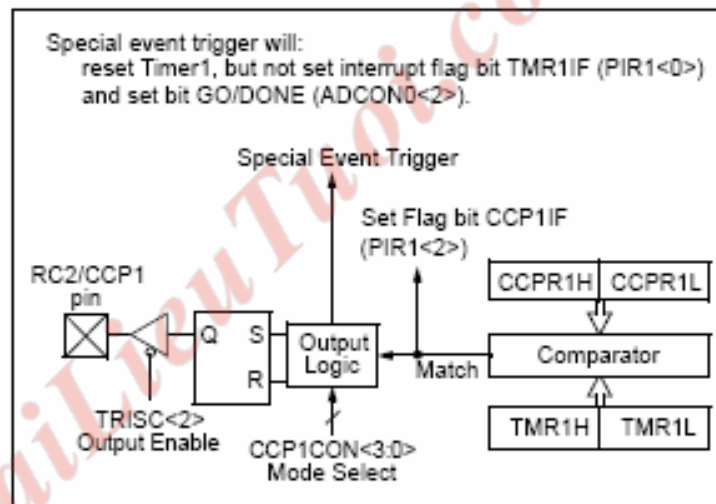
Timer1 phải được hoạt động ở chế độ Timer hoặc chế độ đếm đồng bộ.

Tránh sử dụng ngắt CCP bằng cách clear bit CCPxIE (thanh ghi PIE1), cờ ngắt CCPIF nên được xóa bằng phần mềm mỗi khi được set để tiếp tục nhận định được trạng thái hoạt động của CCP.

CCP còn được tích hợp bộ chia tần số prescaler được điều khiển bởi các bit CCPxM3:CCPxM0. Việc thay đổi đối tượng tác động của prescaler có thể tạo ra hoạt động ngắt. Prescaler được xóa khi CCP không hoạt động hoặc khi reset.

Xem các thanh ghi điều khiển khối CCP (phụ lục 2 để biết thêm chi tiết).

Khi hoạt động ở chế độ Compare, giá trị trong thanh ghi CCPRx sẽ thường xuyên được so sánh với giá trị trong thanh ghi TMR1. Khi hai thanh ghi chứa giá trị bằng nhau, các pin của CCP được thay đổi trạng thái (được đưa lên mức cao, đưa xuống mức thấp hoặc giữ nguyên trạng thái), đồng thời cờ ngắt CCPIF cũng sẽ được set. Sự thay đổi trạng thái của pin có thể được điều khiển bởi các bit CCPxM3:CCPxM0 (CCPxCON <3:0>).

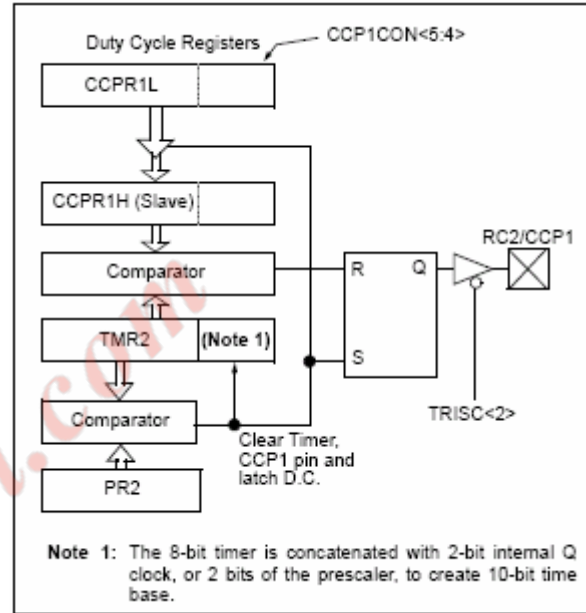


Hình 2.14 Sơ đồ khối CCP (Compare mode).

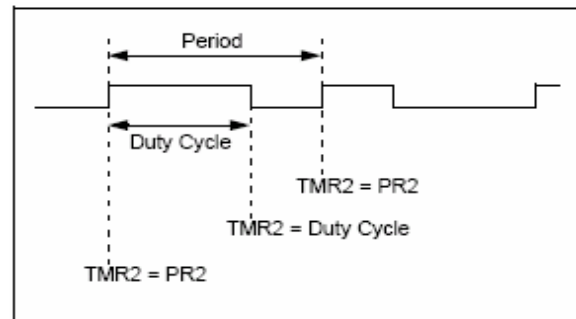
Tương tự như ở chế độ Capture, Timer1 phải được ấn định chế độ hoạt động là timer hoặc đếm đồng bộ. Ngoài ra, khi ở chế độ Compare, CCP có khả năng tạo ra hiện tượng đặc biệt (Special Event trigger) làm reset giá trị thanh ghi TMR1 và khởi động bộ chuyển đổi ADC. Điều này cho phép ta điều khiển giá trị thanh ghi TMR1 một cách linh động hơn.

Khi hoạt động ở chế độ PWM (Pulse Width Modulation _ khối điều chế độ rộng xung), tín hiệu sau khi điều chế sẽ được đưa ra các pin của khối CCP (cần ấn định các pin này là output). Để sử dụng chức năng điều chế này trước tiên ta cần tiến hành các bước cài đặt sau:

1. Thiết lập thời gian của 1 chu kỳ của xung điều chế cho PWM (period) bằng cách đưa giá trị thích hợp vào thanh ghi PR2.
2. Thiết lập độ rộng xung cần điều chế (duty cycle) bằng cách đưa giá trị vào thanh ghi CCPRxL và các bit CCP1CON<5:4>.
3. Điều khiển các pin của CCP là output bằng cách clear các bit tương ứng trong thanh ghi TRISC.
4. Thiết lập giá trị bộ chia tần số prescaler của Timer2 và cho phép Timer2 hoạt động bằng cách đưa giá trị thích hợp vào thanh ghi T2CON.
5. Cho phép CCP hoạt động ở chế độ PWM.



Hình 2.15 Sơ đồ khối CCP (PWM mode).



Hình 2.16 Các tham số của PWM

Trong đó giá trị 1 chu kỳ (period) của xung điều chế được tính bằng công thức:

$$\text{PWM period} = [(PR2)+1]*4*T_{osc}*(\text{giá trị bộ chia tần số của TMR2}).$$

Bộ chia tần số prescaler của Timer2 chỉ có thể nhận các giá trị 1,4 hoặc 16 (xem lại Timer2 để biết thêm chi tiết). Khi giá trị thanh ghi PR2 bằng với giá trị thanh ghi TMR2 thì quá trình sau xảy ra:

Thanh ghi TMR2 tự động được xóa.

Pin của khối CCP được set.

Giá trị thanh ghi CCPR1L (chứa giá trị ấn định độ rộng xung điều chế duty cycle) được đưa vào thanh ghi CCPRxH.

Độ rộng của xung điều chế (duty cycle) được tính theo công thức:

$$\text{PWM duty cycle} = (CCPRxL:CCPxCON<5:4>)*T_{osc}*(\text{giá trị bộ chia tần số TMR2})$$

Như vậy 2 bit CCPxCON<5:4> sẽ chứa 2 bit LSB. Thanh ghi CCPRxL chứa byte cao của giá trị quyết định độ rộng xung. Thanh ghi CCPRxH đóng vai trò là buffer cho khối PWM. Khi giá trị trong thanh ghi CCPRxH bằng với giá trị trong thanh ghi TMR2 và hai bit CCPxCON<5:4> bằng với giá trị 2 bit của bộ chia tần số prescaler, pin của khối CCP lại được đưa về mức thấp, như vậy ta có được hình ảnh của xung điều chế tại ngõ ra của khối PWM như hình 2.14.

Một số điểm cần chú ý khi sử dụng khối PWM:

Timer2 có hai bộ chia tần số prescaler và postscaler. Tuy nhiên bộ postscaler không được sử dụng trong quá trình điều chế độ rộng xung của khối PWM.

Nếu thời gian duty cycle dài hơn thời gian chu kỳ xung period thì xung ngõ ra tiếp tục được giữ ở mức cao sau khi giá trị PR2 bằng với giá trị TMR2.

2.12 GIAO TIẾP NỐI TIẾP

1.12.1 USART

USART (Universal Synchronous Asynchronous Receiver Transmitter) là một trong hai chuẩn giao tiếp nối tiếp. USART còn được gọi là giao diện giao tiếp nối tiếp nối tiếp SCI (Serial Communication Interface). Có thể sử dụng giao diện này cho các giao tiếp với các thiết bị ngoại vi, với các vi điều khiển khác hay với máy tính. Các dạng của giao diện USART ngoại vi bao gồm:

Bất đồng bộ (Asynchronous).

Đồng bộ_ Master mode.

Đồng bộ_ Slave mode.

Hai pin dùng cho giao diện này là RC6/TX/CK và RC7/RX/DT, trong đó RC6/TX/CK dùng để truyền xung clock (baud rate) và RC7/RX/DT dùng để truyền data. Trong trường hợp này ta phải set bit TRISC<7:6> và SPEN (RCSTA<7>) c0 để cho phép giao diện USART.

PIC16F877A được tích hợp sẵn bộ tạo tốc độ baud BRG (Baud Rate Genetator) 8 bit dùng cho giao diện USART. BRG thực chất là một bộ đếm có thể được sử dụng cho cả hai dạng đồng bộ và bất đồng bộ và được điều khiển bởi thanh ghi PSBRG. Ở dạng bất đồng bộ, BRG còn được điều khiển bởi bit BRGH (TXSTA<2>). Ở dạng đồng bộ tác động của bit BRGH được bỏ qua. Tốc độ baud do BRG tạo ra được tính theo công thức sau:

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = $F_{osc}/(64 (X + 1))$	Baud Rate = $F_{osc}/(16 (X + 1))$
1	(Synchronous) Baud Rate = $F_{osc}/(4 (X + 1))$	N/A

Trong đó X là giá trị của thanh ghi RSBRG (X là số nguyên và $0 < X < 255$).

Các thanh ghi liên quan đến BRG bao gồm:

TXSTA (địa chỉ 98h): chọn chế độ đồng bộ hay bất đồng bộ (bit SYNC) và chọn mức tốc độ baud (bit BRGH).

RCSTA (địa chỉ 18h): cho phép hoạt động cổng nối tiếp (bit SPEN).

RSBRG (địa chỉ 99h): quyết định tốc độ baud.

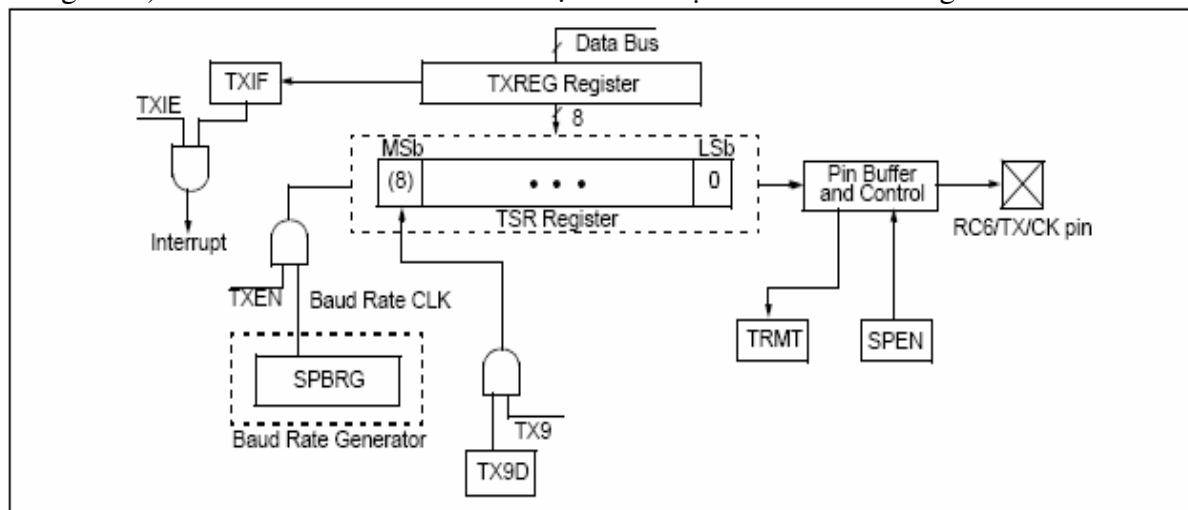
Chi tiết về các thanh ghi sẽ được trình bày cụ thể trong phụ lục 2.

2.12.1.1 USART BẤT ĐỒNG BỘ

Ở chế độ truyền này USART hoạt động theo chuẩn NRZ (None-Return-to-Zero), nghĩa là các bit truyền đi sẽ bao gồm 1 bit Start, 8 hay 9 bit dữ liệu (thông thường là 8 bit) và 1 bit Stop. Bit LSB sẽ được truyền đi trước. Các khối truyền và nhận data độc lập với nhau sẽ dùng chung tần số tương ứng với tốc độ baud cho quá trình dịch dữ liệu (tốc độ baud gấp 16 hay 64 lần tốc độ dịch dữ liệu tùy theo giá trị của bit BRGH), và để đảm bảo tính hiệu quả của dữ liệu thì hai khối truyền và nhận phải dùng chung một định dạng dữ liệu.

2.12.1.1.1 TRUYỀN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART BẤT ĐỒNG BỘ

Thành phần quan trọng nhất của khối truyền dữ liệu là thanh ghi dịch dữ liệu TSR (Transmit Shift Register). Thanh ghi TSR sẽ lấy dữ liệu từ thanh ghi đệm dùng cho quá trình truyền dữ liệu TXREG. Dữ liệu cần truyền phải được đưa trước vào thanh ghi TXREG. Ngay sau khi bit Stop của dữ liệu cần truyền trước đó được truyền xong, dữ liệu từ thanh ghi TXREG sẽ được đưa vào thanh ghi TSR, thanh ghi TXREG bị rỗng, ngắt xảy ra và cờ hiệu TXIF (PIR1<4>) được set. Ngắt này được điều khiển bởi bit TXIE (PIE1<4>). Cờ hiệu TXIF vẫn được set bất chấp trạng thái của bit TXIE hay tác động của chương trình (không thể xóa TXIF bằng chương trình) mà chỉ reset về 0 khi có dữ liệu mới được đưa vào thanh ghi TXREG.



Hình 2.17 Sơ đồ khối của khối truyền dữ liệu USART.

Trong khi cờ hiệu TXIF đóng vai trò chỉ thị trạng thái thanh ghi TXREG thì cờ hiệu TRMT (TXSTA<1>) có nhiệm vụ thể hiện trạng thái thanh ghi TSR. Khi thanh ghi TSR rỗng, bit TRMT sẽ được set. Bit này chỉ đọc và không có ngắt nào được gắn với trạng thái của nó. Một điểm cần chú ý nữa là thanh ghi TSR không có trong bộ nhớ dữ liệu và chỉ được điều khiển bởi CPU.

Khối truyền dữ liệu được cho phép hoạt động khi bit TXEN (TXSTA<5>) được set. Quá trình truyền dữ liệu chỉ thực sự bắt đầu khi đã có dữ liệu trong thanh ghi TXREG và xung truyền baud được tạo ra. Khi khối truyền dữ liệu được khởi động lần đầu tiên, thanh ghi TSR rỗng. Tại thời điểm đó, dữ liệu đưa vào thanh ghi TXREG ngay lập tức được load vào thanh ghi TSR và thanh ghi TXREG bị rỗng. Lúc này ta có thể hình thành một chuỗi dữ liệu liên tục cho quá trình truyền dữ liệu. Trong quá trình truyền dữ liệu nếu bit TXEN bị reset về 0, quá trình truyền kết thúc, khối truyền dữ liệu được reset và pin RC6/TX/CK chuyển đến trạng thái high-impedance.

Trong trường hợp dữ liệu cần truyền là 9 bit, bit TX9 (TXSTA<6>) được set và bit dữ liệu thứ 9 sẽ được lưu trong bit TX9D (TXSTA<0>). Nên ghi bit dữ liệu thứ 9 vào trước, vì khi ghi 8 bit dữ liệu vào thanh ghi TXREG trước có thể xảy ra trường hợp nội dung thanh ghi TXREG sẽ được load vào thanh ghi TSG trước, như vậy dữ liệu truyền đi sẽ bị sai khác so với yêu cầu.

Tóm lại, để truyền dữ liệu theo giao diện USART bất đồng bộ, ta cần thực hiện tuần tự các bước sau:

1. Tạo xung truyền baud bằng cách đưa các giá trị cần thiết vào thanh ghi RSBRG và bit điều khiển mức tốc độ baud BRGH.
2. Cho phép cổng giao diện nối tiếp nối tiếp bất đồng bộ bằng cách clear bit SYNC và set bit PSEN.
3. Set bit TXIE nếu cần sử dụng ngắt truyền.
4. Set bit TX9 nếu định dạng dữ liệu cần truyền là 9 bit.
5. Set bit TXEN để cho phép truyền dữ liệu (lúc này bit TXIF cũng sẽ được set).
6. Nếu định dạng dữ liệu là 9 bit, đưa bit dữ liệu thứ 9 vào bit TX9D.
7. Đưa 8 bit dữ liệu cần truyền vào thanh ghi TXREG.
8. Nếu sử dụng ngắt truyền, cần kiểm tra lại các bit GIE và PEIE (thanh ghi INTCON).

Các thanh ghi liên quan đến quá trình truyền dữ liệu bằng giao diện USART bất đồng bộ:

Thanh ghi INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): cho phép tất cả các ngắt.

Thanh ghi PIR1 (địa chỉ 0Ch): chứa cờ hiệu TXIF.

Thanh ghi PIE1 (địa chỉ 8Ch): chứa bit cho phép ngắt truyền TXIE.

Thanh ghi RCSTA (địa chỉ 18h): chứa bit cho phép cổng truyền dữ liệu (hai pin RC6/TX/CK và RC7/RX/DT).

Thanh ghi TXREG (địa chỉ 19h): thanh ghi chứa dữ liệu cần truyền.

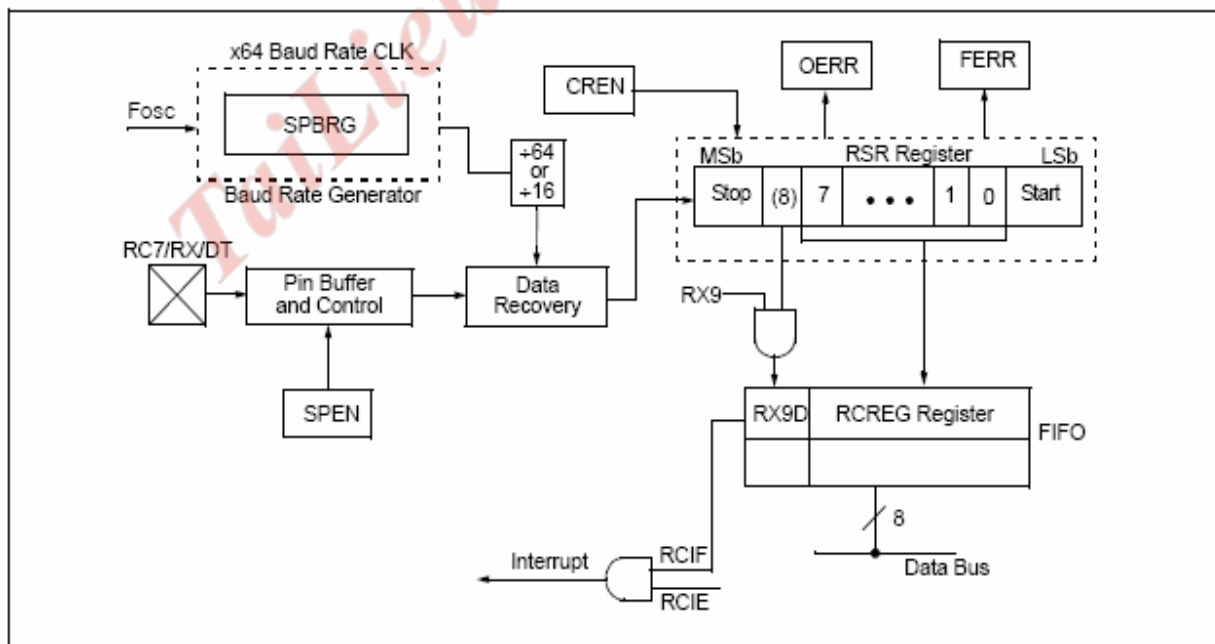
Thanh ghi TXSTA (địa chỉ 98h): xác lập các thông số cho giao diện.

Thanh ghi SPBRG (địa chỉ 99h): quyết định tốc độ baud.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.12.1.1.2 NHẬN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART BẤT ĐỒNG BỘ

Dữ liệu được đưa vào từ chân RC7/RX/DT sẽ kích hoạt khối phục hồi dữ liệu. Khối phục hồi dữ liệu thực chất là một bộ dịch dữ liệu tốc độ cao và có tần số hoạt động gấp 16 lần hoặc 64 lần tần số baud. Trong khi đó tốc độ dịch của thanh ghi nhận dữ liệu sẽ bằng với tần số baud hoặc tần số của oscillator.



Hình 2.18 Sơ đồ khối của khối nhận dữ liệu USART.

Bit điều khiển cho phép khối nhận dữ liệu là bit RCEN (RCSTA<4>). Thành phần quan trọng nhất của khối nhận dữ liệu là thanh ghi nhận dữ liệu RSR (Receive Shift Register). Sau khi nhận diện bit Stop của dữ liệu truyền tới, dữ liệu nhận được trong thanh ghi RSR sẽ được đưa vào thanh ghi RCREG, sau đó cờ hiệu RCIF (PIR1<5>) sẽ được set và ngắt nhận được kích hoạt. Ngắt này được điều khiển bởi bit RCIE (PIE1<5>). Bit cờ hiệu RCIF là bit chỉ đọc và không thể được tác động bởi chương trình. RCIF chỉ reset về 0 khi dữ liệu nhận vào ở thanh ghi RCREG đã được đọc và khi đó thanh ghi RCREG rỗng. Thanh ghi RCREG là thanh ghi

có bộ đệm kép (double-buffered register) và hoạt động theo cơ chế FIFO (First In First Out) cho phép nhận 2 byte và byte thứ 3 tiếp tục được đưa vào thanh ghi RSR. Nếu sau khi nhận được bit Stop của byte dữ liệu thứ 3 mà thanh ghi RCREG vẫn còn đầy, cờ hiệu báo tràn dữ liệu (Overrun Error bit) OERR(RCSTA<1>) sẽ được set, dữ liệu trong thanh ghi RSR sẽ bị mất đi và quá trình đưa dữ liệu từ thanh ghi RSR vào thanh ghi RCREG sẽ bị gián đoạn. Trong trường hợp này cần lấy hết dữ liệu ở thanh ghi RSREG vào trước khi tiếp tục nhận byte dữ liệu tiếp theo. Bit OERR phải được xóa bằng phần mềm và thực hiện bằng cách clear bit RCEN rồi set lại. **Bit FERR (RCSTA<2>) sẽ được set khi phát hiện bit Stop của dữ liệu được nhận vào.** Bit dữ liệu thứ 9 sẽ được đưa vào bit RX9D (RCSTA<0>). Khi đọc dữ liệu từ thanh ghi RCREG, hai bit FERR và RX9D sẽ nhận các giá trị mới. Do đó cần đọc dữ liệu từ thanh ghi RCSTA trước khi đọc dữ liệu từ thanh ghi RCREG để tránh bị mất dữ liệu.

Tóm lại, khi sử dụng giao diện nhận dữ liệu USART bất đồng bộ cần tiến hành tuần tự các bước sau:

1. Thiết lập tốc độ baud (đưa giá trị thích hợp vào thanh ghi SPBRG và bit BRGH).
2. Cho phép cổng giao tiếp USART bất đồng bộ (clear bit SYNC và set bit SPEN).
3. Nếu cần sử dụng ngắt nhận dữ liệu, set bit RCIE.
4. Nếu dữ liệu truyền nhận có định dạng là 9 bit, set bit RX9.
5. Cho phép nhận dữ liệu bằng cách set bit CREN.
6. Sau khi dữ liệu được nhận, bit RCIF sẽ được set và ngắt được kích hoạt (nếu bit RCIE được set).
7. Đọc giá trị thanh ghi RCSTA để đọc bit dữ liệu thứ 9 và kiểm tra xem quá trình nhận dữ liệu có bị lỗi không.
8. Đọc 8 bit dữ liệu từ thanh ghi RCREG.
9. Nếu quá trình truyền nhận có lỗi xảy ra, xóa lỗi bằng cách xóa bit CREN.
10. Nếu sử dụng ngắt nhận cần set bit GIE và PEIE (thanh ghi INTCON).

Các thanh ghi liên quan đến quá trình nhận dữ liệu bằng giao diện USART bất đồng bộ:

Thanh ghi INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): chứa các bit cho phép toàn bộ các ngắt (bit GIER và PEIE).

Thanh ghi PIR1 (địa chỉ 0Ch): chứa cờ hiệu RCIE.

Thanh ghi PIE1 (địa chỉ 8Ch): chứa bit cho phép ngắt RCIE.

Thanh ghi RCSTA (địa chỉ 18h): xác định các trạng thái trong quá trình nhận dữ liệu.

Thanh ghi RCREG (địa chỉ 1Ah): chứa dữ liệu nhận được.

Thanh ghi TXSTA (địa chỉ 98h): chứa các bit điều khiển SYNC và BRGH.

Thanh ghi SPBRG (địa chỉ 99h): điều khiển tốc độ baud.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.12.1.1.2 USART ĐỒNG BỘ

Giao diện USART đồng bộ được kích hoạt bằng cách set bit SYNC. Cổng giao tiếp nối tiếp vẫn là hai chân RC7/RX/DT, RC6/TX/CK và được cho phép bằng cách set bit SPEN. USART cho phép hai chế độ truyền nhận dữ liệu là Master mode và Slave mode. Master mode được kích hoạt bằng cách set bit CSRC (TXSTA<7>), Slave mode được kích hoạt bằng cách clear bit CSRC. Điểm khác biệt duy nhất giữa hai chế độ này là Master mode sẽ lấy xung clock đồng bộ từ bộ tạo xung baud BRG còn Slave mode lấy xung clock đồng bộ từ bên ngoài qua chân RC6/TX/CK. Điều này cho phép Slave mode hoạt động ngay cả khi vi điều khiển đang ở chế độ sleep.

2.12.1.2.1 TRUYỀN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ MASTER MODE

Tương tự như giao diện USART bất đồng bộ, thành phần quan trọng nhất của hồi truyền dữ liệu là thanh ghi dịch TSR (Transmit Shift Register). Thanh ghi này chỉ được điều khiển bởi CPU. Dữ liệu đưa vào thanh ghi TSR được chứa trong thanh ghi TXREG. Cờ hiệu của khối truyền dữ liệu là bit TXIF (chỉ thị trạng thái thanh ghi TXREG), cờ hiệu này được gắn với một ngắt và bit điều khiển ngắt này là TXIE. Cờ hiệu chỉ thị trạng thái thanh ghi TSR là bit TRMT. Bit TXEN cho phép hay không cho phép truyền dữ liệu.

Các bước cần tiến hành khi truyền dữ liệu qua giao diện USART đồng bộ Master mode:

1. Tạo xung truyền baud bằng cách đưa các giá trị cần thiết vào thanh ghi RSBRG và bit điều khiển mức tốc độ baud BRGH.
2. Cho phép cổng giao diện nối tiếp nối tiếp đồng bộ bằng cách set bit SYNC, PSEN và CSRC.
3. Set bit TXIE nếu cần sử dụng ngắt truyền.
4. Set bit TX9 nếu định dạng dữ liệu cần truyền là 9 bit.
5. Set bit TXEN để cho phép truyền dữ liệu.
6. Nếu định dạng dữ liệu là 9 bit, đưa bit dữ liệu thứ 9 vào bit TX9D.
7. Đưa 8 bit dữ liệu cần truyền vào thanh ghi TXREG.
8. Nếu sử dụng ngắt truyền, cần kiểm tra lại các bit GIE và PEIE (thanh ghi INTCON).

Các thanh ghi liên quan đến quá trình truyền dữ liệu bằng giao diện USART đồng bộ Master mode:

Thanh ghi INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): cho phép tất cả các ngắt.

Thanh ghi PIR1 (địa chỉ 0Ch): chứa cờ hiệu TXIF.

Thanh ghi PIE1 (địa chỉ 8Ch): chứa bit cho phép ngắt truyền TXIE.

Thanh ghi RCSTA (địa chỉ 18h): chứa bit cho phép cổng truyền dữ liệu (hai pin RC6/TX/CK và RC7/RX/DT).

Thanh ghi TXREG (địa chỉ 19h): thanh ghi chứa dữ liệu cần truyền.

Thanh ghi TXSTA (địa chỉ 98h): xác lập các thông số cho giao diện.

Thanh ghi SPBRG (địa chỉ 99h): quyết định tốc độ baud.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.12.1.2.2 NHẬN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ MASTER MODE

Cấu trúc khối truyền dữ liệu là không đối xứng với giao diện bất đồng bộ, kể cả các cờ hiệu, ngắt nhận và các thao tác trên các thành phần đó. Điểm khác biệt duy nhất là giao diện này cho phép hai chế độ nhận dữ liệu, đó là chỉ nhận 1 word dữ liệu (set bit SCEN) hay nhận một chuỗi dữ liệu (set bit CREN) cho tới khi ta clear bit CREN. Nếu cả hai bit đều được set, bit điều khiển CREN sẽ được ưu tiên.

Các bước cần tiến hành khi nhận dữ liệu bằng giao diện USART đồng bộ Master mode:

1. Thiết lập tốc độ baud (đưa giá trị thích hợp vào thanh ghi SPBRG và bit BRGH).
2. Cho phép cổng giao tiếp USART bất đồng bộ (set bit SYNC, SPEN và CSRC).
3. Clear bit CREN và SREN.
4. Nếu cần sử dụng ngắt nhận dữ liệu, set bit RCIE.
5. Nếu dữ liệu truyền nhận có định dạng là 9 bit, set bit RX9.
6. Nếu chỉ nhận 1 word dữ liệu, set bit SREN, nếu nhận 1 chuỗi word dữ liệu, set bit CREN.
7. Sau khi dữ liệu được nhận, bit RCIF sẽ được set và ngắt được kích hoạt (nếu bit RCIE được set).
8. Đọc giá trị thanh ghi RCSTA để đọc bit dữ liệu thứ 9 và kiểm tra xem quá trình nhận dữ liệu có bị lỗi không.
9. Đọc 8 bit dữ liệu từ thanh ghi RCREG.
10. Nếu quá trình truyền nhận có lỗi xảy ra, xóa lỗi bằng cách xóa bit CREN.
11. Nếu sử dụng ngắt nhận cần set bit GIE và PEIE (thanh ghi INTCON).

Các thanh ghi liên quan đến quá trình nhận dữ liệu bằng giao diện USART đồng bộ Master mode:

Thanh ghi INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): chứa các bit cho phép toàn bộ các ngắt (bit GIER và PEIE).

Thanh ghi PIR1 (địa chỉ 0Ch): chứa cờ hiệu RCIE.

Thanh ghi PIE1 (địa chỉ 8Ch): chứa bit cho phép ngắt RCIE.

Thanh ghi RCSTA (địa chỉ 18h): xác định các trạng thái trong quá trình nhận dữ liệu.

Thanh ghi RCREG (địa chỉ 1Ah): chứa dữ liệu nhận được.

Thanh ghi TXSTA (địa chỉ 98h): chứa các bit điều khiển SYNC và BRGH.

Thanh ghi SPBRG (địa chỉ 99h): điều khiển tốc độ baud.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.12.1.2.3 TRUYỀN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ SLAVE MODE

Quá trình này không có sự khác biệt so với Master mode khi vi điều khiển hoạt động ở chế độ bình thường. Tuy nhiên khi vi điều khiển đang ở trạng thái sleep, sự khác biệt được thể hiện rõ ràng. Nếu có hai word dữ liệu được đưa vào thanh ghi TXREG trước khi lệnh sleep được thực thi thì quá trình sau sẽ xảy ra:

1. Word dữ liệu đầu tiên sẽ ngay lập tức được đưa vào thanh ghi TSR để truyền đi.
2. Word dữ liệu thứ hai vẫn nằm trong thanh ghi TXREG.
3. Cờ hiệu TXIF sẽ không được set.
4. Sau khi word dữ liệu đầu tiên đã dịch ra khỏi thanh ghi TSR, thanh ghi TXREG tiếp tục truyền word thứ hai vào thanh ghi TSR và cờ hiệu TXIF được set.
5. Nếu ngắt truyền được cho phép hoạt động, ngắt này sẽ đánh thức vi điều khiển và nếu toàn bộ các ngắt được cho phép hoạt động, bộ đếm chương trình sẽ chỉ tới địa chỉ chứa chương trình ngắt (0004h).

Các bước cần tiến hành khi truyền dữ liệu bằng giao diện USART đồng bộ Slave mode:

1. Set bit SYNC, SPEN và clear bit CSRC.
2. Clear bit CREN và SREN.
3. Nếu cần sử dụng ngắt, set bit TXIE.
4. Nếu định dạng dữ liệu là 9 bit, set bit TX9.
5. Set bit TXEN.
6. Đưa bit dữ liệu thứ 9 vào bit TX9D trước (nếu định dạng dữ liệu là 9 bit).
7. Đưa 8 bit dữ liệu vào thanh ghi TXREG.
8. Nếu ngắt truyền được sử dụng, set bit GIE và PEIE (thanh ghi INTCON).

Các thanh ghi liên quan đến quá trình truyền dữ liệu bằng giao diện USART đồng bộ Slave mode:

Thanh ghi INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): cho phép tất cả các ngắt.

Thanh ghi PIR1 (địa chỉ 0Ch): chứa cờ hiệu TXIF.

Thanh ghi PIE1 (địa chỉ 8Ch): chứa bit cho phép ngắt truyền TXIE.

Thanh ghi RCSTA (địa chỉ 18h): chứa bit cho phép cổng truyền dữ liệu (hai pin RC6/TX/CK và RC7/RX/DT).

Thanh ghi TXREG (địa chỉ 19h): thanh ghi chứa dữ liệu cần truyền.

Thanh ghi TXSTA (địa chỉ 98h): xác lập các thông số cho giao diện.

Thanh ghi SPBRG (địa chỉ 99h): quyết định tốc độ baud.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.12.1.2.4 NHẬN DỮ LIỆU QUA CHUẨN GIAO TIẾP USART ĐỒNG BỘ SLAVE MODE

Sự khác biệt của Slave mode so với Master mode chỉ thể hiện rõ ràng khi vi điều khiển hoạt động ở chế độ sleep. Ngoài ra chế độ Slave mode không quan tâm tới bit SREN.

Khi bit CREN (cho phép nhận chuỗi dữ liệu) được set trước khi lệnh sleep được thực thi, 1 word dữ liệu vẫn được tiếp tục nhận, sau khi nhận xong bit thanh ghi RSR sẽ chuyển dữ liệu vào thanh ghi RCREG và bit RCIF được set. Nếu bit RCIE (cho phép ngắt nhận) đã được set trước đó, ngắt sẽ được thực thi và vi điều khiển được “đánh thức, bộ đếm chương trình sẽ chỉ đến địa chỉ 0004h và chương trình ngắt sẽ được thực thi.

Các bước cần tiến hành khi nhận dữ liệu bằng giao diện USART đồng bộ Slave mode:

1. Cho phép cổng giao tiếp USART bất đồng bộ (set bit SYNC, SPEN clear bit CSRC).
2. Nếu cần sử dụng ngắt nhận dữ liệu, set bit RCIE.
3. Nếu dữ liệu truyền nhận có định dạng là 9 bit, set bit RX9.
4. Set bit CREN để cho phép quá trình nhận dữ liệu bắt đầu.
5. Sau khi dữ liệu được nhận, bit RCIF sẽ được set và ngắt được kích hoạt (nếu bit RCIE được set).
6. Đọc giá trị thanh ghi RCSTA để đọc bit dữ liệu thứ 9 và kiểm tra xem quá trình nhận dữ liệu có bị lỗi không.
7. Đọc 8 bit dữ liệu từ thanh ghi RCREG.
8. Nếu quá trình truyền nhận có lỗi xảy ra, xóa lỗi bằng cách xóa bit CREN.
9. Nếu sử dụng ngắt nhận cần set bit GIE và PEIE (thanh ghi INTCON).

Các thanh ghi liên quan đến quá trình nhận dữ liệu bằng giao diện USART đồng bộ Slave mode:

Thanh ghi INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): chứa các bit cho phép toàn bộ các ngắt (bit GIER và PEIE).

Thanh ghi SPBRG (địa chỉ 99h): điều khiển tốc độ baud.

Hình 2.19 Sơ đồ khối MSSP (giao diện SPI)

Thanh ghi điều khiển SSPCON, thanh ghi này cho phép đọc và ghi.

Thanh ghi trạng thái SSPSTAT, thanh ghi này chỉ cho phép đọc và ghi ở 2 bit trên, 6 bit còn lại chỉ cho phép đọc.

Thanh ghi đóng vai trò là buffer truyền nhận SSPBUF, dữ liệu truyền đi hoặc nhận được sẽ được đưa vào thanh ghi này. SSPBUF không có cấu trúc đệm hai lớp (doubled-buffer), do đó dữ liệu ghi vào thanh ghi SSPBUF sẽ lập tức được ghi vào thanh ghi SSPSR.

Thanh ghi dịch dữ liệu SSPSR dùng để dịch dữ liệu vào hoặc ra. Khi 1 byte dữ liệu được nhận hoàn chỉnh, dữ liệu sẽ từ thanh ghi SSPSR chuyển qua thanh ghi SSPBUF và cờ hiệu được set, đồng thời ngắt sẽ xảy ra.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

Khi sử dụng chuẩn giao tiếp SPI trước tiên ta cần thiết lập các chế độ cho giao diện bằng cách đưa các giá trị thích hợp vào hai thanh ghi SSPCON và SSPSTAT. Các thông số cần thiết lập bao gồm:

Master mode hay Slave mode. Đối với Master mode, xung clock đồng bộ sẽ đi ra từ chân RC3/SCK/SCL. Đối với Slave mode, xung clock đồng bộ sẽ được nhận từ bên ngoài qua chân RC3/SCK/SCL.

Các chế độ của Slave mode.

Mức logic của xung clock khi ở trạng thái tạm ngưng quá trình truyền nhận (Idle).

Cạnh tác động của xung clock đồng bộ (cạnh lên hay cạnh xuống).

Tốc độ xung clock (khi hoạt động ở Master mode).

Thời điểm xác định mức logic của dữ liệu (ở giữa hay ở cuối thời gian 1 bit dữ liệu được đưa vào).

Master mode, Slave mode và các chế độ của Slave mode được điều khiển bởi các bit SSPM3:SSPM0 (SSPCON<3:0>). Xem chi tiết ở phụ lục 2.

MSSP bao gồm một thanh ghi dịch dữ liệu SSPSR và thanh ghi đệm dữ liệu SSPBUF. Hai thanh ghi này tạo thành bộ đệm dữ liệu kép (doubled-buffer). Dữ liệu sẽ được dịch vào hoặc ra qua thanh ghi SSPSR, bit MSB được dịch trước. Đây là một trong những điểm khác biệt giữa hai giao diện MSSP và USART (USART dịch bit LSB trước).

Trong quá trình nhận dữ liệu, khi dữ liệu đưa vào từ chân RC4/SDI/SDA trong thanh ghi SSPSR đã sẵn sàng (đã nhận đủ 8 bit), dữ liệu sẽ được đưa vào thanh ghi SSPBUF, bit chỉ thị trạng thái bộ đệm BF (SSPSTAT<0>) sẽ được set để báo hiệu bộ đệm đã đầy, đồng thời cờ ngắt SSPIF (PIR1<3>) cũng được set. Bit BF sẽ tự động reset về 0 khi dữ liệu trong thanh ghi SSPBUF được đọc vào. Bộ đệm kép cho phép đọc tiếp byte tiếp theo trước khi byte dữ liệu trước đó được đọc vào. Tuy nhiên ta nên đọc trước dữ liệu từ thanh ghi SSPBUF trước khi nhận byte dữ liệu tiếp theo.

Quá trình truyền dữ liệu cũng hoàn toàn tương tự nhưng ngược lại. Dữ liệu cần truyền sẽ được đưa vào thanh ghi SSPBUF đồng thời đưa vào thanh ghi SSPSR, khi đó cờ hiệu BF

được set. Dữ liệu được dịch từ thanh ghi SSPSR và đưa ra ngoài qua chân RC5/SDO. Ngắt sẽ xảy ra khi quá trình dịch dữ liệu hoàn tất. Tuy nhiên dữ liệu trước khi được đưa ra ngoài phải được cho phép bởi tín hiệu từ chân $RA5/AN4/\overline{SS}/C2OUT$. Chân này đóng vai trò chọn đối tượng giao tiếp khi SPI ở chế độ Slave mode.

Khi quá trình truyền nhận dữ liệu đang diễn ra, ta không được phép ghi dữ liệu vào thanh ghi SSPBUF. Thao tác ghi dữ liệu này sẽ set bit WCON (SSPCON<7>). Một điều cần chú ý nữa là thanh ghi SSPSR không cho phép truy xuất trực tiếp mà phải thông qua thanh ghi SSPBUF.

Cổng giao tiếp của giao diện SPI được điều khiển bởi bit SSPEN (SSPSON<5>). Bên cạnh đó cần điều khiển chiều xuất nhập của PORTC thông qua thanh ghi TRISC sao cho phù hợp với chiều của giao diện SPI. Cụ thể như sau:

RC4/SDI/SDA sẽ tự động được điều khiển bởi khối giao tiếp SPI.

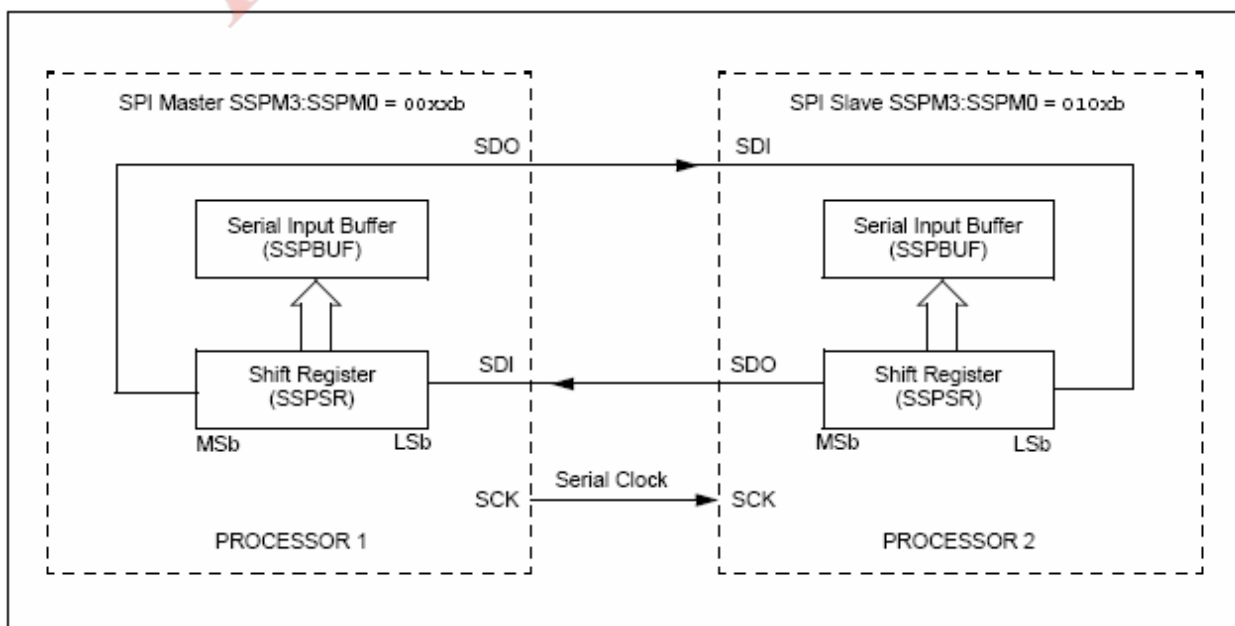
RS5/SDO là ngõ ra dữ liệu, do đó cần clear bit TRISC<5>.

Khi SPI ở dạng Master mode, cần clear bit TRISC<3> để cho phép đưa xung clock đồng bộ ra chân RC3/SCK/SCL.

Khi SPI ở dạng Slave mode, cần set bit TRISC<3> để cho phép nhận xung clock đồng bộ từ bên ngoài qua chân RC3/SCK/SCL.

Set bit TRISC<4> để cho phép chân $RA5/AN4/\overline{SS}/C2OUT$ nhận tín hiệu điều khiển truy xuất dữ liệu khi SPI ở chế độ Slave mode.

Sơ đồ kết nối của chuẩn giao tiếp SPI như sau:



Hình 2.20 Sơ đồ kết nối của chuẩn giao tiếp SPI.

Theo sơ đồ kết nối này, khối Master sẽ bắt đầu quá trình truyền nhận dữ liệu bằng cách gửi tín hiệu xung đồng bộ SCK. Dữ liệu sẽ dịch từ cả hai thanh ghi SSPSR đưa ra ngoài nếu có một cạnh của xung đồng bộ tác động và ngưng dịch khi có tác động của cạnh còn lại.

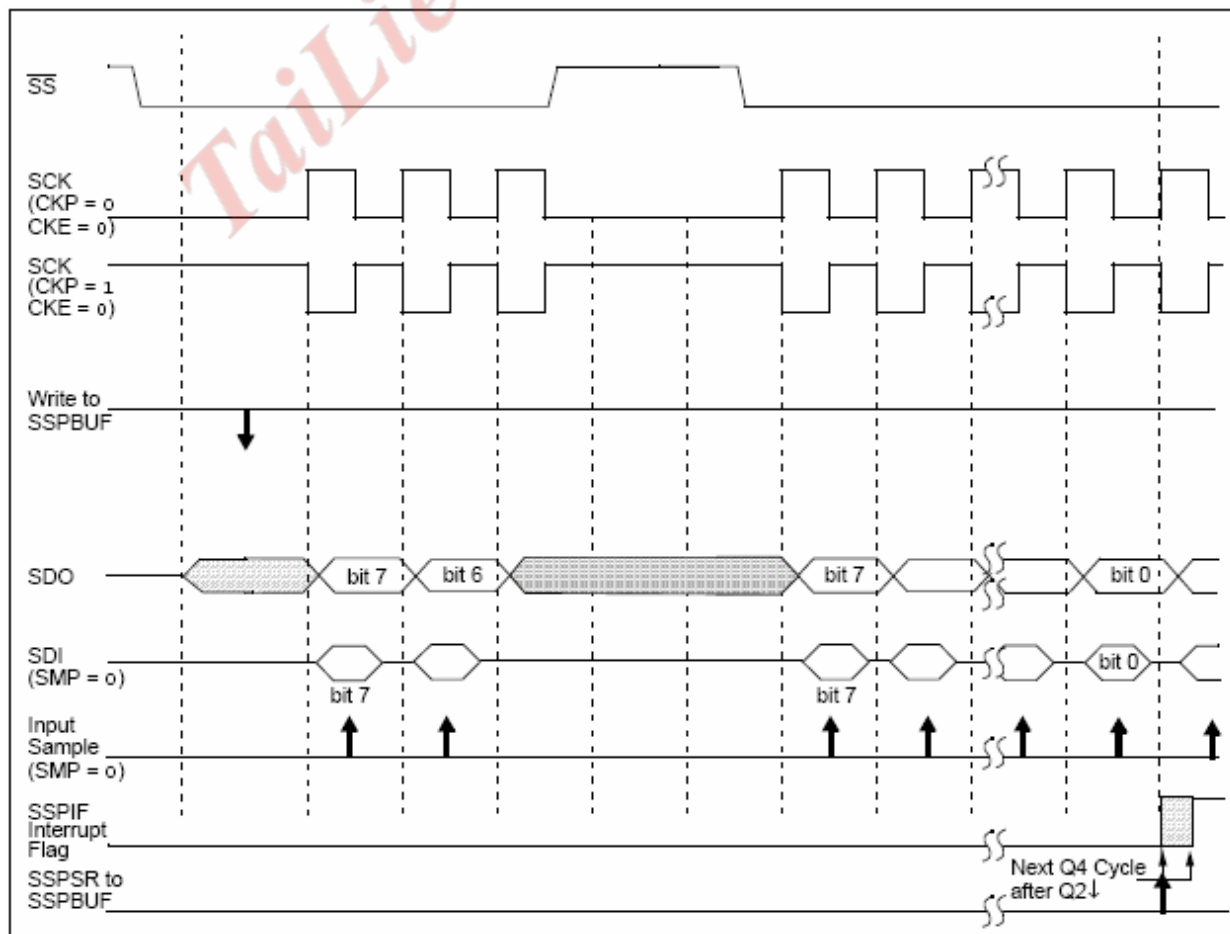
Cả hai khối Master và Slave nên được ấn định chung các qui tắc tác động của xung clock đồng bộ để dữ liệu có thể dịch chuyển đồng thời.

2.12.2.1.1 SPI MASTER MODE.

Ở chế độ Master mode, vi điều khiển có quyền ấn định thời điểm trao đổi dữ liệu (và đối tượng trao đổi dữ liệu nếu cần) vì nó điều khiển xung clock đồng bộ. Dữ liệu sẽ được truyền nhận ngay thời điểm dữ liệu được đưa vào thanh ghi SSPBUF. Nếu chỉ cần nhận dữ liệu, ta có thể ấn định chân SDO là ngõ vào (set bit TRISC<5>). Dữ liệu sẽ được dịch vào thanh ghi SSPSR theo một tốc độ được định sẵn cho xung clock đồng bộ. Sau khi nhận được một byte dữ liệu hoàn chỉnh, byte dữ liệu sẽ được đưa vào thanh ghi SSPBUF, bit BF được set và ngắt xảy ra.

Khi lệnh SLEEP được thực thi trong quá trình truyền nhận, trạng thái của quá trình sẽ được giữ nguyên và tiếp tục sau khi vi điều khiển được đánh thức.

Giải đồ xung của Master mode và các tác động của các bit điều khiển được trình bày trong hình vẽ sau:

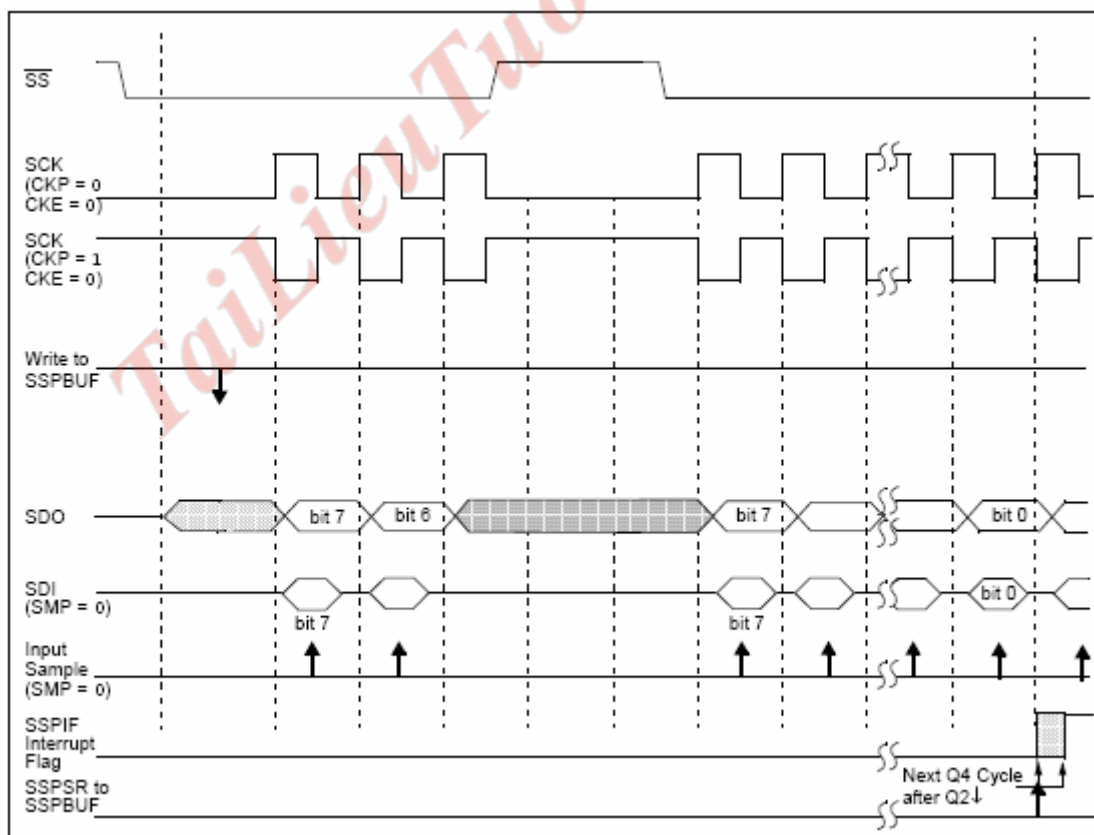


Hình 2.21 Giải đồ xung SPI ở chế độ Master mode.

2.12.2.1.2 SPI SLAVE MODE

Ở chế độ này SPI sẽ truyền và nhận dữ liệu khi có xung đồng bộ xuất hiện ở chân SCK. Khi truyền nhận xong bit dữ liệu cuối cùng, cờ ngắt SSPIF sẽ được set. Slave mode hoạt động ngay cả khi vi điều khiển đang ở chế độ sleep, và ngắt truyền nhận cho phép “đánh thức” vi điều khiển. Khi chỉ cần nhận dữ liệu, ta có thể ấn định RC5/SDO là ngõ vào (set bit TRISC<5>).

Slave mode cho phép sự tác động của chân điều khiển $\overline{RA5/AN4/\overline{SS}/C2OUT}$ (SSPCON<3:0> = 0100). Khi chân $\overline{RA5/AN4/\overline{SS}/C2OUT}$ ở mức thấp, chân RC5/SDO được cho phép xuất dữ liệu và khi $\overline{RA5/AN4/\overline{SS}/C2OUT}$ ở mức cao, dữ liệu ra ở chân RC5/SDO bị khóa, đồng thời SPI được reset (bộ đếm bit dữ liệu được gán giá trị 0).



Hình 2.22 Giải đồ xung chuẩn giao tiếp SPI (Slave mode).

Các thanh ghi liên quan đến chuẩn giao tiếp SPI bao gồm:

Thanh ghi INTCON (địa chỉ 0Bh, 8Bh, 10Bh, 18Bh): chứa bit cho phép toàn bộ các ngắt (GIE và PEIE).

Thanh ghi PIR1 (địa chỉ 0Ch): chứa ngắt SSPIE.

Thanh ghi PIE1 (địa chỉ 8Ch): chứa bit cho phép ngắt SSPIE.

Thanh ghi TRISC (địa chỉ 87h): điều khiển xuất nhập PORTC.

Thanh ghi SSPBUF (địa chỉ 13h): thanh ghi đệm dữ liệu.

Thanh ghi SSPCON (địa chỉ 14h): điều khiển chuẩn giao tiếp SPI.

Thanh ghi SSPSTAT (địa chỉ 94h): chứa các bit chỉ thị trạng thái chuẩn giao tiếp SPI.

Thanh ghi TRISA (địa chỉ 85h): điều khiển xuất nhập chân $RA5/AN4/\overline{SS}/C2OUT$.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

2.12.2.2 I2C

Đây là một dạng khác của MSSP. Chuẩn giao tiếp I2C cũng có hai chế độ Master, Slave và cũng được kết nối với ngắt. I2C sẽ sử dụng 2 pin để truyền nhận dữ liệu:

RC3/SCK/SCL: chân truyền dẫn xung clock.

RC4/SDI/SDA: chân truyền dẫn dữ liệu.

Các khối cơ bản trong sơ đồ khối của I2C không có nhiều khác biệt so với SPI. Tuy nhiên I2C còn có thêm khối phát hiện bit Start và bit Stop của dữ liệu (Start and Stop bit detect) và khối xác định địa chỉ (Match detect).

Các thanh ghi liên quan đến I2C bao gồm:

Thanh ghi SSPCON và SSPCON2: điều khiển MSSP.

Thanh ghi SSPSTAT: thanh ghi chứa các trạng thái hoạt động của MSSP.

Thanh ghi SSPBUF: buffer truyền nhận nối tiếp.

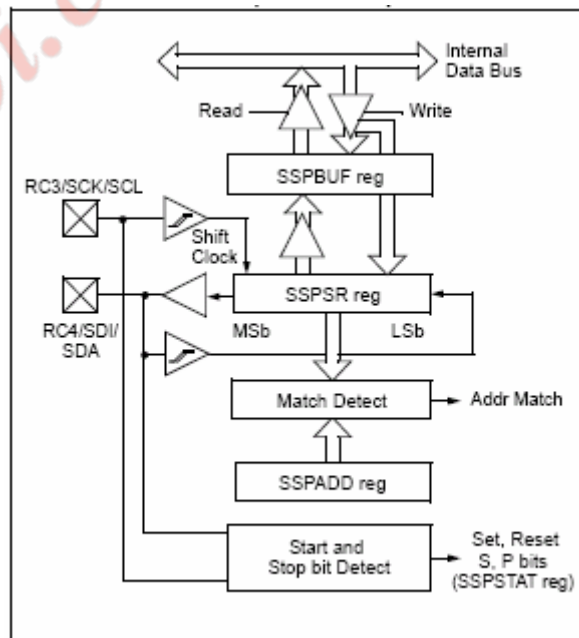
Thanh ghi SSPSR: thanh ghi dịch dùng để truyền nhận dữ liệu.

Thanh ghi SSPADD: thanh ghi chứa địa chỉ của giao diện MSSP.

Các thanh ghi SSPCON, SSPCON2 cho phép đọc và ghi. Thanh ghi SSPSTAT chỉ cho phép đọc và ghi ở 2 bit đầu, 6 bit còn lại chỉ cho phép đọc.

Thanh ghi SSPBUF chứa dữ liệu sẽ được truyền đi hoặc nhận được và đóng vai trò như một thanh ghi đệm cho thanh ghi dịch dữ liệu SSPSR.

Thanh ghi SSPADD chứa địa chỉ của thiết bị ngoại vi cần truy xuất dữ liệu của I2C khi hoạt động ở Slave mode. Khi hoạt động ở Master mode, thanh ghi SSPADD chứa giá trị tạo ra tốc độ baud cho xung clock dùng để truyền nhận dữ liệu.



Hình 2.23 Sơ đồ khối MSSP (I2C slave mode).

Trong quá trình nhận dữ liệu, sau khi nhận được 1 byte dữ liệu hoàn chỉnh, thanh ghi SSPSR sẽ chuyển dữ liệu vào thanh ghi SSPBUF. Thanh ghi SSPSR không đọc và ghi được, quá trình truy xuất thanh ghi này phải thông qua thanh ghi SSPBUF.

Trong quá trình truyền dữ liệu, dữ liệu cần truyền khi được đưa vào thanh ghi SSPBUF cũng sẽ đồng thời đưa vào thanh ghi SSPSR.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.

I2C có nhiều chế độ hoạt động và được điều khiển bởi các bit SSPCON<3:0>, bao gồm:

I2C Master mode, xung clock = $f_{osc}/4 * (SSPADD+1)$.

I2C Slave mode, 7 bit địa chỉ.

I2C Slave mode, 10 bit địa chỉ.

I2C Slave mode, 7 bit địa chỉ, cho phép ngắt khi phát hiện bit Start và bit Stop.

I2C Slave mode, 10 bit địa chỉ, cho phép ngắt khi phát hiện bit Start và bit Stop.

I2C Firmware Control Master mode.

Địa chỉ truyền đi sẽ bao gồm các bit địa chỉ và một bit R/\overline{W} để xác định thao tác (đọc hay ghi dữ liệu) với đối tượng cần truy xuất dữ liệu.

Khi lựa chọn giao diện I2C và khi set bit SSPEN, các pin SCL và SDA sẽ ở trạng thái cực thu hở. Do đó trong trường hợp cần thiết ta phải sử dụng điện trở kéo lên ở bên ngoài vì điều khiển, bên cạnh đó cần ấn định các giá trị phù hợp cho các bit TRISC<4:3> (bit điều khiển xuất nhập các chân SCL và SDA).

2.12.2.2.1 I2C SLAVE MODE.

Việc trước tiên là phải set các pin SCL và SDA là input (set bit TRISC<4:3>). I2C của vi điều khiển sẽ được điều khiển bởi một vi điều khiển hoặc một thiết bị ngoại vi khác thông qua các địa chỉ. Khi địa chỉ này chỉ đến vi điều khiển, thì tại thời điểm này và tại thời điểm dữ liệu đã được truyền nhận xong sau đó, vi điều khiển sẽ tạo ra xung \overline{ACK} để báo hiệu kết thúc dữ liệu, giá trị trong thanh ghi SSPSR sẽ được đưa vào thanh ghi SSPBUF. Tuy nhiên xung \overline{ACK} sẽ không được tạo ra nếu một trong các trường hợp sau xảy ra:

Bit BF (SSPSTAT<0>) báo hiệu buffer đầy đã được set trước khi quá trình truyền nhận xảy ra.

Bit SSPOV (SSPCON<6>) được set trước khi quá trình truyền nhận xảy ra (SSPOV được set trong trường hợp khi một byte khác được nhận vào trong khi dữ liệu trong thanh ghi SSPBUF trước đó vẫn chưa được lấy ra).

Trong các trường hợp trên, thanh ghi SSPSR sẽ không đưa giá trị vào thanh ghi SSPBUF, nhưng bit SSPIF (PIR1<3>) sẽ được set. Để quá trình truyền nhận dữ liệu được tiếp tục, cần đọc dữ liệu từ thanh ghi SSPBUF vào trước, khi đó bit BF sẽ tự động được xóa, còn bit SSPOV phải được xóa bằng chương trình.

Khi MSSP được kích hoạt, nó sẽ chờ tín hiệu để bắt đầu hoạt động. Sau khi nhận được tín hiệu bắt đầu hoạt động (cạnh xuống đầu tiên của pin SDA), dữ liệu 8 bit sẽ được dịch vào thanh ghi SSPSR. Các bit đưa vào sẽ được lấy mẫu tại cạnh lên của xung clock. Giá trị nhận được từ thanh ghi SSPSR sẽ được so sánh với giá trị trong thanh ghi SSPADD tại cạnh xuống của xung clock thứ 8. Nếu kết quả so sánh bằng nhau, tức là I2C Master chỉ định đối tượng giao tiếp là vi điều khiển đang ở chế độ Slave mode (ta gọi hiện tượng này là address match), bit BF và SSPOV sẽ được xóa về 0 và gây ra các tác động sau:

1. Giá trị trong thanh ghi SSPSR được đưa vào thanh ghi SSPBUF.
2. Bit BF tự động được set.
3. Một xung \overline{ACK} được tạo ra.
4. Cờ ngắt SSPIF được set (ngắt được kích hoạt nếu được cho phép trước đó) tại cạnh xuống của xung clock thứ 9.

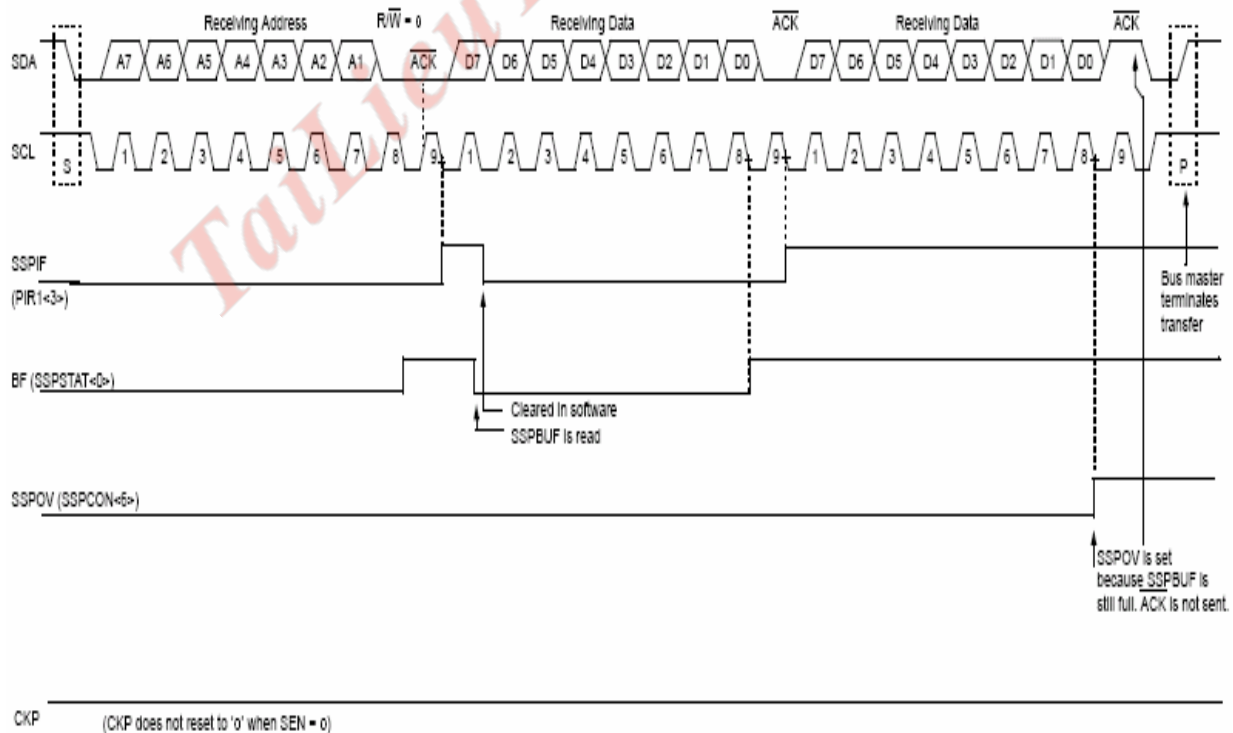
Khi MSSP ở chế độ I2C Slave mode 10 bit địa chỉ, vi điều khiển cần phải nhận vào 10 bit địa chỉ để so sánh. Bit R/\overline{W} (SSPSTAT<2>) phải được xóa về 0 để cho phép nhận 2 byte địa chỉ. Byte đầu tiên có định dạng là '11110 A9 A8 0' trong đó A9, A8 là hai bit MSB của 10 bit địa chỉ. Byte thứ 2 là 8 bit địa chỉ còn lại.

Quá trình nhận dạng địa chỉ của MSSP ở chế độ I2C Slave mode 10 bit địa chỉ như sau:

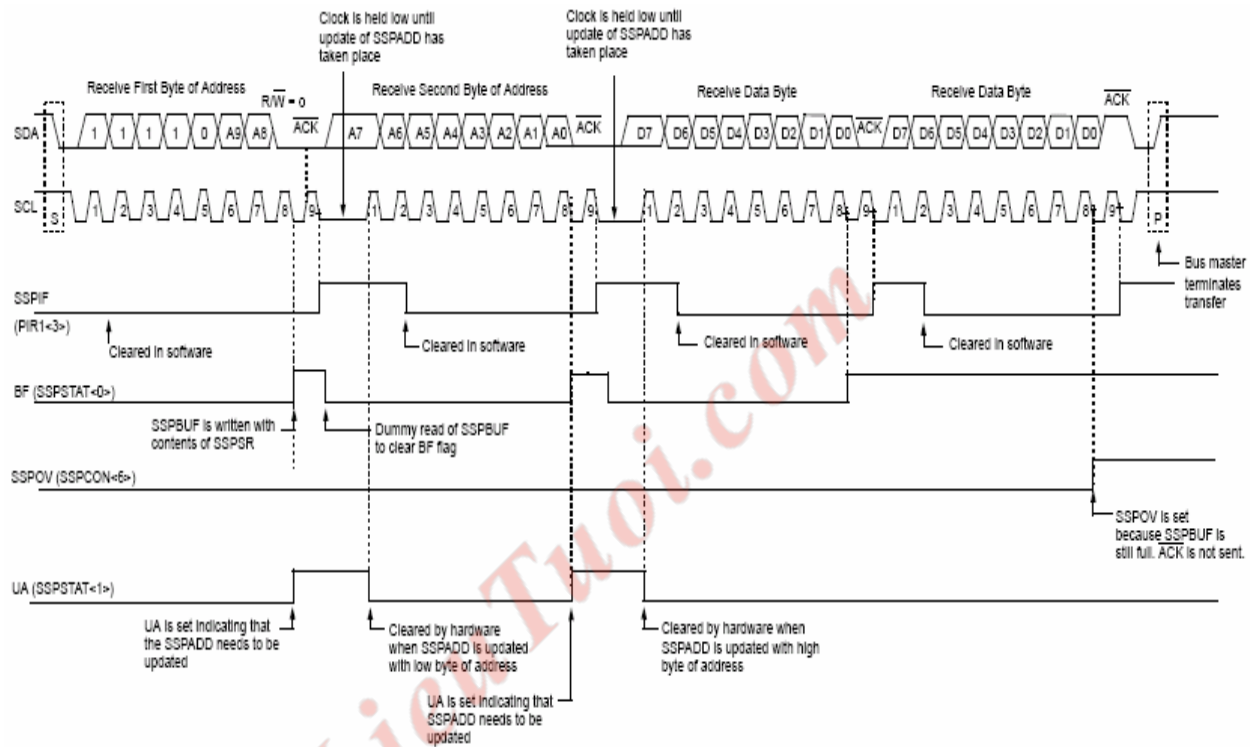
1. Đầu tiên 2 bit MSB của 10 bit địa chỉ được nhận trước, bit SSPIF, BF và UA (SSPSTAT<1>) được set (byte địa chỉ đầu tiên có định dạng là '11110 A9 A8 0').
2. Cập nhật vào 8 bit địa chỉ thấp của thanh ghi SSPADD, bit UA sẽ được xóa bởi vi điều khiển để khởi tạo xung clock ở pin SCL sau khi quá trình cập nhật hoàn tất.
3. Đọc giá trị thanh ghi SSPBUF (bit BF sẽ được xóa về 0) và xóa cờ ngắt SSPIF.
4. Nhận 8 bit địa chỉ cao, bit SSPIF, BF và UA được set.
5. Cập nhật 8 bit địa chỉ đã nhận được vào 8 bit địa chỉ cao của thanh ghi SSPADD, nếu địa chỉ nhận được là đúng (address match), xung clock ở chân SCL được khởi tạo và bit UA được set.
6. Đọc giá trị thanh ghi SSPBUF (bit BF sẽ được xóa về 0) và xóa cờ ngắt SSPIF.
7. Nhận tín hiệu Start.
8. Nhận byte địa chỉ cao (bit SSPIF và BF được set).
9. Đọc giá trị thanh ghi SSPBUF (bit BF được xóa về 0) và xóa cờ ngắt SSPIF.

Trong đó các bước 7,8,9 xảy ra trong quá trình truyền dữ liệu ở chế độ Slave mode. Xem giản đồ xung của I2C để có được hình ảnh cụ thể hơn về các bước tiến hành trong quá trình nhận dạng địa chỉ.

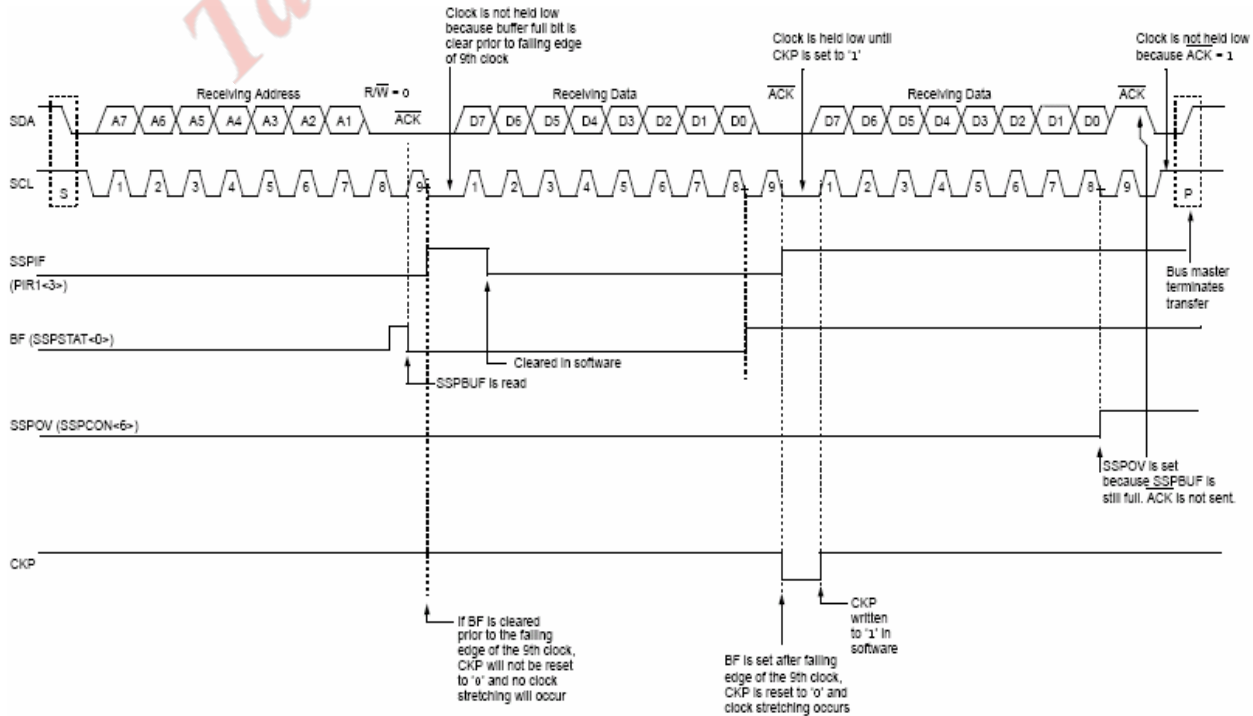
Xét quá trình nhận dữ liệu ở chế độ Slave mode, các bit địa chỉ sẽ được I2C Master đưa vào trước. Khi bit $\overline{R/W}$ trong các bit địa chỉ có giá trị bằng 0 (bit này được nhận dạng sau khi các bit địa chỉ đã được nhận xong) và địa chỉ được chỉ định đúng (address match), bit $\overline{R/W}$ của thanh ghi SSPSTAT được xóa về 0 và đường dữ liệu SDI được đưa về mức logic thấp (xung \overline{ACK}). Khi bit SEN (SSPCON<0>) được set, sau khi 1 byte dữ liệu được nhận, xung clock từ chân RC3/SCK/SCL sẽ được đưa xuống mức thấp, muốn khởi tạo lại xung clock ta set bit CKP (SSPCON<4>). Điều này sẽ làm cho hiện tượng tràn dữ liệu không xảy ra vì bit SEN cho phép ta điều khiển được xung clock dịch dữ liệu thông qua bit CKP (tham khảo giản đồ xung để biết thêm chi tiết). Khi hiện tượng tràn dữ liệu xảy ra, bit BF hoặc bit SSPOV sẽ được set. Ngắt sẽ xảy ra khi một byte dữ liệu được nhận xong, cờ ngắt SSPIF sẽ được set và phải được xóa bằng chương trình.



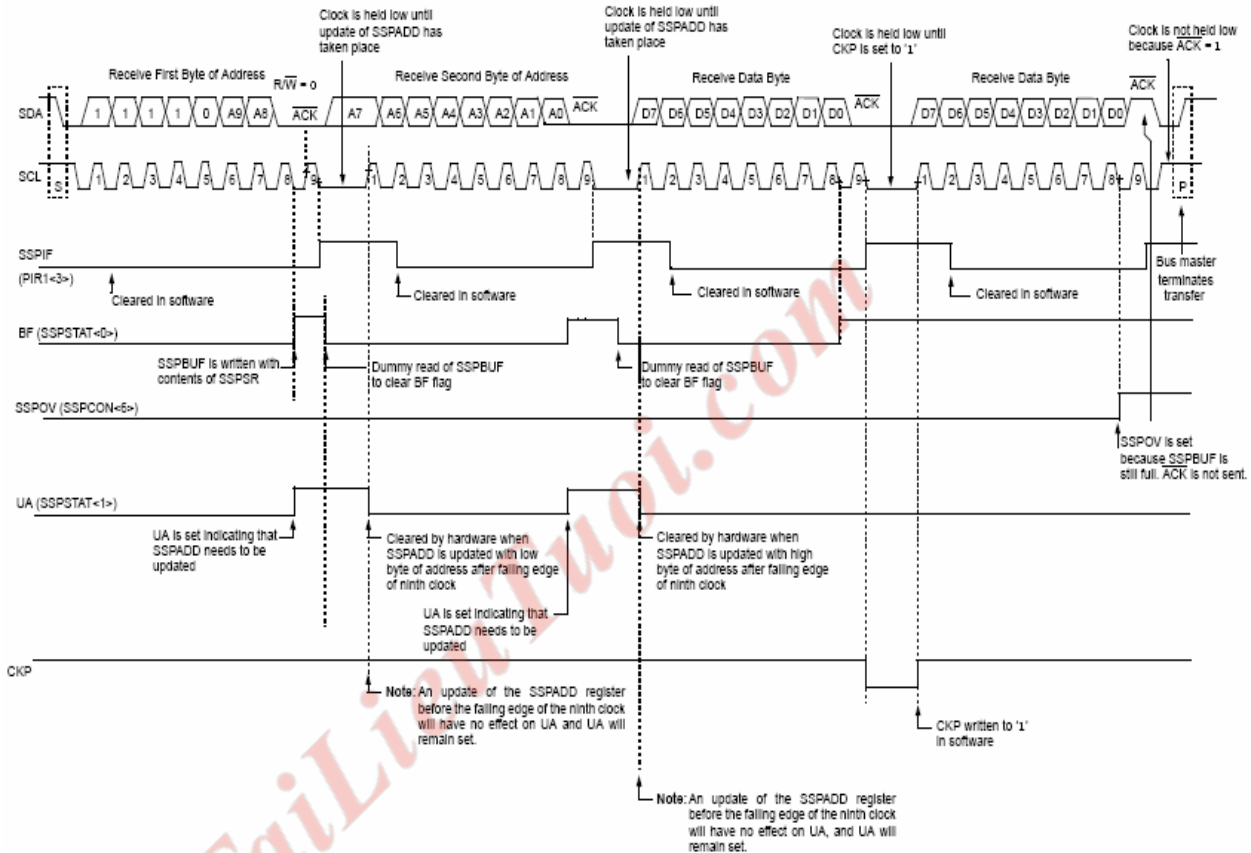
Hình 2.24 Giản đồ xung của I2C Slave mode 7 bit địa chỉ trong quá trình nhận dữ liệu (bit SEN = 0).



Hình 2.25 Giải đồ xung của I2C Slave mode 10 bit địa chỉ trong quá trình nhận dữ liệu (bit SEN = 0).



Hình 2.26 Giải đồ xung của I2C Slave mode 7 bit địa chỉ trong quá trình nhận dữ liệu (bit SEN = 1).



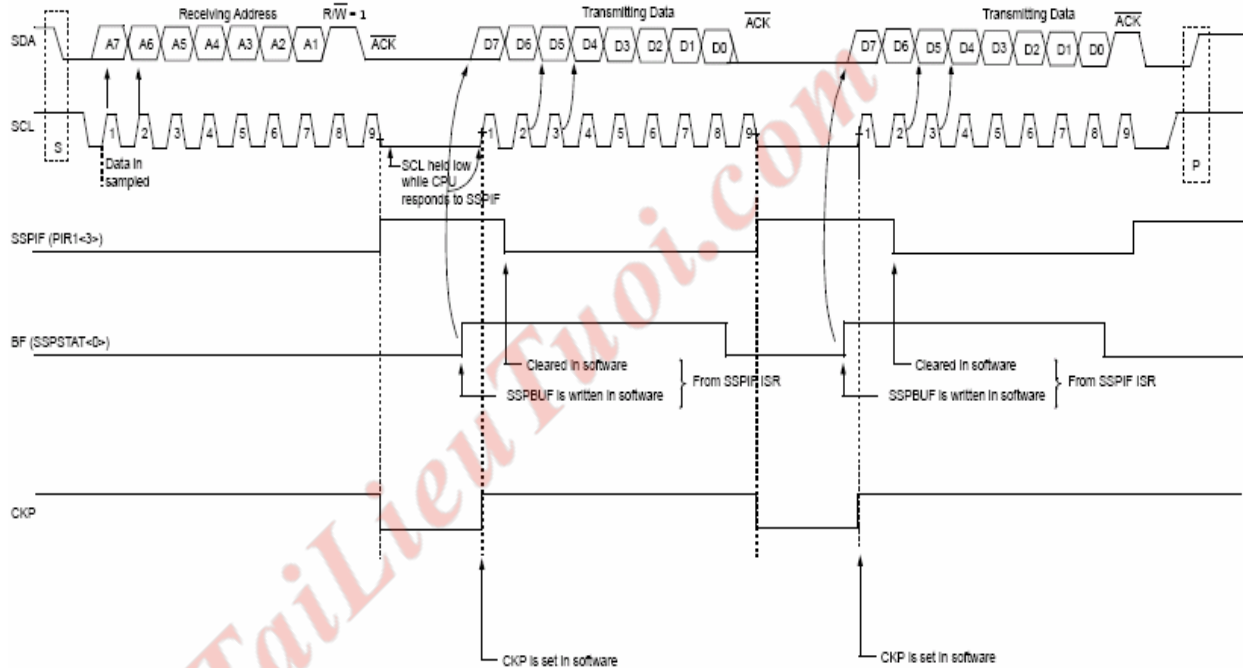
Hình 2.27 Giản đồ xung của I2C Slave mode 10 bit địa chỉ trong quá trình nhận dữ liệu (bit SEN = 1).

Xét quá trình truyền dữ liệu, khi bit R/\overline{W} trong các bit dữ liệu mang giá trị 1 và địa chỉ được chỉ định đúng (address match), bit R/\overline{W} của thanh ghi SSPSTAT sẽ được set. Các bit địa chỉ được nhận trước và đưa vào thanh ghi SSPBUF. Sau đó xung \overline{ACK} được tạo ra, xung clock ở chân RC3/SCK/SCL được đưa xuống mức thấp bất chấp trạng thái của bit SEN. Khi đó I2C Master sẽ không được đưa xung clock vào I2C Slave cho đến khi dữ liệu ở thanh ghi SSPSR ở trạng thái sẵn sàng cho quá trình truyền dữ liệu (dữ liệu đưa vào thanh ghi SSPBUF sẽ đồng thời được đưa vào thanh ghi SSPSR). Tiếp theo cho phép xung ở pin RC3/SCK/SCL bằng cách set bit CKP (SSPCON<4>). Từng bit của byte dữ liệu sẽ được dịch ra ngoài tại mỗi cạnh xuống của xung clock. Như vậy dữ liệu sẽ sẵn sàng ở ngõ ra khi xung clock ở mức logic cao, giúp cho I2C Master nhận được dữ liệu tại mỗi cạnh lên của xung clock. Như vậy trong quá trình truyền dữ liệu bit SEN không đóng vai trò quan trọng như trong quá trình nhận dữ liệu.

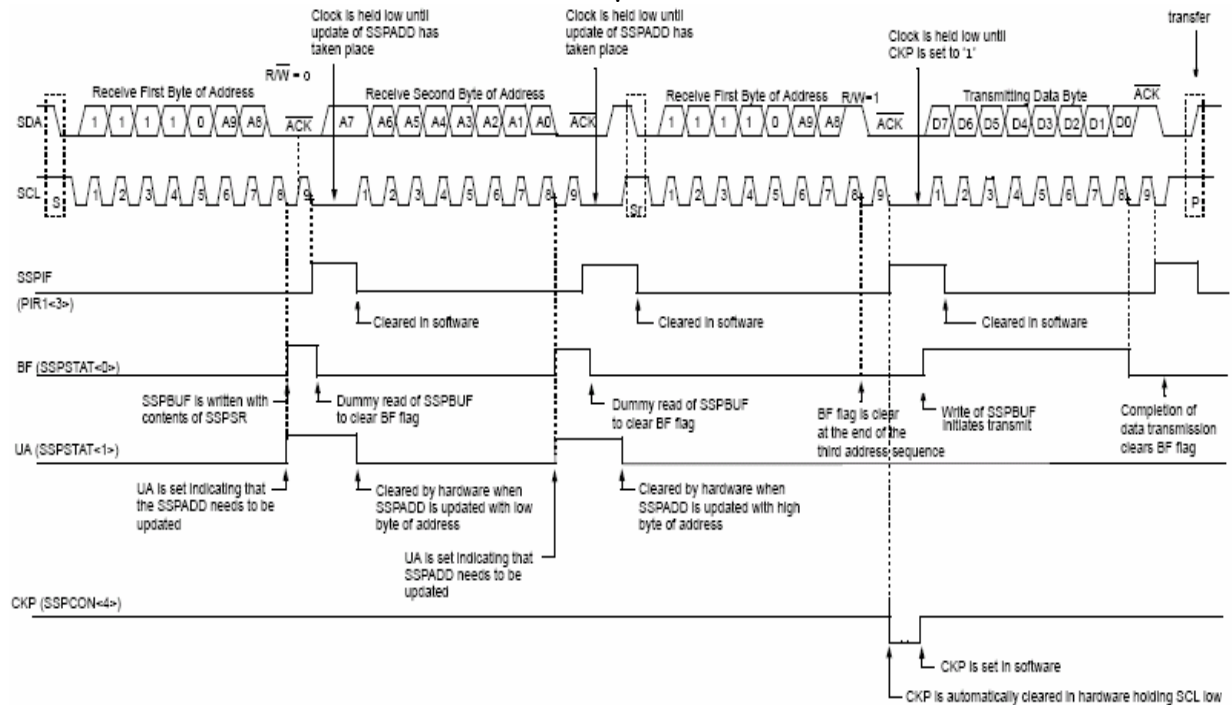
Tại cạnh lên xung clock thứ 9, dữ liệu đã được dịch hoàn toàn vào I2C Master, xung \overline{ACK} sẽ được tạo ra ở I2C Master, đồng thời pin SDA sẽ được giữ ở mức logic cao. Trong trường hợp xung \overline{ACK} được chốt bởi I2C Slave, thanh ghi SSPSTAT sẽ được reset. I2C Slave

sẽ chờ tín hiệu của bit Start để tiếp tục truyền byte dữ liệu tiếp theo (đưa byte dữ liệu tiếp theo vào thanh ghi SSPBUF và set bit CKP).

Ngắt MSSP xảy ra khi một byte dữ liệu kết thúc quá trình truyền, bit SSPIF được set tại cạnh xuống của xung clock thứ 9 và phải được xóa bằng chương trình để đảm bảo sẽ được set khi byte dữ liệu tiếp theo truyền xong.

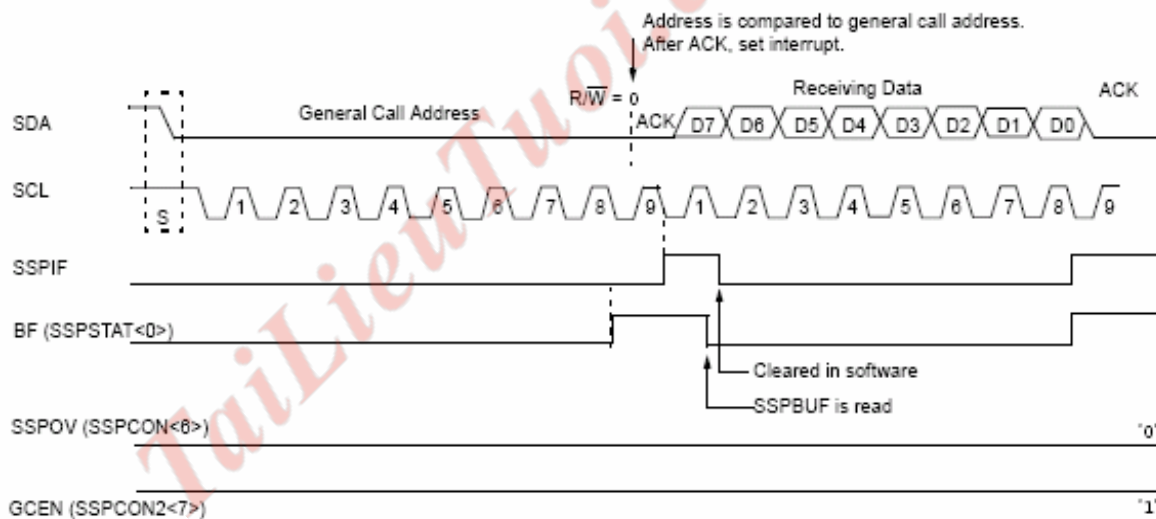


Hình 2.28 Giải đồ xung của I2C Slave mode 7 bit địa chỉ trong quá trình truyền dữ liệu.



Hình 2.29 Giải đồ xung của I2C Slave mode 10 bit địa chỉ trong quá trình truyền dữ liệu.

Quá trình truyền nhận các bit địa chỉ cho phép I2C Master chọn lựa đối tượng I2C Slave cần truy xuất dữ liệu. Bên cạnh đó I2C còn cung cấp thêm một địa chỉ GCA (General Call Address) cho phép chọn tất cả các I2C Slave. Đây là một trong 8 địa chỉ đặc biệt của protocol I2C. Địa chỉ này được định dạng là một chuỗi '0' với $R/\overline{W} = 0$ và được cho phép bằng cách set bit GCEN (SSPCON2<7>). Khi đó địa chỉ nhận vào sẽ được so sánh với thanh ghi SSPADD và với địa chỉ GCA.

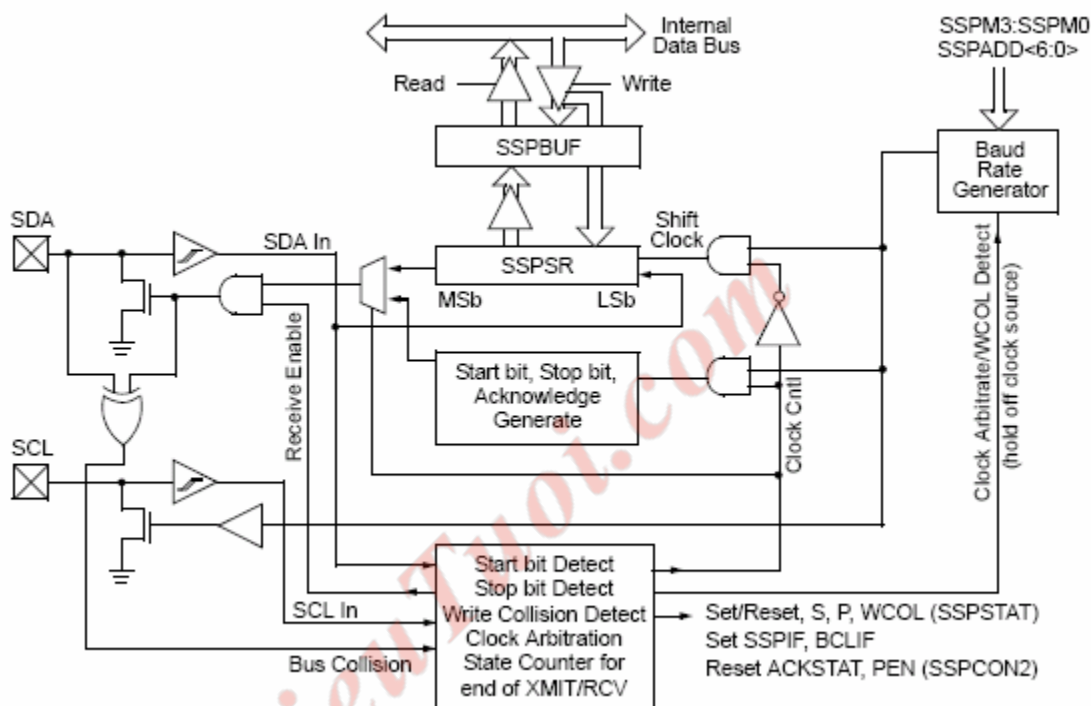


Hình 2.30 Giải đồ xung của I2C Slave khi nhận địa chỉ GCA.

Quá trình nhận dạng địa chỉ GCA cũng tương tự như khi nhận dạng các địa chỉ khác và không có sự khác biệt rõ ràng khi I2C hoạt động ở chế độ địa chỉ 7 bit hay 10 bit.

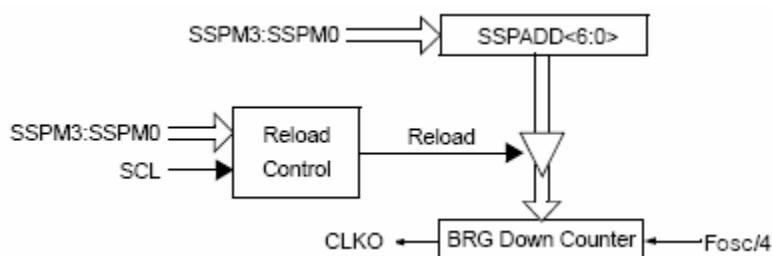
2.12.2.2.2 I2C MASTER MODE

I2C Master mode được xác lập bằng cách đưa các giá trị thích hợp vào các bit SSPM của thanh ghi SSPCON và set bit SSPEN. Ở chế độ Master, các pin SCK và SDA sẽ được điều khiển bởi phần cứng của MSSP.



I2C Master đóng vai trò tích cực trong quá trình giao tiếp và điều khiển các I2C Slave thông qua việc chủ động tạo ra xung giao tiếp và các điều kiện Start, Stop khi truyền nhận dữ liệu. Một byte dữ liệu có thể được bắt đầu bằng điều kiện Start, kết thúc bằng điều kiện Stop hoặc bắt đầu và kết thúc với cùng một điều kiện khởi động lặp lại (Repeated Start Condition).

Xung giao tiếp nối tiếp sẽ được tạo ra từ BRG (Baud Rate Generator), giá trị ấn định tần số xung clock nối tiếp được lấy từ 7 bit thấp của thanh ghi SSPADD. Khi dữ liệu được đưa vào thanh ghi SSPBUF, bit BF được set và BRG tự động đếm ngược về 0 và dừng lại, pin SCL được giữ nguyên trạng thái trước đó. Khi dữ liệu tiếp theo được đưa vào, BRG sẽ cần một khoảng thời gian T_{BRG} tự động reset lại giá trị để tiếp tục quá trình đếm ngược. Mỗi vòng lệnh (có thời gian T_{CY}) BRG sẽ giảm giá trị 2 lần.



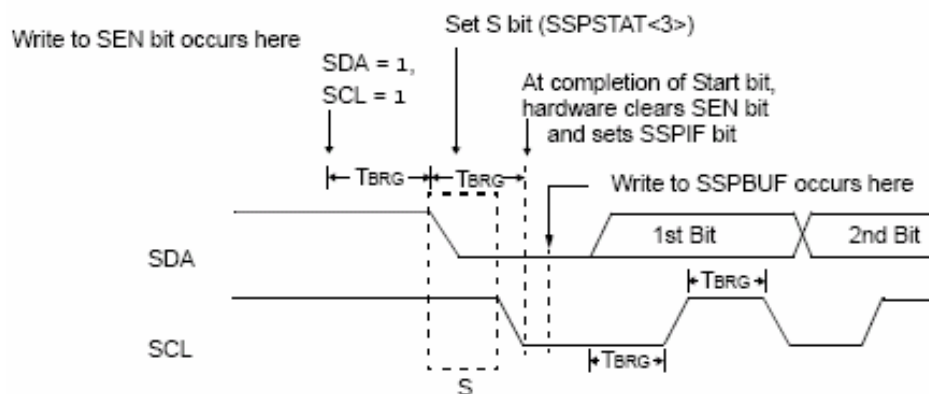
Các giá trị cụ thể của tần số xung nối tiếp do BRG tạo ra được liệt kê trong bảng sau:

Fcy	Fcy*2	BRG Value	FsCL (2 Rollovers of BRG)
10 MHz	20 MHz	19h	400 kHz ⁽¹⁾
10 MHz	20 MHz	20h	312.5 kHz
10 MHz	20 MHz	3Fh	100 kHz
4 MHz	8 MHz	0Ah	400 kHz ⁽¹⁾
4 MHz	8 MHz	0Dh	308 kHz
4 MHz	8 MHz	28h	100 kHz
1 MHz	2 MHz	03h	333 kHz ⁽¹⁾
1 MHz	2 MHz	0Ah	100 kHz
1 MHz	2 MHz	00h	1 MHz ⁽¹⁾

Trong đó giá trị BRG là giá trị được lấy từ 7 bit thấp của thanh ghi SSPADD. Do I2C ở chế độ Master mode, thanh ghi SSPADD sẽ không được sử dụng để chứa địa chỉ, thay vào đó chức năng của SSPADD là thanh ghi chứa giá trị của BRG.

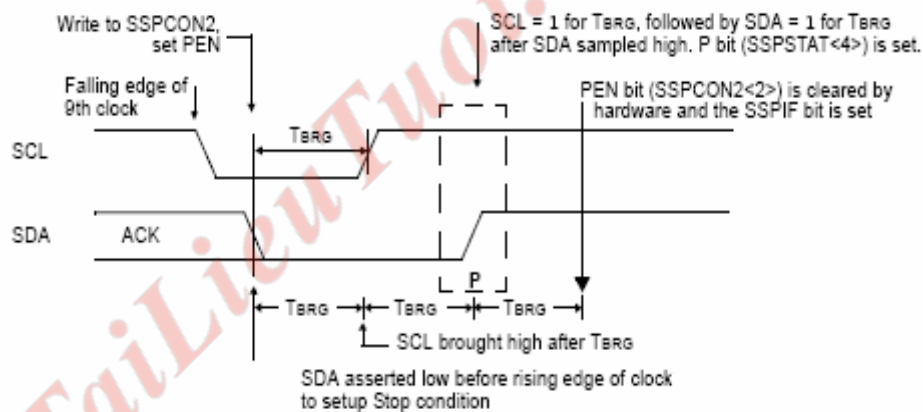
Để tạo được điều kiện Start, trước hết cần đưa hai pin SCL và SDA lên mức logic cao và bit SEN (SSPCON2<0>) phải được set. Khi đó BRG sẽ tự động đọc giá trị 7 bit thấp của thanh ghi SSPADD và bắt đầu đếm. Sau khoảng thời gian T_{BRG} , pin SDA được đưa xuống mức logic thấp. Trạng thái pin SDA ở mức logic thấp và pin SCL ở mức logic cao chính là điều kiện Start của I2C Master mode. Khi đó bit S (SSPSTAT<3>) sẽ được set. Tiếp theo BRG tiếp tục lấy giá trị từ thanh ghi SSPADD để tiếp tục quá trình đếm, bit SEN được tự động xóa và cờ ngắt SSPIF được set.

Trong trường hợp pin SCL và SDA ở trạng thái logic thấp, hoặc là trong quá trình tạo điều kiện Start, pin SCL được đưa về trạng thái logic thấp trước khi pin SDA được đưa về trạng thái logic thấp, điều kiện Start sẽ không được hình thành, cờ ngắt BCLIF sẽ được set và I2C sẽ ở trạng thái tạm ngưng hoạt động (Idle).



Hình 2.33 Giải đồ xung I2C Master mode trong quá trình tạo điều kiện Start.

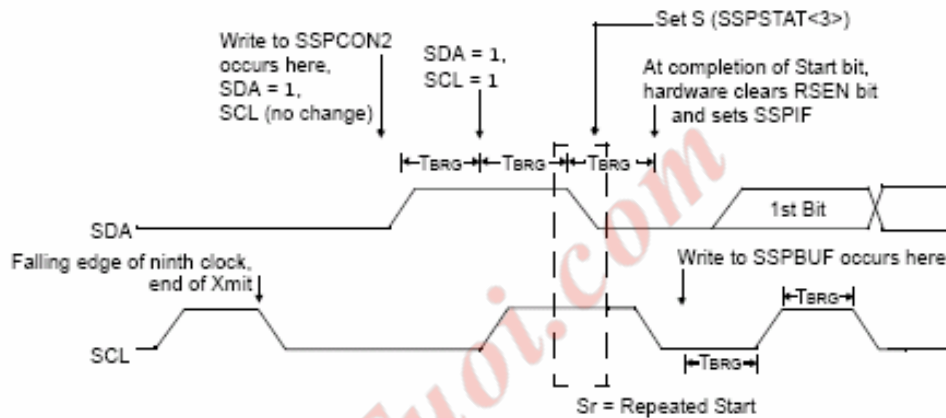
Tín hiệu Stop sẽ được đưa ra pin SDA khi kết thúc dữ liệu bằng cách set bit PEN (SSPCON2<2>). Sau cạnh xuống của xung clock thứ 9 và với tác động của bit điều khiển PEN, pin SDA cũng được đưa xuống mức thấp, BRG lại bắt đầu quá trình đếm. Sau một khoảng thời gian T_{BRG} , pin SCL được đưa lên mức logic cao và sau một khoảng thời gian T_{BRG} nữa pin SDA cũng được đưa lên mức cao. Ngay tại thời điểm đó bit P (SSPSTAT<4>) được set, nghĩa là điều kiện Stop đã được tạo ra. Sau một khoảng thời gian T_{BRG} nữa, bit PEN tự động được xóa và cờ ngắt SSPIF được set.



Hình 2.34 Giải đồ xung I2C Master mode trong quá trình tạo điều kiện Stop.

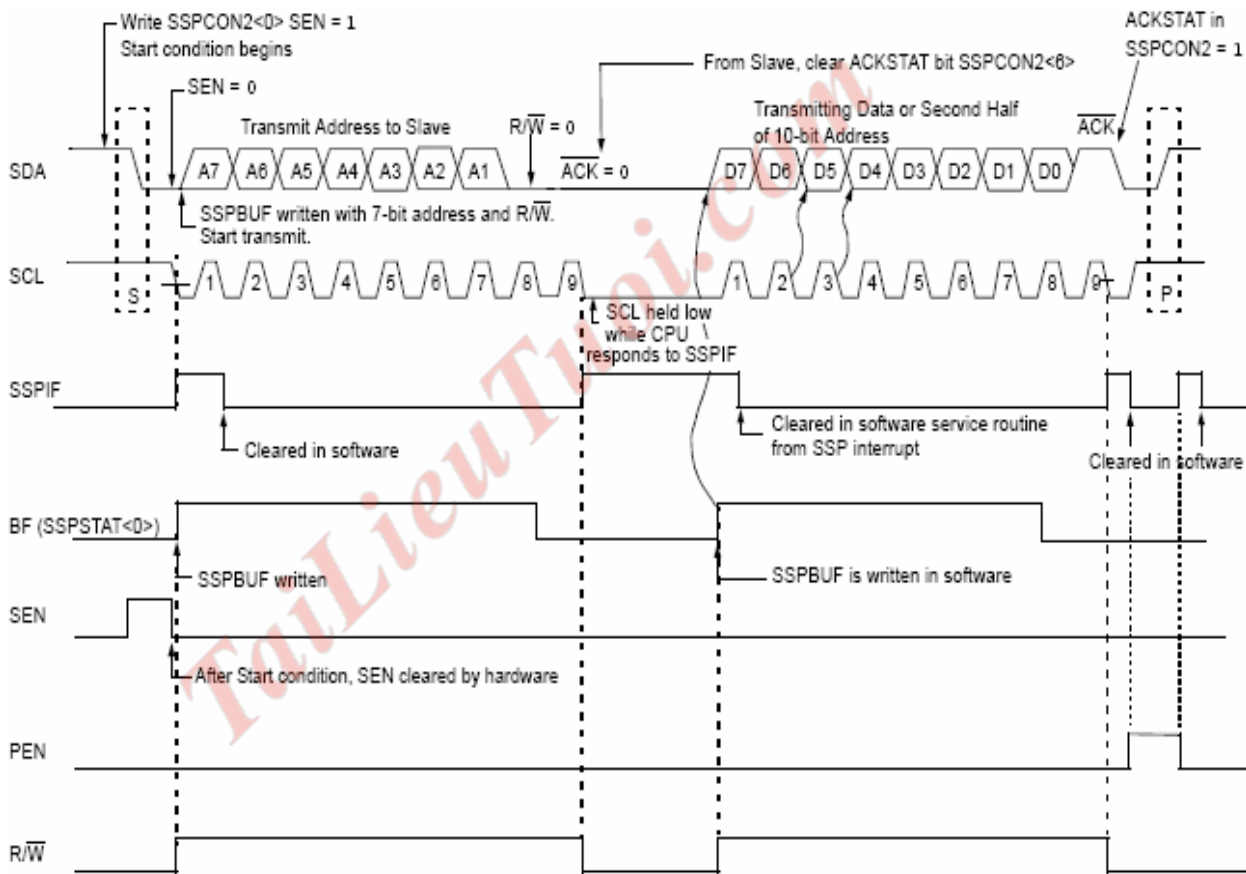
Để tạo được điều kiện Start lặp lại liên tục trong quá trình truyền dữ liệu, trước hết cần set bit RSEN (SSPCON2<1>). Sau khi set bit RSEN, pin SCL được đưa xuống mức logic thấp, pin SDA được đưa lên mức logic cao, BRG lấy giá trị từ thanh ghi SSPADD vào để bắt đầu quá trình đếm. Sau khoảng thời gian T_{BRG} , pin SCL cũng được đưa lên mức logic cao trong khoảng thời gian T_{BRG} tiếp theo. Trong khoảng thời gian T_{BRG} kế tiếp, pin SDA lại được đưa xuống mức logic thấp trong khi SCL vẫn được giữ ở mức logic cao. Ngay thời điểm đó bit S (SSPSTAT<3>) được set để báo hiệu điều kiện Start được hình thành, bit RSEN tự động được xóa và cờ ngắt SSPIF sẽ được set sau một khoảng thời gian T_{BRG} nữa. Lúc này địa chỉ của I2C Slave có thể được đưa vào thanh ghi SSPBUF, sau đó ta chỉ việc đưa tiếp địa chỉ hoặc dữ liệu tiếp theo vào thanh ghi SSPBUF mỗi khi nhận được tín hiệu ACK từ I2C Slave, I2C Master sẽ tự động tạo tín hiệu Start lặp lại liên tục cho quá trình truyền dữ liệu liên tục.

Cần chú ý là bất cứ một trình tự nào sai trong quá trình tạo điều kiện Start lặp lại sẽ làm cho bit BCLIF được set và I2C được đưa về trạng thái "Idle".



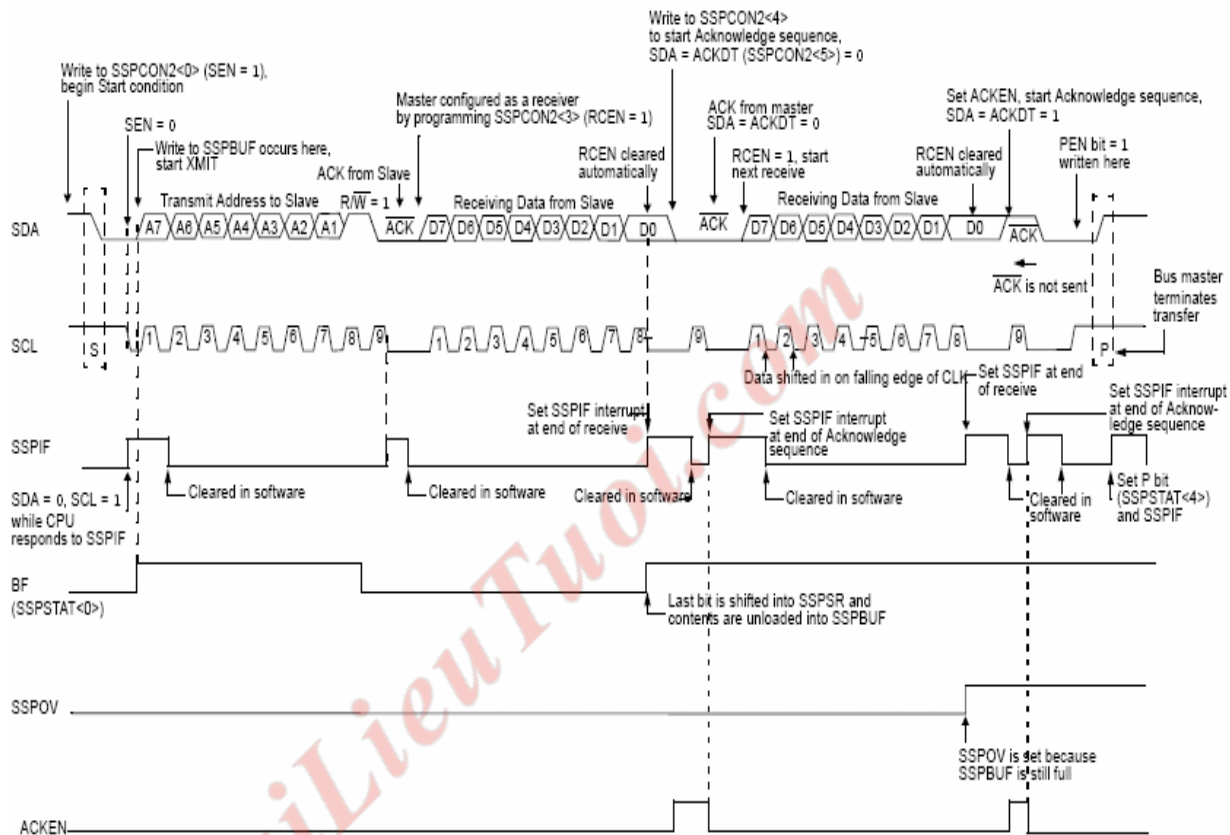
Hình 2.35 Giải đồ xung I2C Master mode trong quá trình tạo điều kiện Start liên tục.

Xét quá trình truyền dữ liệu, xung clock sẽ được đưa ra từ pin SCL và dữ liệu được đưa ra từ pin SDA. Byte dữ liệu đầu tiên phải là byte địa chỉ xác định I2C Slave cần giao tiếp và bit $\overline{R/W}$ (trong trường hợp này $\overline{R/W} = 0$). Đầu tiên các giá trị địa chỉ sẽ được đưa vào thanh ghi SSPBUF, bit BF tự động được set lên 1 và bộ đếm tạo xung clock nối tiếp BRG (Baud Rate Generator) bắt đầu hoạt động. Khi đó từng bit dữ liệu (hoặc địa chỉ và bit $\overline{R/W}$) sẽ được dịch ra ngoài theo từng cạnh xuống của xung clock sau khi cạnh xuống đầu tiên của pin SCL được nhận diện (điều kiện Start), BRG bắt đầu đếm ngược về 0. Khi tất cả các bit của byte dữ liệu được đã được đưa ra ngoài, bộ đếm BRG mang giá trị 0. Sau đó, tại cạnh xuống của xung clock thứ 8, I2C Master sẽ ngưng tác động lên pin SDA để chờ đợi tín hiệu từ I2C Slave (tín hiệu xung \overline{ACK}). Tại cạnh xuống của xung clock thứ 9, I2C Master sẽ lấy mẫu tín hiệu từ pin SDA để kiểm tra xem địa chỉ đã được I2C Slave nhận dạng chưa, trạng thái \overline{ACK} được đưa vào bit ACKSTAT (SSPCON2<6>). Cũng tại thời điểm cạnh xuống của xung clock thứ 9, bit BF được tự động clear, cờ ngắt SSPIF được set và BRG tạm ngưng hoạt động cho tới khi dữ liệu hoặc địa chỉ tiếp theo được đưa vào thanh ghi SSPBUF, dữ liệu hoặc địa chỉ sẽ tiếp tục được truyền đi tại cạnh xuống của xung clock tiếp theo.



Hình 2.36 Giải đồ xung I2C Master mode trong quá trình truyền dữ liệu.

Xét quá trình nhận dữ liệu ở chế độ I2C Master mode. Trước tiên ta cần set bit cho phép nhận dữ liệu RCEN (SSPCON2<3>). Khi đó BRG bắt đầu quá trình đếm, dữ liệu sẽ được dịch vào I2C Master qua pin SDA tại cạnh xuống của pin SCL. Tại cạnh xuống của xung clock thứ 8, bit cờ hiệu cho phép nhận RCEN tự động được xóa, dữ liệu trong thanh ghi SSPSR được đưa vào thanh ghi SSPBUF, cờ hiệu BF được set, cờ ngắt SSPIF được set, BRG ngừng đếm và pin SCL được đưa về mức logic thấp. Khi đó MSSP ở trạng thái tạm ngưng hoạt động để chờ đợi lệnh tiếp theo. Sau khi đọc giá trị thanh ghi SSPBUF, cờ hiệu BF tự động được xóa. Ta còn có thể gửi tín hiệu \overline{ACK} bằng cách set bit ACKEN (SSPCON2<4>).



Hình 2.37 Giải đồ xung I2C Master mode trong quá trình nhận dữ liệu.

2.13 CỔNG GIAO TIẾP SONG SONG PSP (PARALLEL SLAVE PORT)

Ngoài các cổng nối tiếp và các giao diện nối tiếp được trình bày ở phần trên, vi điều khiển PIC16F877A còn được hỗ trợ một cổng giao tiếp song song và chuẩn giao tiếp song song thông qua PORTD và PORTE. Do cổng song song chỉ hoạt động ở chế độ Slave mode nên vi điều khiển khi giao tiếp qua giao diện này sẽ chịu sự điều khiển của thiết bị bên ngoài thông qua các pin của PORTE, trong khi dữ liệu sẽ được đọc hoặc ghi theo dạng bất đồng bộ thông qua 8 pin của PORTD.

Bit điều khiển PSP là PSPMODE (TRISE<4>). PSPMODE được set sẽ thiết lập chức năng các pin của PORTE là các pin cho phép đọc dữ liệu \overline{RD} (RE0/ \overline{RD} /AN5), cho phép ghi dữ liệu \overline{WR} (RE1/ \overline{WR} /AN6) và pin chọn vi điều khiển \overline{CS} (RE2/ \overline{CS} /AN7) phục vụ cho việc truyền nhận dữ liệu song song thông qua bus dữ liệu 8 bit của PORTD. PORTD lúc này đóng vai trò là thanh ghi chốt dữ liệu 8 bit, đồng thời tác động của thanh ghi TRISD cũng sẽ được bỏ qua do PORTD lúc này chịu sự điều khiển của các thiết bị bên ngoài. PORTE vẫn chịu sự tác động của thanh ghi TRISE, do đó cần xác lập trạng thái các pin PORTE là input bằng cách set các bit TRISE<2:0>. Ngoài ra cần đưa giá trị thích hợp các bit PCFG3:PCFG0 (thanh ghi

ADCON1<3:0>) để ấn định các pin của PORTE là các pin I/O dạng digital (PORTE còn là các pin chức năng của khối ADC).

Khi các pin \overline{CS} và \overline{WR} cùng ở mức thấp, dữ liệu từ bên ngoài sẽ được ghi lên PORTD. Khi một trong hai pin trên chuyển lên mức logic cao, cờ hiệu báo dữ liệu trong buffer đã đầy BIF (TRISE<7>) được set và cờ ngắt PSPIF (PIR1<7>) được set để báo hiệu kết thúc ghi dữ liệu. Bit BIF chỉ được xóa về 0 khi dữ liệu vừa nhận được ở PORTD được đọc vào. Bit báo hiệu dữ liệu nhận được trong buffer bị tràn IBOV (TRISE<5>) sẽ được set khi vi điều khiển nhận tiếp dữ liệu tiếp theo trong khi chưa đọc vào dữ liệu đã nhận được trước đó.

Khi các pin \overline{CS} và \overline{RD} cùng ở mức logic thấp, bit báo hiệu buffer truyền dữ liệu đã đầy BOF (TRISE<6>) sẽ được xóa ngay lập tức để báo hiệu PORTD đã sẵn sàng cho quá trình đọc dữ liệu. Khi một trong hai pin trên chuyển sang mức logic cao, cờ ngắt PSPIF

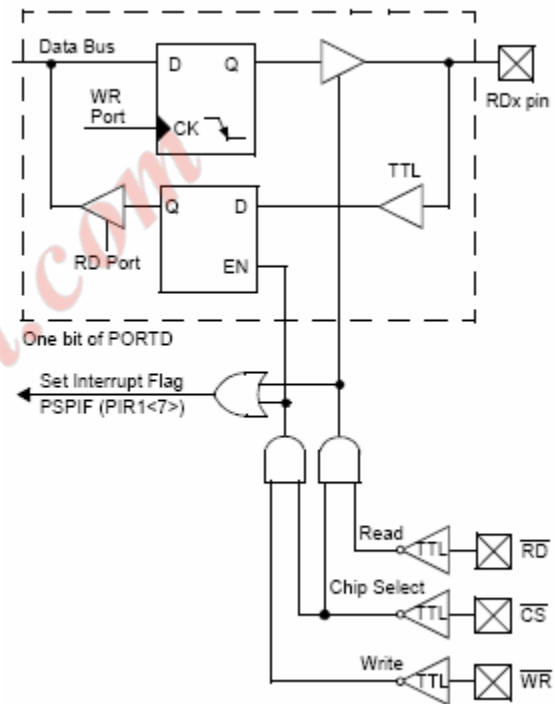
sẽ được set để báo hiệu quá trình đọc dữ liệu hoàn tất. Bit BOF vẫn được giữ ở mức logic 0 cho đến khi dữ liệu tiếp theo được đưa vào PORTD.

Cần chú ý là ngắt SSPIF được điều khiển bởi bit PSPIE (PIE1<7>) và phải được xóa bằng chương trình.

Các thanh ghi liên quan đến PSP bao gồm:

- Thanh ghi PORTD (địa chỉ 08h): chứa dữ liệu cần đọc hoặc ghi.
- Thanh ghi PORTE (địa chỉ 09h): chứa giá trị các pin PORTE.
- Thanh ghi TRISE (địa chỉ 89h): chứa các bit điều khiển PORTE và PSP.
- Thanh ghi PIR1 (địa chỉ 0Ch): chứa cờ ngắt PSPIF.
- Thanh ghi PIE1 (địa chỉ 8Ch): chứa bit cho phép ngắt PSP.
- Thanh ghi ADCON1 (địa chỉ 9Fh): điều khiển khối ADC tại PORTE.

Chi tiết về các thanh ghi sẽ được trình bày cụ thể ở phụ lục 2.



Hình 2.38 Sơ đồ khối của PORTD và PORTE khi hoạt động ở chế độ PSP Slave mode.

2.14 TỔNG QUAN VỀ MỘT SỐ ĐẶC TÍNH CỦA CPU.

2.14.1 CONFIGURATION BIT

Đây là các bit dùng để lựa chọn các đặc tính của CPU. Các bit này được chứa trong bộ nhớ chương trình tại địa chỉ 2007h và chỉ có thể được truy xuất trong quá trình lập trình cho vi điều khiển. Chi tiết về các bit này như sau:

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	PWRTEN	WDTEN	Fosc1	Fosc0
bit 13													bit0

Bit 13 CP: (Code Protection)

1: tắt chế độ bảo vệ mã chương trình.

0: bật chế độ bảo vệ mã chương trình.

Bit 12, 5, 4: không quan tâm và được mặc định mang giá trị 0.

Bit 11 DEBUG (In-circuit debug mode bit)

1: không cho phép, RB7 và RB6 được xem như các pin xuất nhập bình thường.

0: cho phép, RB7 và RB6 là các pin được sử dụng cho quá trình debug.

Bit 10-9 WRT1:WRT0 Flash Program Memory Write Enable bit

11: Tắt chức năng chống ghi, EECON sẽ điều khiển quá trình ghi lên toàn bộ nhớ chương trình.

10: chỉ chống từ địa chỉ 0000h:00FFh.

01: chỉ chống ghi từ địa chỉ 0000h:07FFh.

00: chỉ chống ghi từ địa chỉ 0000h:0FFFh.

Bit 8 CPD Data EEPROM Memory Write Protection bit

1: Tắt chức năng bảo vệ mã của EEPROM.

0: Bật chức năng bảo vệ mã.

Bit 7 LVP Low-Voltage (Single supply) In-Circuit Serial Programming Enable bit

1: Cho phép chế độ nạp điện áp thấp, pin RB3/PGM được sử dụng cho chế độ này.

0: Không cho phép chế độ nạp điện áp thấp, điện áp cao được đưa vào từ pin $\overline{\text{MCLR}}$, pin RB3 là pin I/O bình thường.

Bit 6 BODEN Brown-out Reset Enable bit

1: cho phép BOR (Brown-out Reset)

0: không cho phép BOR.

Bit 3 $\overline{\text{PWRTEN}}$ Power-up Timer Enable bit

1: không cho phép PWR.

0: cho phép PWR.

Bit 2 WDTEN Watchdog Timer Enable bit

1: cho phép WDT.

0: không cho phép WDT.

Bit 1-0 Fosc1:Fosc0 lựa chọn loại oscillator

11: sử dụng RC oscillator.

10: sử dụng HS oscillator.

01: sử dụng XT oscillator.

00: sử dụng LP oscillator.

Chi tiết về các đặc tính sẽ được đề cập cụ thể trong các phần tiếp theo.

2.14.2 CÁC ĐẶC TÍNH CỦA OSCILLATOR

PIC16F877A có khả năng sử dụng một trong 4 loại oscillator, đó là:

LP: (Low Power Crystal).

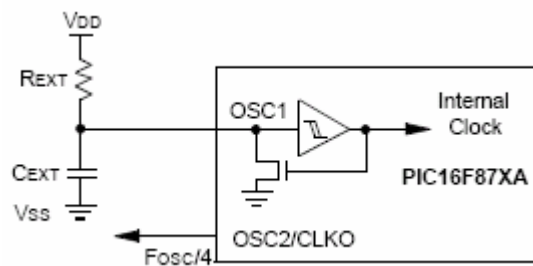
XT: Thạch anh bình thường.

HS: (High-Speed Crystal).

RC: (Resistor/Capacitor) dao động do mạch RC tạo ra.

Đối với các loại oscillator LP, HS, XT, oscillator được gắn vào vi điều khiển thông qua các pin OSC1/CLKI và OSC2/CLKO.

Đối với các ứng dụng không cần các loại oscillator tốc độ cao, ta có thể sử dụng mạch dao động RC làm nguồn cung cấp xung hoạt động cho vi điều khiển. Tần số tạo ra phụ thuộc vào các giá trị điện áp, giá trị điện trở và tụ điện, bên cạnh đó là sự ảnh hưởng của các yếu tố như nhiệt độ, chất lượng của các linh kiện.



Hình 2.39 RC oscillator.

Các linh kiện sử dụng cho mạch RC oscillator phải bảo đảm các giá trị sau:

$$3\text{ K} < R_{\text{EXT}} < 100\text{ K}$$

$$C_{\text{EXT}} > 20\text{ pF}$$

2.14.3 CÁC CHẾ ĐỘ RESET

Có nhiều chế độ reset vi điều khiển, bao gồm:

Power-on Reset POR (Reset khi cấp nguồn hoạt động cho vi điều khiển).

$\overline{\text{MCLR}}$ reset trong quá trình hoạt động.

$\overline{\text{MCLR}}$ từ chế độ sleep.

WDT reset (reset do khối WDT tạo ra trong quá trình hoạt động).

WDT wake up từ chế độ sleep.

Brown-out reset (BOR).

Ngoại trừ reset POR trạng thái các thanh ghi là không xác định và WDT wake up không ảnh hưởng đến trạng thái các thanh ghi, các chế độ reset còn lại đều đưa giá trị các thanh ghi về giá trị ban đầu được ấn định sẵn. Các bit \overline{TO} và \overline{PD} chỉ thị trạng thái hoạt động, trạng thái reset của vi điều khiển và được điều khiển bởi CPU.

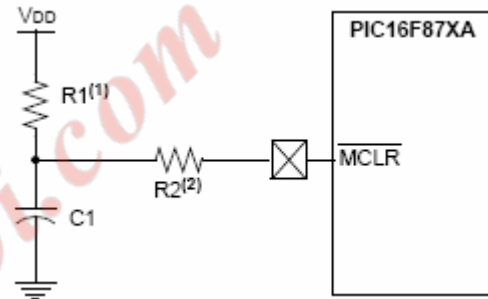
\overline{MCLR} reset: Khi pin \overline{MCLR} ở mức logic thấp, vi điều khiển sẽ được reset. Tín hiệu reset được cung cấp bởi một mạch ngoại vi với các yêu cầu cụ thể sau:

Không nối pin \overline{MCLR} trực tiếp lên nguồn V_{DD} .

$R1$ phải nhỏ hơn 40 K để đảm bảo các đặc tính điện của vi điều khiển.

$R2$ phải lớn hơn 1 K để hạn dòng đi vào vi điều khiển.

\overline{MCLR} reset còn được chống nhiễu bởi một bộ lọc để tránh các tín hiệu nhiễu tác động lên pin \overline{MCLR} .



Hình 2.40 Mạch reset qua pin \overline{MCLR} .

Power-on reset (POR): Đây là xung reset do vi điều khiển tạo ra khi phát hiện nguồn cung cấp V_{DD} . Khi hoạt động ở chế độ bình thường, vi điều khiển cần được đảm bảo các thông số về dòng điện, điện áp để hoạt động bình thường. Nhưng nếu các tham số này không được đảm bảo, xung reset do POR tạo ra sẽ đưa vi điều khiển về trạng thái reset và chỉ tiếp tục hoạt động khi nào các tham số trên được đảm bảo.

Power-up Timer (PWRT): đây là bộ định thời hoạt động dựa vào mạch RC bên trong vi điều khiển. Khi PWRT được kích hoạt, vi điều khiển sẽ được đưa về trạng thái reset. PWRT sẽ tạo ra một khoảng thời gian delay (khoảng 72 ms) để V_{DD} tăng đến giá trị thích hợp.

Oscillator Start-up Timer (OST): OST cung cấp một khoảng thời gian delay bằng 1024 chu kỳ xung của oscillator sau khi PWRT ngừng tác động (vi điều khiển đã đủ điều kiện hoạt động) để đảm bảo sự ổn định của xung do oscillator phát ra. Tác động của OST còn xảy ra đối với POR reset và khi vi điều khiển được đánh thức từ chế độ sleep. OST chỉ tác động đối với các loại oscillator là XT, HS và LP.

Brown-out reset (BOR): Nếu V_{DD} hạ xuống thấp hơn giá trị V_{BOR} (khoảng 4V) và kéo dài trong khoảng thời gian lớn hơn T_{BOR} (khoảng 100 us), BOR được kích hoạt và vi điều khiển được đưa về trạng thái BOR reset. Nếu điện áp cung cấp cho vi điều khiển hạ xuống thấp hơn V_{BOR} trong khoảng thời gian ngắn hơn T_{BOR} , vi điều khiển sẽ không được reset. Khi điện áp cung cấp đủ cho vi điều khiển hoạt động, PWRT được kích hoạt để tạo ra một khoảng thời gian delay (khoảng 72ms). Nếu trong khoảng thời gian này điện áp cung cấp cho

vi điều khiển lại tiếp tục hạ xuống dưới mức điện áp V_{BOR} , BOR reset sẽ lại được kích hoạt khi vi điều khiển đủ điện áp hoạt động. Một điểm cần chú ý là khi BOR reset được cho phép, PWRT cũng sẽ hoạt động bất chấp trạng thái của bit PWRT.

Tóm lại để vi điều khiển hoạt động được từ khi cấp nguồn cần trải qua các bước sau:

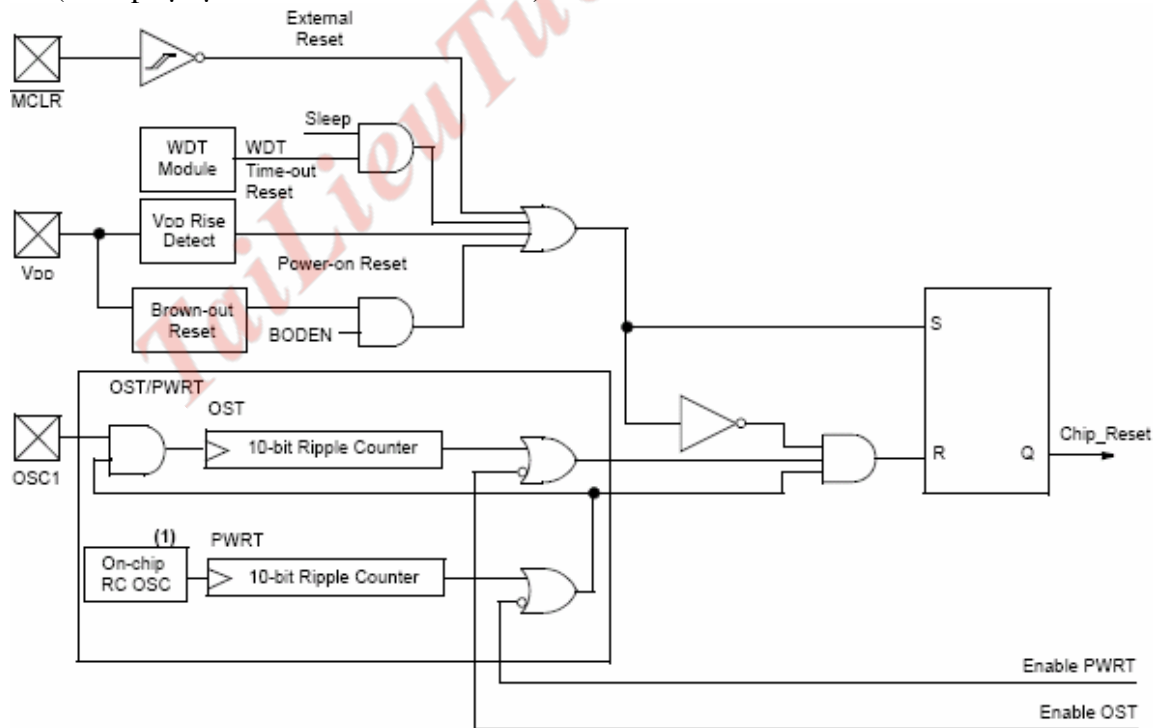
POR tác động.

PWRT (nếu được cho phép hoạt động) tạo ra khoảng thời gian delay T_{PWRT} để ổn định nguồn cung cấp.

OST (nếu được cho phép) tạo ra khoảng thời gian delay bằng 1024 chu kì xung của oscillator để ổn định tần số của oscillator.

Đến thời điểm này vi điều khiển mới bắt đầu hoạt động bình thường.

Thanh ghi điều khiển và chỉ thị trạng thái nguồn cung cấp cho vi điều khiển là thanh ghi PCON (xem phụ lục 2 để biết thêm chi tiết).



Hình 2.41 Sơ đồ các chế độ reset của PIC16F877A.

2.14.4 NGẮT (INTERRUPT)

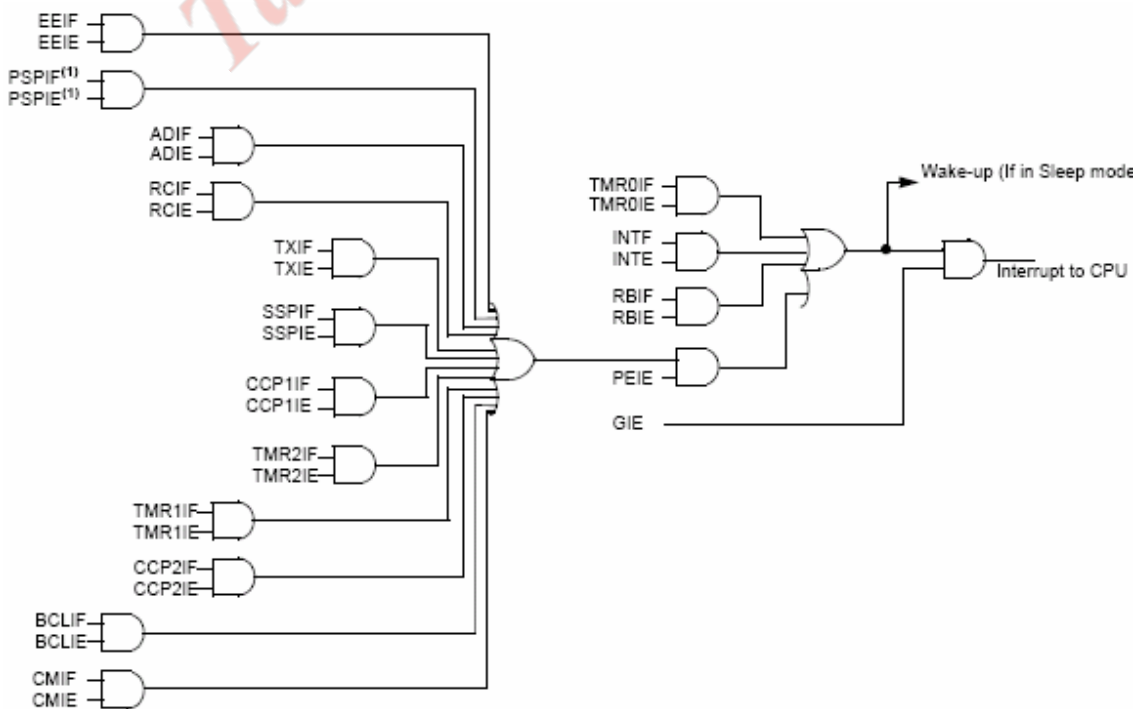
PIC16F877A có đến 15 nguồn tạo ra hoạt động ngắt được điều khiển bởi thanh ghi INTCON (bit GIE). Bên cạnh đó mỗi ngắt còn có một bit điều khiển và cờ ngắt riêng. Các cờ ngắt vẫn được set bình thường khi thỏa mãn điều kiện ngắt xảy ra bất chấp trạng thái của bit GIE, tuy nhiên hoạt động ngắt vẫn phụ thuộc vào bit GIE và các bit điều khiển khác. Bit điều khiển ngắt RB0/INT và TMR0 nằm trong thanh ghi INTCON, thanh ghi này còn chứa bit cho

phép các ngắt ngoại vi PEIE. Bit điều khiển các ngắt nằm trong thanh ghi PIE1 và PIE2. Cờ ngắt của các ngắt nằm trong thanh ghi PIR1 và PIR2.

Trong một thời điểm chỉ có một chương trình ngắt được thực thi, chương trình ngắt được kết thúc bằng lệnh RETFIE. Khi chương trình ngắt được thực thi, bit GIE tự động được xóa, địa chỉ lệnh tiếp theo của chương trình chính được cất vào trong bộ nhớ Stack và bộ đếm chương trình sẽ chỉ đến địa chỉ 0004h. Lệnh RETFIE được dùng để thoát khỏi chương trình ngắt và quay trở về chương trình chính, đồng thời bit GIE cũng sẽ được set để cho phép các ngắt hoạt động trở lại. Các cờ hiệu được dùng để kiểm tra ngắt nào đang xảy ra và phải được xóa bằng chương trình trước khi cho phép ngắt tiếp tục hoạt động trở lại để ta có thể phát hiện được thời điểm tiếp theo mà ngắt xảy ra.

Đối với các ngắt ngoại vi như ngắt từ chân INT hay ngắt từ sự thay đổi trạng thái các pin của PORTB (PORTB Interrupt on change), việc xác định ngắt nào xảy ra cần 3 hoặc 4 chu kỳ lệnh tùy thuộc vào thời điểm xảy ra ngắt.

Cần chú ý là trong quá trình thực thi ngắt, chỉ có giá trị của bộ đếm chương trình được cất vào trong Stack, trong khi một số thanh ghi quan trọng sẽ không được cất và có thể bị thay đổi giá trị trong quá trình thực thi chương trình ngắt. Điều này nên được xử lý bằng chương trình để tránh hiện tượng trên xảy ra.



Hình 2.42 Sơ đồ logic của tất cả các ngắt trong vi điều khiển PIC16F877A.

2.14.4.1 NGẮT INT

Ngắt này dựa trên sự thay đổi trạng thái của pin RB0/INT. Cạnh tác động gây ra ngắt có thể là cạnh lên hay cạnh xuống và được điều khiển bởi bit INTEDG (thanh ghi OPTION_REG <6>). Khi có cạnh tác động thích hợp xuất hiện tại pin RB0/INT, cờ ngắt INTF được set bất chấp trạng thái các bit điều khiển GIE và PEIE. Ngắt này có khả năng đánh thức vi điều khiển từ chế độ sleep nếu bit cho phép ngắt được set trước khi lệnh SLEEP được thực thi.

2.14.4.2 NGẮT DO SỰ THAY ĐỔI TRẠNG THÁI CÁC PIN TRONG PORTB

Các pin PORTB<7:4> được dùng cho ngắt này và được điều khiển bởi bit RBIE (thanh ghi INTCON<4>). Cờ ngắt của ngắt này là bit RBIF (INTCON<0>).

2.14.5 WATCHDOG TIMER (WDT)

Watchdog timer (WDT) là bộ đếm độc lập dùng nguồn xung đếm từ bộ tạo xung được tích hợp sẵn trong vi điều khiển và không phụ thuộc vào bất kì nguồn xung clock ngoại vi nào. Điều đó có nghĩa là WDT vẫn hoạt động ngay cả khi xung clock được lấy từ pin OSC1/CLKI và pin OSC2/CLKO của vi điều khiển ngừng hoạt động (chẳng hạn như do tác động của lệnh sleep). Bit điều khiển của WDT là bit WDTE nằm trong bộ nhớ chương trình ở địa chỉ 2007h (Configuration bit).

WDT sẽ tự động reset vi điều khiển (Watchdog Timer Reset) khi bộ đếm của WDT bị tràn (nếu WDT được cho phép hoạt động), đồng thời bit \overline{TO} tự động được xóa. Nếu vi điều khiển đang ở chế độ sleep thì WDT sẽ đánh thức vi điều khiển (Watchdog Timer Wake-up) khi bộ đếm bị tràn. Như vậy WDT có tác dụng reset vi điều khiển ở thời điểm cần thiết mà không cần đến sự tác động từ bên ngoài, chẳng hạn như trong quá trình thực thi lệnh, vi điều khiển bị “kẹt” ở một chỗ nào đó mà không thoát ra được, khi đó vi điều khiển sẽ tự động được reset khi WDT bị tràn để chương trình hoạt động đúng trở lại. Tuy nhiên khi sử dụng WDT cũng có sự phiền toái vì vi điều khiển sẽ thường xuyên được reset sau một thời gian nhất định, do đó cần tính toán thời gian thích hợp để xóa WDT (dùng lệnh CLRWDT). Và để việc ấn định thời gian reset được linh động, WDT còn được hỗ trợ một bộ chia tần số prescaler được điều khiển bởi thanh ghi OPTION_REG (prescaler này được chia xẻ với Timer0).

Một điểm cần chú ý nữa là lệnh sleep sẽ xóa bộ đếm WDT và prescaler. Ngoài ra lệnh xóa CLRWDT chỉ xóa bộ đếm chứ không làm thay đổi đối tượng tác động của prescaler (WDT hay Timer0).

Xem lại Timer0 và thanh ghi OPTION_REG (phụ lục 2) để biết thêm chi tiết.

2.14.6 CHẾ ĐỘ SLEEP

Đây là chế độ hoạt động của vi điều khiển khi lệnh SLEEP được thực thi. Khi đó nếu được cho phép hoạt động, bộ đếm của WDT sẽ bị xóa nhưng WDT vẫn tiếp tục hoạt động, bit \overline{PD} (STATUS<3>) được reset về 0, bit \overline{TO} được set, oscillator ngừng tác động và các PORT giữ nguyên trạng thái như trước khi lệnh SLEEP được thực thi.

Do khi ở chế độ SLEEP, dòng cung cấp cho vi điều khiển là rất nhỏ nên ta cần thực hiện các bước sau trước khi vi điều khiển thực thi lệnh SLEEP:

Đưa tất cả các pin về trạng thái V_{DD} hoặc V_{SS}

Cần bảo đảm rằng không có mạch ngoại vi nào được điều khiển bởi dòng điện của vi điều khiển vì dòng điện nhỏ không đủ khả năng cung cấp cho các mạch ngoại vi hoạt động.

Tạm ngưng hoạt động của khối A/D và không cho phép các xung clock từ bên ngoài tác động vào vi điều khiển.

Để ý đến chức năng kéo lên điện trở ở PORTB.

Pin \overline{MCLR} phải ở mức logic cao.

2.14.6.1 “ĐÁNH THỨC” VI ĐIỀU KHIỂN

Vi điều khiển có thể được “đánh thức” dưới tác động của một trong số các hiện tượng sau:

1. Tác động của reset ngoại vi thông qua pin \overline{MCLR} .
2. Tác động của WDT khi bị tràn.
3. Tác động từ các ngắt ngoại vi từ PORTB (PORTB Interrupt on change hoặc pin INT).

Các bit \overline{PD} và \overline{TO} được dùng để thể hiện trạng thái của vi điều khiển và để phát hiện nguồn tác động làm reset vi điều khiển. Bit \overline{PD} được set khi vi điều khiển được cấp nguồn và được reset về 0 khi vi điều khiển ở chế độ sleep. Bit \overline{TO} được reset về 0 khi WDT tác động do bộ đếm bị tràn.

Ngoài ra còn có một số nguồn tác động khác từ các chức năng ngoại vi bao gồm:

1. Đọc hay ghi dữ liệu thông qua PSP (Parallel Slave Port).
2. Ngắt Timer1 khi hoạt động ở chế độ đếm bất đồng bộ.
3. Ngắt CCP khi hoạt động ở chế độ Capture.
4. Các hiện tượng đặc biệt làm reset Timer1 khi hoạt động ở chế độ đếm bất đồng bộ dùng nguồn xung clock ở bên ngoài).
5. Ngắt SSP khi bit Start/Stop được phát hiện.
6. SSP hoạt động ở chế độ Slave mode khi truyền hoặc nhận dữ liệu.
7. Tác động của USART từ các pin RX hay TX khi hoạt động ở chế độ Slave mode đồng bộ.
8. Khối chuyển đổi A/D khi nguồn xung clock hoạt động ở dạng RC.
9. Hoàn tất quá trình ghi vào EEPROM.
10. Ngõ ra bộ so sánh thay đổi trạng thái.

Các tác động ngoại vi khác không có tác dụng đánh thức vi điều khiển vì khi ở chế độ sleep các xung clock cung cấp cho vi điều khiển ngừng hoạt động. Bên cạnh đó cần cho phép các ngắt hoạt động trước khi lệnh SLEEP được thực thi để bảo đảm tác động của các ngắt. Việc đánh thức vi điều khiển từ các ngắt vẫn được thực thi bất chấp trạng thái của bit GIE. Nếu bit GIE mang giá trị 0, vi điều khiển sẽ thực thi lệnh tiếp theo sau lệnh SLEEP của chương trình (vì chương trình ngắt không được cho phép thực thi). Nếu bit GIE được set trước khi lệnh SLEEP được thực thi, vi điều khiển sẽ thực thi lệnh tiếp theo của chương trình và sau đó nhảy tới địa chỉ chứa chương trình ngắt (0004h). Trong trường hợp lệnh tiếp theo không đóng vai trò quan trọng trong chương trình, ta cần đặt thêm lệnh NOP sau lệnh SLEEP để bỏ qua tác động của lệnh này, đồng thời giúp ta dễ dàng hơn trong việc kiểm soát hoạt động của chương trình ngắt. Tuy nhiên cũng có một số điểm cần lưu ý như sau:

Nếu ngắt xảy ra trước khi lệnh SLEEP được thực thi, lệnh SLEEP sẽ không được thực thi và thay vào đó là lệnh NOP, đồng thời các tác động của lệnh SLEEP cũng sẽ được bỏ qua.

Nếu ngắt xảy ra trong khi hay sau khi lệnh SLEEP được thực thi, vi điều khiển lập tức được đánh thức từ chế độ sleep, và lệnh SLEEP sẽ được thực thi ngay sau khi vi điều khiển được đánh thức.

Để kiểm tra xem lệnh SLEEP đã được thực thi hay chưa, ta kiểm tra bit \overline{PD} . Nếu bit \overline{PD} vẫn mang giá trị 1 tức là lệnh SLEEP đã không được thực thi và thay vào đó là lệnh NOP. Bên cạnh đó ta cần xóa WDT để chắc chắn rằng WDT đã được xóa trước khi thực thi lệnh SLEEP, qua đó cho phép ta xác định được thời điểm vi điều khiển được đánh thức do tác động của WDT.

CHƯƠNG 3 TẬP LỆNH CỦA VI ĐIỀU KHIỂN PIC

3.1 VÀI NÉT SƠ LƯỢC VỀ TẬP LỆNH CỦA VI ĐIỀU KHIỂN PIC

Như đã trình bày ở chương 1, PIC là vi điều khiển có tập lệnh rút gọn RISC (Reduced Instruction Set Computer), bao gồm 35 lệnh và có thể được phân ra thành 3 nhóm cơ bản:

Nhóm lệnh thao tác trên bit.

Nhóm lệnh thao tác trên byte.

Nhóm lệnh điều khiển.

Đối với dòng vi điều khiển PIC16Fxxx, mỗi lệnh được mã hóa thành 14 bit word, bao gồm các bit opcode (dùng để xác định lệnh nào được mã hóa) và các bit mô tả một hay vài tham số của lệnh.

Đối với nhóm lệnh thao tác trên byte, ta có 2 tham số f (xác định địa chỉ byte cần thao tác) và d (xác định nơi chứa kết quả thực thi lệnh). Nếu d = 0, kết quả sẽ được đưa vào thanh ghi W. Nếu d = 1, kết quả được đưa vào thanh ghi được mô tả bởi tham số f.

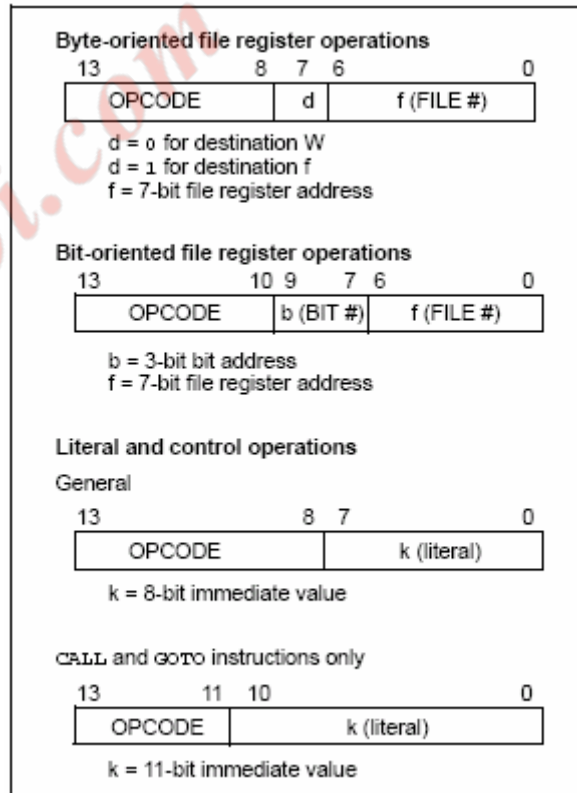
Đối với nhóm lệnh thao tác trên bit, ta có hai tham số b (xác định bit cần thao tác) và f (xác định địa chỉ byte dữ liệu cần thao tác).

Đối với nhóm lệnh điều khiển chỉ có một tham số duy nhất là k (k có thể là 8 bit trong trường hợp các lệnh bình thường hay 11 bit trong trường hợp là lệnh CALL và lệnh GOTO) dùng để mô tả đối tượng tác động của vi điều khiển (một label, một hằng số nào đó).

Mỗi lệnh sẽ được vi điều khiển thực thi xong trong vòng một chu kỳ lệnh, ngoại trừ các lệnh làm thay đổi giá trị bộ đếm chương trình PC cần 2 chu kỳ lệnh. Một chu kỳ lệnh gồm 4 xung clock của oscillator. Ví dụ ta sử dụng oscillator có tần số 4 MHz thì tần số thực thi lệnh sẽ là $4\text{MHz}/4 = 1\text{ MHz}$, như vậy một chu kỳ lệnh có thời gian 1 μs .

Các lệnh thao tác trên một thanh ghi bất kỳ đều thực hiện cơ chế Read-Modify-Write, tức là thanh ghi sẽ được đọc, dữ liệu được thao tác và kết quả được đưa vào thanh ghi chứa kết quả (nơi chứa kết quả tùy thuộc vào lệnh thực thi và tham số d). Ví dụ như khi thực thi lệnh “CLRF PORTB”, vi điều khiển sẽ đọc giá trị thanh ghi PORTB, xóa tất cả các bit và ghi kết quả trở lại thanh ghi PORTB.

Sau đây ta sẽ đi sâu vào cấu trúc, cú pháp và tác động cụ thể của từng lệnh.



Hình 3.1 Cơ chế mã hóa lệnh của PIC16Fxxx.

3.2 TẬP LỆNH CỦA VI ĐIỀU KHIỂN PIC

3.2.1 Lệnh ADDLW

Cú pháp: ADDLW k ($0 \leq k \leq 255$)
Tác dụng: cộng giá trị k vào thanh ghi W, kết quả được chứa trong thanh ghi W.
Bit trạng thái: C, DC, Z

3.2.2 Lệnh ADDWF

Cú pháp: ADDWF f, d
($0 \leq f \leq 255, d \in [0, 1]$).
Tác dụng: cộng giá trị hai thanh ghi W và thanh ghi f . Kết quả được chứa trong thanh ghi W nếu $d = 0$ hoặc thanh ghi f nếu $d = 1$.
Bit trạng thái: C, DC, Z

3.2.3 Lệnh ANDLW

Cú pháp: ANDLW k ($0 \leq k \leq 255$)
Tác dụng: thực hiện phép toán AND giữa thanh ghi W và giá trị k , kết quả được chứa trong thanh ghi W.
Bit trạng thái: Z

3.2.4 Lệnh ANDWF

Cú pháp: ANDWF f, d
($0 \leq f \leq 127, d \in [0, 1]$).
Tác dụng: thực hiện phép toán AND giữa các giá trị chứa trong hai thanh ghi W và f . Kết quả được đưa vào thanh ghi W nếu $d=0$ hoặc thanh ghi f nếu $d = 1$.
Bit trạng thái: Z

3.2.5 Lệnh BCF

Cú pháp: BCF f, b ($0 \leq f \leq 127, 0 \leq b \leq 7$)
Tác dụng: xóa bit b trong thanh ghi f về giá trị 0.
Bit trạng thái: không có.

3.2.6 Lệnh BSF

Cú pháp: BSF f, b
($0 \leq f \leq 127, 0 \leq b \leq 7$)
Tác dụng: set bit b trong trnh ghi f .
Bit trạng thái: không có

3.2.7 Lệnh BTFSS

Cú pháp: BTFSS f, b
($0 \leq f \leq 127, 0 \leq b \leq 7$)
Tác dụng: kiểm tra bit b trong thanh ghi f . Nếu bit b bằng 0, lệnh tiếp theo được thực thi. Nếu bit b bằng 1, lệnh tiếp theo được bỏ qua và thay vào đó là lệnh NOP.
Bit trạng thái: không có

3.2.8 Lệnh BTFSC

Cú pháp: BTFSC f, b
($0 \leq f \leq 127, 0 \leq b \leq 7$)
Tác dụng: kiểm tra bit b trong thanh ghi f . Nếu bit b bằng 1, lệnh tiếp theo được thực thi. Nếu bit b bằng 0, lệnh tiếp theo được bỏ qua và thay vào đó là lệnh NOP.
Bit trạng thái: không có

3.2.9 Lệnh CALL

Cú pháp: CALL k ($0 \leq k \leq 2047$)

Tác dụng: gọi một chương trình con. Trước hết địa chỉ quay trở về từ chương trình con (PC+1) được cất vào trong Stack, giá trị địa chỉ mới được đưa vào bộ đếm gồm 11 bit của biến k và 2 bit PCLATH<4:3>.

Bit trạng thái: không có

3.2.10 Lệnh CLRF

Cú pháp CLRF f ($0 \leq f \leq 127$)

Tác dụng: xóa thanh ghi f và bit Z được set.

Bit trạng thái: Z

3.2.11 Lệnh CLRW

Cú pháp CLRW

Tác dụng: xóa thanh ghi W và bit Z được set.

Bit trạng thái: Z

3.2.12 Lệnh CLRWDT

Cú pháp: CLRWDT

Tác dụng: reset Watchdog Timer, đồng thời prescaler cũng được reset, các bit \overline{PD} và \overline{TO} được set lên 1.

Bit trạng thái: \overline{TO} , \overline{PD}

3.2.13 Lệnh COMF

Cú pháp: COMF f,d

($0 \leq f \leq 127, d \in [0,1]$).

Tác dụng: đảo các bit trong thanh ghi f. Kết quả được đưa vào thanh ghi W nếu d=0 hoặc thanh ghi f nếu d=1.

Bit trạng thái: Z

3.2.14 Lệnh DECF

Cú pháp: DECF f,d

($0 \leq f \leq 127, d \in [0,1]$).

Tác dụng: giá trị thanh ghi f được giảm đi 1 đơn vị. Kết quả được đưa vào thanh ghi W nếu d = 0 hoặc thanh ghi f nếu d = 1.

Bit trạng thái: Z

3.2.15 Lệnh DECFSZ

Cú pháp: DECFSZ f,d

($0 \leq f \leq 127, d \in [0,1]$)

Tác dụng: giá trị thanh ghi f được giảm 1 đơn vị. Nếu kết quả sau khi giảm khác 0, lệnh tiếp theo được thực thi, nếu kết quả bằng 0, lệnh tiếp theo không được thực thi và thay vào đó là lệnh NOP. Kết quả được đưa vào thanh ghi W nếu d = 0 hoặc thanh ghi f nếu d = 1.

Bit trạng thái: không có

3.2.16 Lệnh GOTO

Cú pháp: GOTO k ($0 \leq k \leq 2047$)

Tác dụng: nhảy tới một label được định nghĩa bởi tham số k và 2 bit PCLATH<4:3>.

Bit trạng thái: không có.

3.2.17 Lệnh INCF

Cú pháp: INCF f,d

($0 \leq f \leq 127, d \in [0,1]$)

Tác dụng: tăng giá trị thanh ghi f lên 1 đơn vị. Kết quả được đưa vào thanh ghi W nếu d = 0 hoặc thanh ghi f nếu d = 1.

Bit trạng thái: Z

3.2.18 Lệnh INCFSZ

Cú pháp: INCFSZ f,d

$(0 \leq f \leq 127, d \in [0,1])$

Tác dụng: tăng giá trị thanh ghi f lên 1 đơn vị. Nếu kết quả khác 0, lệnh tiếp theo được thực thi, nếu kết quả bằng 0, lệnh tiếp theo được thay bằng lệnh NOP. Kết quả sẽ được đưa vào thanh ghi f nếu d=1 hoặc thanh ghi W nếu d = 0.

Bit trạng thái: không có.

3.2.19 Lệnh IORLW

Cú pháp: IORLW k ($0 \leq k \leq 255$)

Tác dụng: thực hiện phép toán OR giữa thanh ghi W và giá trị k. Kết quả được chứa trong thanh ghi W.

Bit trạng thái: Z

3.2.20 Lệnh IORWF

Cú pháp: IORWF f,d

$(0 \leq f \leq 127, d \in [0,1])$

Tác dụng: thực hiện phép toán OR giữa hai thanh ghi W và f. Kết quả được đưa vào thanh ghi W nếu d=0 hoặc thanh ghi f nếu d=1.

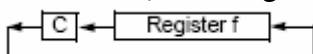
Bit trạng thái: Z

3.2.21 Lệnh RLF

Cú pháp: RLF f,d

$(0 \leq f \leq 127, d \in [0,1])$

Tác dụng: dịch trái các bit trong thanh ghi f qua cờ carry. Kết quả được lưu trong thanh ghi W nếu d=0 hoặc thanh ghi f nếu d=1.



Bit trạng thái: C

3.2.22 Lệnh RETURN

Cú pháp: RETURN

Tác dụng: quay trở về chương trình chính từ một chương trình con

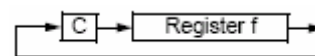
Bit trạng thái: không có

3.2.23 Lệnh RRF

Cú pháp: RRF f,d

$(0 \leq f \leq 127, d \in [0,1])$

Tác dụng: dịch phải các bit trong thanh ghi f qua cờ carry. Kết quả được lưu trong thanh ghi W nếu d=0 hoặc thanh ghi f nếu d=1.



Bit trạng thái: C

3.2.24 Lệnh SLEEP

Cú pháp: SLEEP

Tác dụng: đưa vi điều khiển về chế độ sleep. Khi đó WDT bị xóa về 0, bit \overline{PD} được xóa về 0, bit \overline{TO} được set lên 1 và oscillator không được cho phép hoạt động.

Bit trạng thái: \overline{TO} , \overline{PD} .

3.2.25 Lệnh SUBLW

Cú pháp: SUBLW k

Tác dụng: lấy giá trị k trừ giá trị trong thanh ghi W. Kết quả được chứa trong thanh ghi W.

Bit trạng thái: C, DC, Z

3.2.26 Lệnh SUBWF

Cú pháp: SUBWF f,d

$(0 \leq f \leq 127, d \in [0,1])$

Tác dụng: lấy giá trị trong thanh ghi f đem trừ cho thanh ghi W. Kết quả được lưu trong thanh ghi W nếu d=0 hoặc thanh ghi f nếu d=1.

Bit trạng thái: C, DC, Z

3.2.27 Lệnh SWAP

Cú pháp: SWAP f,d

$(0 \leq f \leq 127, d \in [0,1])$

Tác dụng: đảo 4 bit thấp với 4 bit cao trong thanh ghi f. Kết quả được chứa trong thanh ghi W nếu d=0 hoặc thanh ghi f nếu d=1.

Bit trạng thái: không có

Ngoài các lệnh trên còn có một số lệnh dùng trong chương trình như:

3.2.30 Lệnh #DEFINE

Cú pháp: #DEFINE <text1> <text2>

Tác dụng: thay thế một chuỗi kí tự này bằng một chuỗi kí tự khác, có nghĩa là mỗi khi chuỗi kí tự text1 xuất hiện trong chương trình, trình biên dịch sẽ tự động thay thế chuỗi kí tự đó bằng chuỗi kí tự <text2>.

3.2.31 Lệnh INCLUDE

Cú pháp: #INCLUDE <filename> hoặc #INCLUDE "filename"

Tác dụng: đính kèm một file khác vào chương trình, tương tự như việc ta copy file đó vào vị trí xuất hiện lệnh INCLUDE. Nếu dùng cú pháp <filename> thì file đính kèm là file hệ thống (sitem file), nếu dùng cú pháp "filename" thì file đính kèm là file của người sử dụng.

Thông thường chương trình được đính kèm theo một "header file" chứa các thông tin định nghĩa các biến (thanh ghi W, thanh ghi F,..) và các địa chỉ của các thanh ghi chức năng đặc biệt trong bộ nhớ dữ liệu. Nếu không có header file, chương trình sẽ khó đọc và khó hiểu hơn.

3.2.28 Lệnh XORLW

Cú pháp: XORLW k $(0 \leq k \leq 255)$

Tác dụng: thực hiện phép toán XOR giữa giá trị k và giá trị trong thanh ghi W. Kết quả được lưu trong thanh ghi W.

Bit trạng thái: Z

3.2.29 Lệnh XORWF

Cú pháp: XORWF f,d

Tác dụng: thực hiện phép toán XOR giữa hai giá trị chứa trong thanh ghi W và thanh ghi f. Kết quả được lưu vào trong thanh ghi W nếu d=0 hoặc thanh ghi f nếu d=1.

Bit trạng thái: Z

3.2.32 Lệnh CONSTANT

Cú pháp: CONSTANT <name>=<value>

Tác dụng: khai báo một hằng số, có nghĩa là khi phát hiện chuỗi kí tự “name” trong chương trình, trình biên dịch sẽ tự động thay bằng chuỗi kí tự bằng giá trị “value” đã được định nghĩa trước đó.

3.2.33 Lệnh VARIABLE

Cú pháp: VARIABLE <name>=<value>

Tác dụng: tương tự như lệnh CONSTANT, chỉ có điểm khác biệt duy nhất là giá trị “value” khi dùng lệnh VARIABLE có thể thay đổi được trong quá trình thực thi chương trình còn lệnh CONSTANT thì không.

3.2.34 Lệnh SET

Cú pháp: <name variable> SET <value>

Tác dụng: gán giá trị cho một tên biến. Tên của biến có thể thay đổi được trong quá trình thực thi chương trình.

3.2.35 Lệnh EQU

Cú pháp: <name constant> EQU <value>

Tác dụng: gán giá trị cho tên của tên của hằng số. Tên của hằng số không thay đổi trong quá trình thực thi chương trình.

3.2.36 Lệnh ORG

Cú pháp: ORG <value>

Tác dụng: định nghĩa một địa chỉ chứa chương trình trong bộ nhớ chương trình của vi điều khiển.

3.2.37 Lệnh END

Cú pháp: END

Tác dụng: đánh dấu kết thúc chương trình.

3.2.38 Lệnh __CONFIG

Cú pháp:

Tác dụng: thiết lập các bit điều khiển các khối chức năng của vi điều khiển được chứa trong bộ nhớ chương trình (Configuration bit).

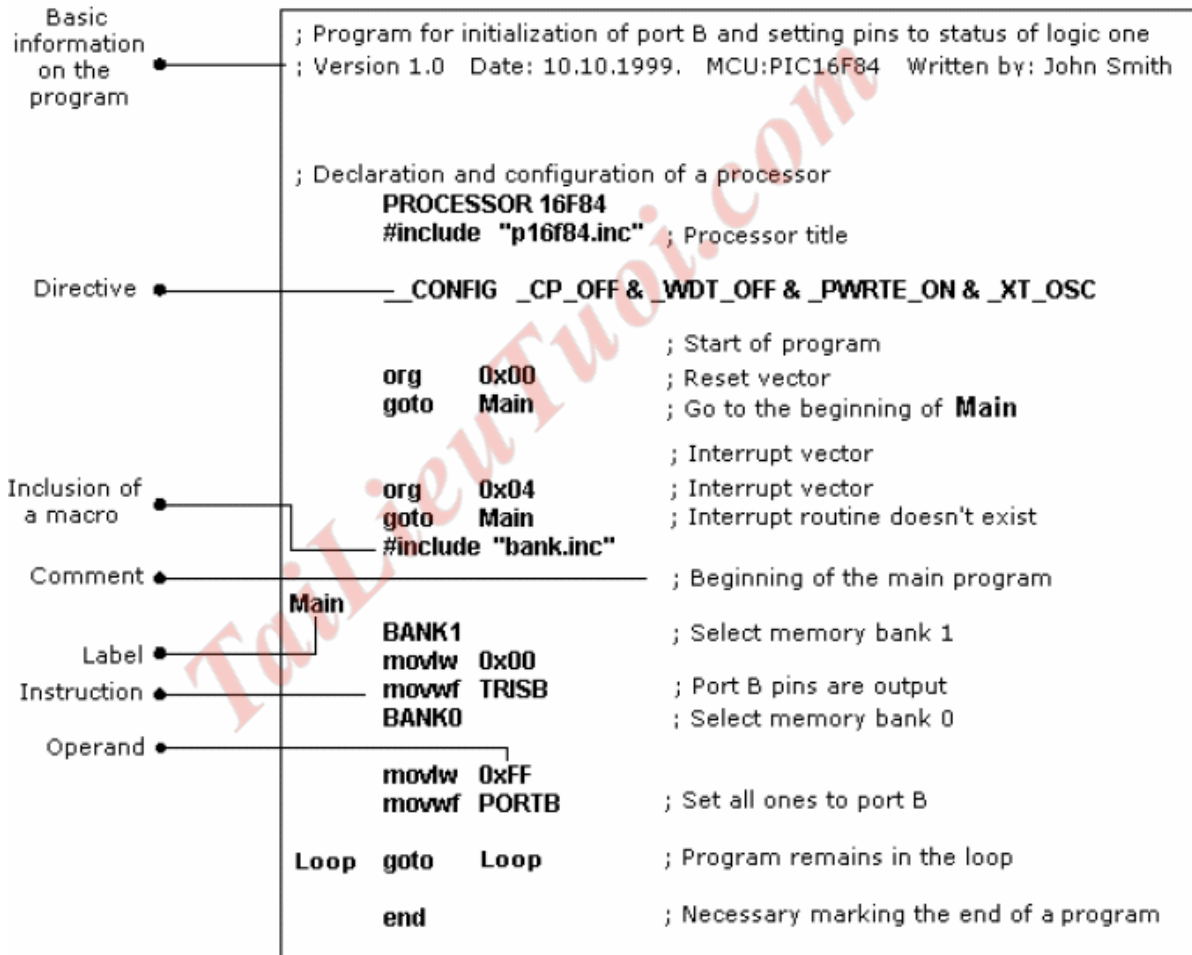
3.2.39 Lệnh PROCESSOR

Cú pháp: PROCESSOR <processor type>

Tác dụng: định nghĩa vi điều khiển nào sử dụng chương trình.

3.3 CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH ASSEMBLY VIẾT CHO VI ĐIỀU KHIỂN PIC

Một chương trình Assembly bao gồm nhiều thành phần như chương trình chính, chương trình ngắt, chương trình con,... Ở đây chỉ trình bày cấu trúc một chương trình đơn giản nhất khi mới bắt đầu làm quen với việc lập trình cho vi điều khiển PIC.



Hình 3.2 Cấu trúc một chương trình Assembly viết cho vi điều khiển PIC.

Ta nhận thấy rằng không có sự khác biệt lớn trong cấu trúc của một chương trình Assembly viết cho vi điều khiển PIC so với vi điều khiển khác, chỉ có sự khác biệt về các lệnh sử dụng trong chương trình. Dấu ";" được dùng để đưa một ghi chú vào chương trình và chỉ có hiệu lực trên một hàng của chương trình. Hình trên là ví dụ về một chương trình đơn giản với các bước khởi tạo cơ bản ban đầu, ngoài ra nếu cần thiết ta vẫn có thể khai báo thêm các biến, hằng và các tham số khác trước chương trình chính (label "Main").

Trong trường hợp cần sử dụng đến chương trình ngắt, ta cần một cấu trúc chương trình phức tạp hơn với nhiều bước khởi tạo phức tạp và phải tuân theo một thứ tự lệnh nhất định. Tuy nhiên nếu sử dụng trình biên dịch MPLAB, cấu trúc của chương trình dành cho một vi điều khiển PIC nhất định đã được viết sẵn, ta chỉ việc viết đoạn chương trình điều khiển vào các vị trí thích hợp trên mẫu chương trình được viết trước đó. Đây là một lợi thế rất lớn khi sử dụng MPLAB để soạn thảo các chương trình viết cho vi điều khiển PIC.

CHƯƠNG 4 MỘT SỐ ỨNG DỤNG CỤ THỂ CỦA PIC16F877A

Trong chương này ta sẽ đi sâu vào một số ứng dụng cụ thể của vi điều khiển PIC16F877A. Các ứng dụng này được xây dựng dựa trên các chức năng ngoại vi được tích hợp sẵn bên trong vi điều khiển, qua đó giúp ta nắm rõ hơn và điều khiển được các khối chức năng đó. Tuy nhiên trước tiên sẽ là một số ứng dụng đơn giản giúp ta bước đầu làm quen với tập lệnh và cách viết chương trình cho vi điều khiển PIC.

4.1 ĐIỀU KHIỂN CÁC PORT I/O.

Đây là một trong những ứng dụng đơn giản nhất giúp ta làm quen với vi điều khiển. Trong ứng dụng này ta sẽ xuất một giá trị nào đó ra một PORT của vi điều khiển, chẳng hạn như PORTB. Giá trị này sẽ được kiểm tra bằng cách gắn vào các pin của PORTB các LED. Khi đó pin mang giá trị mức logic 1 sẽ làm cho LED sáng và pin mang giá trị mức logic 0 sẽ làm cho LED tắt.

Sau đây là một vài điểm cần chú ý cho ứng dụng này:

Để LED sáng bình thường thì điện áp đặt lên LED vào khoảng 1.8 đến 2.2 Volt tùy theo màu sắc của LED, trong khi điện áp tại ngõ ra của 1 pin trong PORTB nếu ở mức logic 1 thường là 5 volt. Do đó ta cần có thêm điện trở mắc nối tiếp với LED để hạn dòng (có thể dùng điện trở 0.33 K).

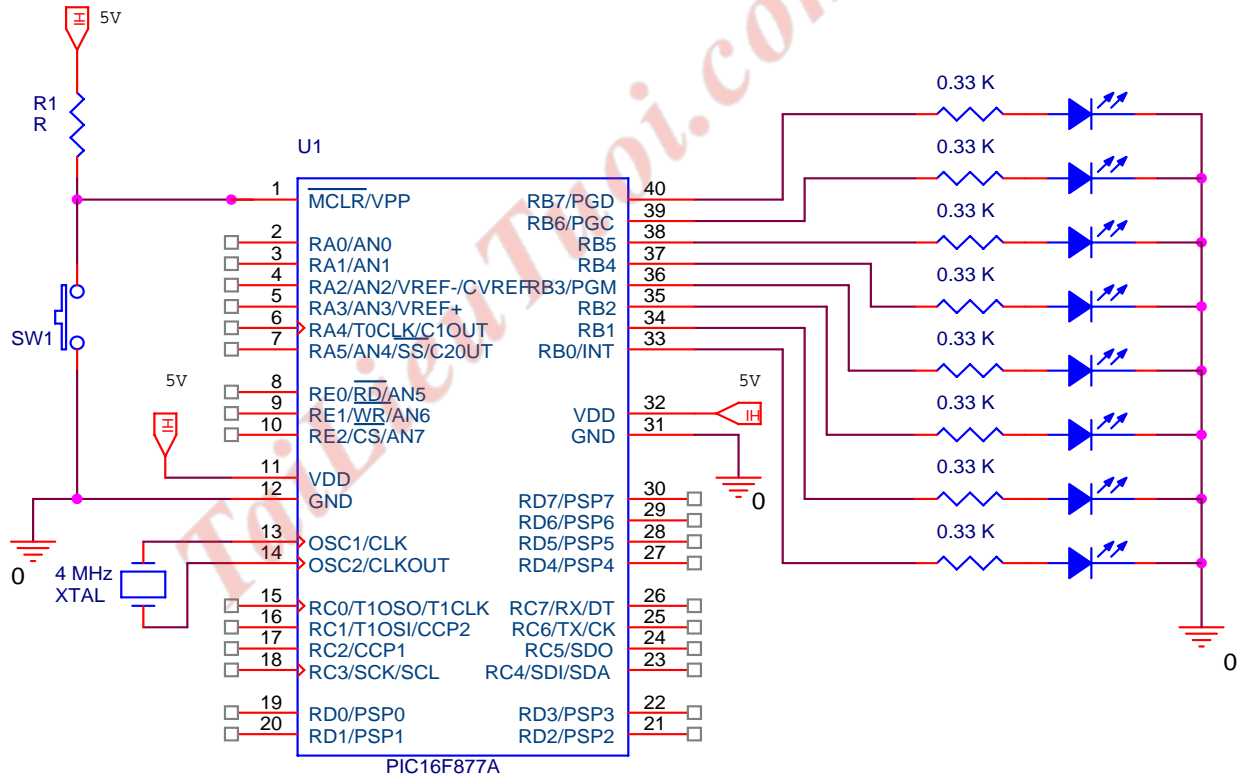
Để xuất được giá trị ra PORTB, trước hết ta cần khởi tạo các pin của PORTB là output. Điều này được thực hiện bằng cách clear các bit trong thanh ghi TRISB. Tuy nhiên hai thanh ghi PORTB và TRISB nằm ở hai bank khác nhau trong bộ nhớ dữ liệu. Do đó trước khi muốn truy xuất giá trị trong một thanh ghi nào đó cần chọn bank dữ liệu chứa thanh ghi đó bằng cách đưa các giá trị thích hợp vào 2 bit RP1:RP0 của thanh ghi STATUS (xem phụ lục 2 và sơ đồ bộ nhớ dữ liệu).

Do trong tập lệnh của vi điều khiển PIC không có lệnh nào cho phép đưa một byte vào một thanh ghi cho trước, do đó cần sử dụng một thanh ghi trung gian (thanh ghi W) và dùng hai lệnh MOVLW (đưa byte vào thanh ghi W) và lệnh MOVWF (đưa giá trị trong thanh ghi W vào thanh ghi f nào đó mà ta muốn).

Ngoài ra cần dùng lệnh ORG để chỉ ra địa chỉ bắt đầu chương trình khi vi điều khiển được reset. Thông thường địa chỉ bắt đầu chương trình sẽ là địa chỉ 0000h.

Trong trường hợp cần dùng đến chế độ reset của pin MCLR, ta có thể thiết kế thêm một mạch reset ngoại vi (vi điều khiển sẽ được reset khi pin MCLR chuyển từ mức logic 1 xuống mức logic 0).

Sau đây là sơ đồ mạch của ứng dụng trên:



Hình 4.1 Mạch nguyên lí của ứng dụng điều khiển các PORT của vi điều khiển.

Một điểm cần chú ý là vi điều khiển PIC16F877A có đến 2 pin V_{DD} và 2 pin GND. Trong trường hợp này ta phải cấp nguồn vào tất cả các pin trên, khi đó vi điều khiển mới có đủ điện áp để hoạt động.

Chương trình viết cho ứng dụng trên như sau:

```
;chương trình 4.1.1
;PORTBTEST.ASM

processor    16f877a           ; khai báo vi điều khiển
include     <p16f877a.inc>    ; header file đính kèm

__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON &
_XT_OSC & _WRT_OFF & _LVP_OFF & _CPD_OFF
                        ; khai báo các "Configuration bits"

                ORG            0x000           ; địa chỉ bắt đầu chương trình
                GOTO           start
Start
                BCF            STATUS,RP1
                BCF            STATUS,RP0       ; chọn BANK0

                CLRF           PORTB           ; xóa PORTB
                BSF            STATUS,RP0       ; chọn BANK1

                MOVLW          0x00
                MOVWF          TRISB           ; PORTB <- outputs

                BCF            STATUS,RP0       ; chọn BANK0

                MOVLW          0x8F           ; giá trị cần đưa ra PORTB
                MOVWF          PORTB          ; PORTB <- 8Fh

                loop    GOTO    loop           ; vòng lặp vô hạn

END                                                  ; kết thúc chương trình
```

Các bước tiếp theo để hoàn tất ứng dụng trên là biên dịch chương trình trên bằng một trình biên dịch Assembly dành cho vi điều khiển PIC (trình biên dịch MPLAB chẳng hạn), sau đó dùng mạch nạp để nạp chương trình vào vi điều khiển PIC và kiểm tra kết quả. Nếu không có lỗi nào xảy ra, LED gắn vào các pin RB7, RB3, RB2, RB1, RB0 sẽ sáng, LED gắn vào các pin còn lại sẽ tắt (do giá trị ta đưa ra PORTB là 8Fh).

Hoàn toàn tương tự ta có thể viết chương trình đưa một giá trị bất kỳ vào các PORT của vi điều khiển PIC16F877A. Tuy nhiên có một điều cần chú ý là đối với PORTA, do pin

RA4 có cực thu để hở (xem phụ lục 1) nên muốn PORTA hiển thị kết quả một cách chính xác ta cần dùng một điện trở kéo lên gắn thêm vào bên ngoài pin RA4.

4.1.1 CHƯƠNG TRÌNH DELAY

Chương trình trên giúp ta đưa giá trị ra các PORT của vi điều khiển và các LED sẽ sáng hay tắt tùy theo mức logic đưa ra các PORT. Bây giờ ta lại muốn các LED sẽ chớp tắt sau một khoảng thời gian định trước. Muốn vậy ta dùng thêm một đoạn chương trình DELAY. Thực chất của chương trình DELAY là cho vi điều khiển làm một công việc vô nghĩa nào đó trong một khoảng thời gian định trước. Khoảng thời gian này được tính toán dựa trên quá trình thực thi lệnh, hay cụ thể hơn là dựa vào thời gian của một chu kì lệnh. Có thể viết chương trình DELAY dựa trên đoạn chương trình sau:

```
        MOVLW    0X20      ; giá trị 20h
        MOVWF    delay-reg ; đưa vào thanh ghi delay
loop    DECFSZ    delay-reg ; giảm giá trị thanh ghi delay-reg 1 đơn vị
        GOTO     loop      ; nhảy tới label "loop" nếu thanh ghi delay-reg
                               ;sau khi giảm 1 đơn vị chứa giá trị khác 0.
        .....            ; lệnh này được thực thi khi delay-reg bằng 0
```

Nếu dùng đoạn chương trình này thì thời gian delay được tính gần đúng như sau:

$$t_d = 3(1+t_v)t_i$$

Trong đó t_d là thời gian delay, t_v là giá trị đưa vào thanh ghi delay-reg và t_i là thời gian của một chu kì lệnh và được tính theo công thức:

$$t_i = 4/f_0$$

Với f_0 là tần số của oscillator. Sở dĩ có công thức này là vì một chu kì lệnh bao gồm 4 xung clock. Công thức này chỉ gần đúng vì ta đã bỏ qua thời gian thực thi các lệnh trước label "loop" và một chu kì lệnh phát sinh khi thanh ghi delay-reg mang giá trị 0 (trường hợp này cần hai chu kì lệnh để thực thi lệnh DECFSZ).

Do thanh ghi delay-reg chỉ mang giá trị lớn nhất là FFh nên thời gian delay chỉ giới hạn ở một khoảng thời gian nhất định tùy thuộc vào xung clock sử dụng để cấp cho vi điều khiển. Muốn tăng thời gian delay ta có thể gọi chương trình delay nhiều lần hoặc tăng số lượng vòng lặp của chương trình delay như sau:

```
        MOVLW    0Xff
        MOVWF    delay-reg1
loop    DECFSZ    delay-reg1
        GOTO     loop1      ; thực thi dòng lệnh này nếu delay-reg khác 0
        GOTO     exit       ; thực thi dòng lệnh này nếu delay-reg bằng 0
Loop1   MOVLW    0Xff
        MOVWF    delay-reg2
        DECFSZ    delay-reg2
```

	MOVWF	loop1	; thực thi dừng lệnh này nếu delay-reg khác 0
	GOTO	loop	; thực thi dừng lệnh này nếu delay-reg bằng 0
Exit		; lệnh tiếp theo sau thời gian delay

Với đoạn chương trình trên thời gian delay chỉ kết thúc khi cả hai thanh ghi delay-reg1 và delay-reg2 đều mang giá trị 0.

Sau đây là một ví dụ cụ thể. Yêu cầu đặt ra là cho các LED trong chương trình 4.1 chớp tắt sau mỗi 100 miligiây. Giả sử ta đang sử dụng oscillator 4MHz. Khi đó thời gian của một chu kì lệnh là:

$$t_i = 4/4 \text{ MHz} = 1 \text{ uS.}$$

Với thời gian cần delay là t_d bằng 1s thì giá trị cần đưa vào thanh ghi delay-reg là:

$$t_v = (t_d/3t_i) - 1 = 33332.$$

Như vậy ta đưa vào thanh ghi delay-reg2 giá trị 255 (FFh) và thanh ghi delay-reg1 giá trị $33332/255 = 131$ (83h).

Chương trình được viết như sau:

;chương trình 4.1.2

;PORTBTESTANDDELAY.ASM

;Version 1.1

processor	16f877a	; khai báo vi điều khiển
include	<p16f877a.inc>	; header file đính kèm

__CONFIG __CP_OFF & __WDT_OFF & __BODEN_OFF & __PWRTE_ON & __XT_OSC & __WRT_OFF & __LVP_OFF & __CPD_OFF		; khai báo các "Configuration bits"
---------------------------------------------------------------------------------------------------------	--	-------------------------------------

delay_reg1	EQU	0x20	; khai báo địa chỉ các ô nhớ chứa các thanh ghi
delay_reg2	EQU	0x21	; delay-reg1 và delay-reg2

	ORG	0x000	; địa chỉ bắt đầu chương trình
	GOTO	start	
start			; chương trình chính bắt đầu tại đây

BCF	STATUS,RP1	
BCF	STATUS,RP0	; chọn BANK0

CLRF	PORTB	; xóa PORTB
BSF	STATUS,RP0	; chọn BANK1

MOVLW	0x00	
-------	------	--

	MOVWF	TRISB	; PORTB <- outputs
	BCF	STATUS,RP0	; chọn BANK0
loop	MOVLW	0x8F	; giá trị cần đưa ra PORTB
	MOVWF	PORTB	; PORTB <- 8Fh
	MOVLW	0x83	
	MOVWF	delay_reg1	
	MOVLW	0xFF	
	MOVWF	delay_reg2	
loop1	DECFSZ	delay_reg1	
	GOTO	loop2	
	GOTO	exit1	
loop2	DECFSZ	delay_reg2	
	GOTO	loop2	
	GOTO	loop1	; delay 100 ms
exit1	CLRF	PORTB	; xóa PORTB
	MOVLW	0x83	
	MOVWF	delay_reg1	
	MOVLW	0xFF	
	MOVWF	delay_reg2	
loop3	DECFSZ	delay_reg1	
	GOTO	loop4	
	GOTO	exit2	
loop4	DECFSZ	delay_reg2	
	GOTO	loop4	
	GOTO	loop3	; delay 100 ms
exit2			
	GOTO	loop	; vòng lặp vô hạn
END			; kết thúc chương trình

Với chương trình này các pin của PORTB sẽ thay đổi trạng thái sau mỗi khoảng thời gian delay là 100 ms. Điều này cho phép ta nhận thấy bằng mắt thường vì trong một giây các pin của PORTB sẽ thay đổi trạng thái 10 lần.

Tuy nhiên ta dễ dàng nhận thấy một nhược điểm của chương trình trên là cần tới hai đoạn chương trình delay với cấu trúc chương trình, thuật toán và chức năng hoàn toàn giống nhau. Điều này làm cho chương trình trở nên phức tạp và tốn nhiều dung lượng bộ nhớ của vi điều khiển. Điều này cần được chú trọng vì dung lượng bộ nhớ chương trình của một vi điều khiển thường nhỏ (đối với PIC16F877A dung lượng bộ nhớ chương trình là 8K word với một word là 14 bit). Một phương pháp để khắc phục nhược điểm này là sử dụng chương trình con và dùng lệnh “CALL” để gọi chương trình con đó. Chương trình con có thể được đặt tại bất cứ vị trí nào trong chương trình chính. Chương trình 4.2 khi đó được viết lại như sau:

;chương trình 4.1.3

;PORTBTESTANDDELAY.ASM

;Version 1.2

processor 16f877a ; khai báo vi điều khiển

include <p16f877a.inc> ; header file đính kèm

_CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

; khai báo các “Configuration bits”

delay_reg1 EQU 0x20 ; khai báo địa chỉ các ô nhớ chứa các thanh ghi

delay_reg2 EQU 0x21 ; delay-reg1 và delay-reg2

ORG 0x000 ; địa chỉ bắt đầu chương trình

GOTO start

start ; chương trình chính bắt đầu tại đây

BCF STATUS,RP1

BCF STATUS,RP0 ; chọn BANK0

CLRF PORTB ; xóa PORTB

BSF STATUS,RP0 ; chọn BANK1

MOVLW 0x00

MOVWF TRISB ; PORTB <- outputs

BCF STATUS,RP0 ; chọn BANK0

loop MOVLW 0x8F ; giá trị bất kì cần đưa ra PORTB

MOVWF PORTB ; PORTB <- 8Fh

CALL delay100ms ; gọi chương trình con delay100ms


```

CLRF      PORTB          ; xóa PORTB
CALL      delay100ms
GOTO      loop           ; vòng lặp vô hạn

```

Delay100ms

```

        MOVLW    0x83
        MOVWF    delay_reg1
        MOVLW    0xFF
        MOVWF    delay_reg2

loop1   DECFSZ    delay_reg1
        GOTO     loop2
        GOTO     exit
loop2   DECFSZ    delay_reg2
        GOTO     loop2
        GOTO     loop1      ; delay 100 ms

Exit
        RETURN    ; trở về chương trình chính

END      ; kết thúc chương trình

```

Với cách viết chương trình sử dụng chương trình con, cấu trúc chương trình sẽ trở nên gọn gàng dễ hiểu hơn, linh hoạt hơn và tiết kiệm được nhiều dung lượng bộ nhớ chương trình.

Bây giờ ta sẽ bàn đến một thuật toán khác để viết chương trình delay. Về nguyên tắc thì thuật toán mới này không có nhiều khác biệt so với thuật toán cũ, tuy nhiên lệnh sử dụng trong chương trình và cách tính toán thời gian delay thì khác nhau. Chương trình con delay100ms với oscillator 4 MHz có thể được viết lại như sau:

delay100ms

```

        MOVLW    d'100'
        MOVWF    count1
d1      MOVLW    0xC7
        MOVWF    counta
        MOVLW    0x01
        MOVWF    countb
delay_0
        DECFSZ    counta,1
        GOTO     $+2

```

DECFSZ	countb,1
GOTO	delay_0
DECFSZ	count1,1
GOTO	d1
RETLW	0x00

END

Trước tiên ta xét đoạn chương trình kể từ label “delay_0”. Lệnh DECFSZ mất một chu kì lệnh (trừ trường hợp thanh ghi counta mang giá trị 0 thì cần 2 chu kì lệnh), lệnh GOTO \$+2 mất hai chu kì lệnh. Lệnh này có tác dụng cộng vào bộ đếm chương trình giá trị 2, khi đó chương trình sẽ nhảy tới lệnh có địa chỉ (PC+2), tức là lệnh GOTO delay_0, lệnh này cũng tốn hai chu kì lệnh. Như vậy ta cần tổng cộng 5 chu kì lệnh để giảm giá trị trong thanh ghi counta 1 đơn vị. Thanh ghi counta mang giá trị 199 (C7h), do đó đoạn chương trình này sẽ tạo ra một khoảng thời gian delay:

$$t_d = 5(\text{counta}+1) \cdot t_i = 5(199+1) \cdot 1 \text{ uS} = 1 \text{ mS}$$

Muốn tạo ra thời gian delay 100 mS, ta chỉ việc đưa giá trị 100 vào thanh ghi count1.

Với giải thuật này thời gian delay tạo ra sẽ dài hơn so với giải thuật mà ta sử dụng ở chương trình 4.2. Bên cạnh đó ta có thể viết một chương trình con có tác dụng delay một khoảng thời gian bất kì là bội số của 1 mS một cách dễ dàng.

Trong chương trình trên ta còn sử dụng thêm một lệnh khá lạ là lệnh RETLW. Lệnh này có tác dụng trở về vị trí mà chương trình con được gọi và thanh ghi W khi đó mang giá trị là tham số của lệnh RETLW (00h). Trong trường hợp này thanh ghi W không cần mang một giá trị cụ thể khi quay trở về chương trình chính nên lệnh RETLW chỉ có tác dụng như lệnh RETURN.

4.1.2 MỘT SỐ ỨNG DỤNG VỀ ĐẶC TÍNH I/O CỦA CÁC PORT ĐIỀU KHIỂN

Dựa vào chương trình delay và thao tác đưa dữ liệu ra các PORT, ta phát triển thêm một số chương trình nhỏ với mục đích làm quen với cách viết chương trình cho vi điều khiển PIC16F877A.

Ứng dụng 4.1:

Dựa vào mạch nguyên lí hình 4.1 viết chương trình điều khiển LED chạy. Cụ thể là sau thời gian delay 250 ms, LED tiếp theo sẽ sáng một cách tuần tự từ trên xuống dưới.

Chương trình này được viết dựa vào chương trình 4.3 với một vài thay đổi nhỏ. Thay vì đưa một giá trị bất kì ra PORT, ta đưa ra PORB giá trị 80h, sau đó dịch phải giá trị 80h sau mỗi khoảng thời gian delay (dùng lệnh RRF).

; Chương trình 4.1.4

; Chương trình điều khiển LED chạy

```
processor    16f877a                ; khai báo vi điều khiển
include     <p16f877a.inc>          ; header file đính kèm
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
; khai báo các "Configuration bits"
```

```
;-----
;Khai báo biến
;-----
```

```
count1      EQU    0x20            ; dùng cho chương trình delay
counta      EQU    0x21            ; dùng cho chương trình delay
countb      EQU    0x22            ; dùng cho chương trình delay

ORG          0x000                ; địa chỉ bắt đầu chương trình
GOTO        start

start        ; chương trình chính bắt đầu tại đây
BCF         STATUS,RP1
BCF         STATUS,RP0            ; chọn BANK0
CLRF        PORTB                 ; xóa PORTB
BSF         STATUS,RP0            ; chọn BANK1
MOVLW       0x00
MOVWF       TRISB                 ; PORTB <- outputs
BCF         STATUS,RP0            ; chọn BANK0
MOVLW       0x8F
MOVWF       PORTB                 ; PORTB <- 8Fh
loop        CALL    delay100ms     ; gọi chương trình con delay100ms
RRF         PORTB,1                ; dịch phải PORTB
GOTO        loop                  ; vòng lặp vô hạn
```

```
delay100ms
    MOVLW    d'100'
    MOVWF    count1
d1    MOVLW    0xC7
    MOVWF    counta
    MOVLW    0x01
```

```

        MOVWF    countb
delay_0
        DECFSZ   counta,1
        GOTO     $+2
        DECFSZ   countb,1
        GOTO     delay_0
        DECFSZ   count1,1
        GOTO     d1                ; delay 100ms
        RETLW    0x00              ; trở về chương trình chính
END                                  ; kết thúc chương trình

```

Như vậy dựa trên một số chương trình cơ bản, ta chỉ cần thay đổi một số chi tiết là có thể tạo ra một ứng dụng mới.

Một phương pháp khác để viết chương trình trên là dùng bảng dữ liệu. Phương pháp bảng dữ liệu được đưa ra ở đây không mang tính chất tối ưu hóa giải thuật chương trình mà chỉ mang tính chất làm quen với một giải thuật mới, qua đó tạo điều kiện thuận lợi hơn trong việc viết các chương trình ứng dụng phức tạp hơn sau này. Ta có thể viết lại chương trình trên theo phương pháp bảng dữ liệu như sau:

; Chương trình 4.1.5

; Chương trình điều khiển LED chạy dùng bảng dữ liệu

```

processor    16f877a                ; khai báo vi điều khiển
include      <p16f877a.inc>          ; header file đính kèm
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
                                           ; khai báo các "Configuration bits"
count1      EQU    0x20              ; dùng cho chương trình delay
counta      EQU    0x21              ; dùng cho chương trình delay
countb      EQU    0x22              ; dùng cho chương trình delay
count       EQU    0x23              ; dùng để tra bảng dữ liệu

        ORG      0x000              ; địa chỉ bắt đầu chương trình
        GOTO     start

start
                                           ; chương trình chính bắt đầu tại đây
        BCF      STATUS,RP1
        BCF      STATUS,RP0          ; chọn BANK0
        CLRF     PORTB               ; xóa PORTB
        BSF      STATUS,RP0          ; chọn BANK1
        MOVLW    0x00
        MOVWF    TRISB               ; PORTB <- outputs

```

	BCF	STATUS,RP0	; chọn BANK0
Loop1	CLRF	count	; reset thanh ghi chứa giá trị đếm
Loop2	MOVF	count, 0	; đưa giá trị đếm vào thanh ghi W
	CALL	Table	; gọi chương trình con Table
	MOVWF	PORTB	; xuất giá trị chứa trong thanh ghi W ra PORTB
	CALL	delay100ms	; gọi chương trình con delay100ms
	INCF	count, 0	; tăng giá trị thanh ghi count và chứa kết quả trong ; thanh ghi W
	XORLW	d'8'	; so sánh thanh ghi W với giá trị 8
	BTFSC	STATUS,Z	; kiểm tra bit Z (Zero)
	GOTO	Loop1	; nhảy về label Loop1 nếu W = 0
	INCF	count, 1	; thực thi lệnh này nếu W khác 0
	GOTO	Loop2	
Table	ADDWF	PCL,1	; cộng giá trị thanh ghi W vào thanh ghi PCL, kết ; quả chứa trong thanh ghi PCL
	RETLW	b'10000000'	
	RETLW	b'01000000'	
	RETLW	b'00100000'	
	RETLW	b'00010000'	
	RETLW	b'00001000'	
	RETLW	b'00000100'	
	RETLW	b'00000010'	
	RETLW	b'00000001'	
delay100ms	MOVLW	d'100'	
	MOVWF	count1	
d1	MOVLW	0xC7	
	MOVWF	counta	
	MOVLW	0x01	
	MOVWF	countb	
delay_0	DECFSZ	counta,1	
	GOTO	\$(+2)	
	DECFSZ	countb,1	
	GOTO	delay_0	
	DECFSZ	count1,1	
	GOTO	d1	; delay 100ms

RETURN	; trở về chương trình chính
END	; kết thúc chương trình

Ở phần trước ta đã từng đề cập đến lệnh RETLW nhưng khi đó lệnh này chỉ có tác dụng như lệnh RETURN. Tuy nhiên trong trường hợp này lệnh RETLW có một vai trò cụ thể hơn là mang dữ liệu từ bảng dữ liệu trở về chương trình chính và xuất ra PORTB dữ liệu vừa mang về đó. Sau mỗi lần mang dữ liệu về biến count sẽ tăng giá trị đếm lên. Giá trị đếm được đưa vào thanh ghi W để cộng vào thanh ghi PCL. Thanh ghi PCL là thanh ghi chứa giá trị bộ đếm chương trình, giá trị từ biến count được cộng vào thanh ghi PCL thông qua thanh ghi W sẽ điều khiển chương trình nhảy tới đúng địa chỉ cần lấy dữ liệu từ bảng dữ liệu vào thanh ghi W và thanh ghi W mang dữ liệu đó trở về chương trình chính thông qua lệnh RETLW.

Để đề phòng trường hợp giá trị biến count cộng vào thanh ghi PCL sẽ điều khiển chương trình đến vị trí vượt qua vị trí của bảng dữ liệu (trường hợp này xảy ra khi biến count mang giá trị lớn hơn 8, khi đó vị trí lệnh cần thực thi do bộ đếm chương trình chỉ đến không còn đúng nữa), ta so sánh biến count với giá trị 8. Nếu biến count mang giá trị 8 thì phép toán XOR giữa biến cao và giá trị sẽ có kết quả bằng 0 và cờ Z trong thanh ghi STATUS sẽ được set. Lúc này ta cần reset lại biến count bằng cách nhảy về label Loop1.

Việc dùng bảng dữ liệu trong trường hợp này làm cho chương trình trở nên dài hơn, quá trình thực thi chương trình lâu hơn vì bộ đếm chương trình liên tục bị thay đổi giá trị, tuy nhiên ta cũng thấy được một ưu điểm của việc dùng bảng dữ liệu là cho phép ta sắp xếp bố trí dữ liệu một cách linh hoạt. Điều này thể hiện qua việc chỉ cần thay đổi dữ liệu trong bảng dữ liệu, ta sẽ có được nhiều cách điều khiển các LED sáng hay tắt theo nhiều qui luật khác nhau chứ không chỉ đơn thuần là dịch LED sáng sang trái hoặc sang phải. Ứng dụng sau đây cho ta thấy rõ hơn hiệu quả của bảng dữ liệu.

Ứng dụng 4.2: Tương tự như ứng dụng 1, nhưng lần này ta cho LED chạy từ vị trí giữa sang hai phía sau mỗi khoảng thời gian delay 100 ms.

Chương trình cho ứng dụng này hoàn toàn tương tự như trong ứng dụng, ta chỉ cần thay đổi bảng dữ liệu một cách thích hợp.

; Chương trình 4.1.6
; Chương trình điều khiển hiển thị LED

```
processor    16f877a                ; khai báo vi điều khiển
include      <p16f877a.inc>         ; header file đính kèm
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```

; khai báo các “Configuration bits”

;Khai báo biến

count1	EQU	0x20	; dùng cho chương trình delay
counta	EQU	0x21	; dùng cho chương trình delay
countb	EQU	0x22	; dùng cho chương trình delay
count	EQU	0x23	; dùng để tra bảng dữ liệu
	ORG	0x000	; địa chỉ bắt đầu chương trình
	GOTO	start	
start			; chương trình chính bắt đầu tại đây
	BCF	STATUS,RP1	
	BCF	STATUS,RP0	; chọn BANK0
	CLRF	PORTB	; xóa PORTB
	BSF	STATUS,RP0	; chọn BANK1
	MOVLW	0x00	
	MOVWF	TRISB	; PORTB <- outputs
	BCF	STATUS,RP0	; chọn BANK0
Loop1			
	CLRF	count	; reset thanh ghi chứa giá trị đếm
Loop2			
	MOVF	count, 0	; đưa giá trị đếm vào thanh ghi W
	CALL	Table	; gọi chương trình con Table
	MOVWF	PORTB	; xuất giá trị chứa trong thanh ghi W ra PORTB
	CALL	delay100ms	; gọi chương trình con delay100ms
	INCF	count, 0	; tăng giá trị thanh ghi count và chứa kết quả trong
			; thanh ghi W
	XORLW	d'8'	; so sánh thanh ghi W với giá trị 8
	BTFSC	STATUS,Z	; kiểm tra bit Z (Zero)
	GOTO	Loop1	; nhảy về label Loop1 nếu W = 0
	INCF	count, 1	; thực thi lệnh này nếu W khác 0
	GOTO	Loop2	
Table			
	ADDWF	PCL,1	; cộng giá trị thanh ghi W vào thanh ghi PCL, kết
			; quả chứa trong thanh ghi PCL
	RETLW	b'00011000'	
	RETLW	b'00100100'	
	RETLW	b'01000010'	


```

RETLW    b'10000001'
RETLW    b'01000010'
RETLW    b'00100100'
RETLW    b'00011000'
RETLW    b'00100100'

```

delay100ms

```

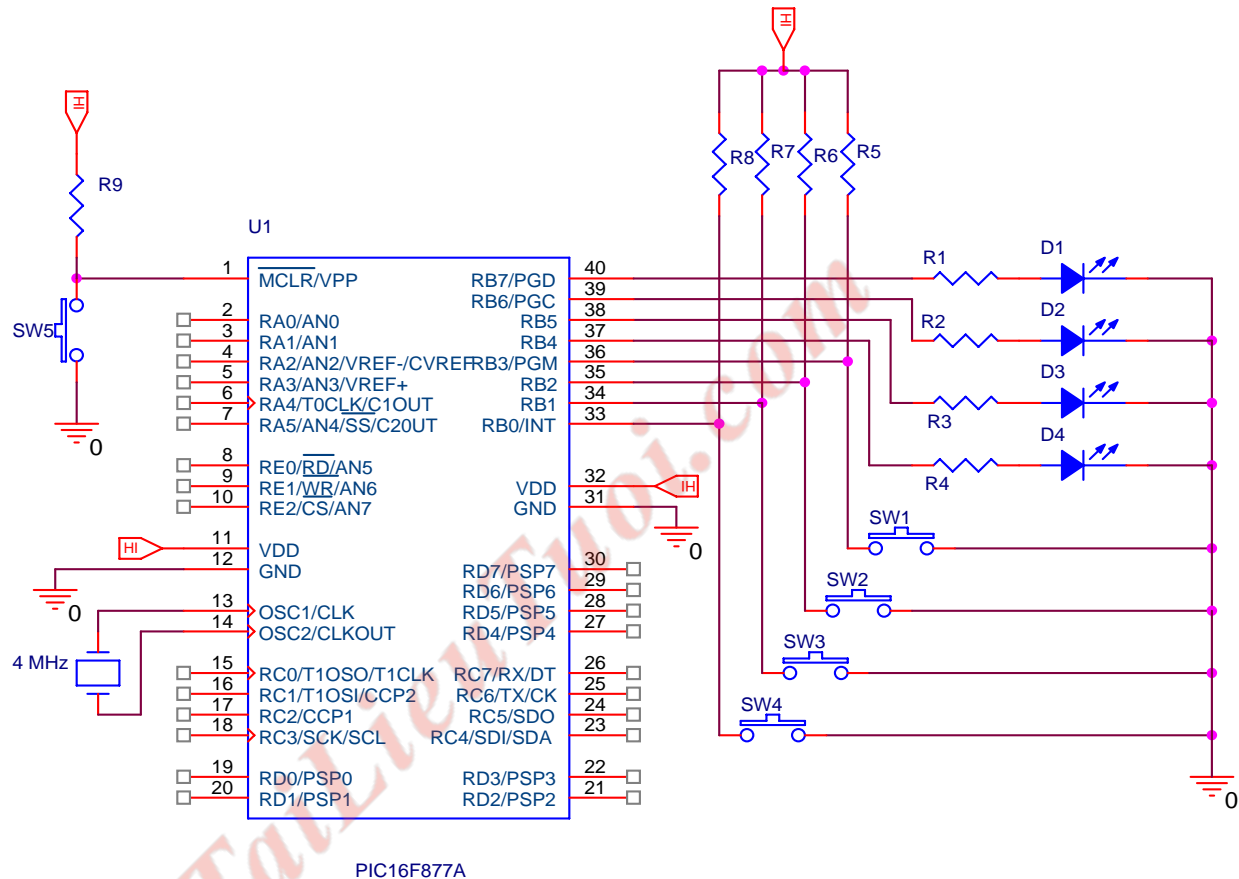
                MOVLW    d'100'
                MOVWF    count1
d1      MOVLW    0xC7
                MOVWF    counta
                MOVLW    0x01
                MOVWF    countb
delay_0
                DECFSZ   counta,1
                GOTO     $+2
                DECFSZ   countb,1
                GOTO     delay_0
                DECFSZ   count1,1
                GOTO     d1                ; delay 100ms
                RETURN                ; trở về chương trình chính

END                ; kết thúc chương trình

```

Ứng dụng 4.3: Test chức năng Input/Output của các pin của vi điều khiển.

Ở các ứng dụng trước ta chỉ làm một việc là xuất tín hiệu điều khiển ra các PORT theo một số qui tắc định sẵn nào đó. Trong ứng dụng này ta sẽ phát triển thêm một chức năng nữa của các PORT là khả năng nhận tín hiệu điều khiển từ bên ngoài. Vi điều khiển sẽ đọc tín hiệu 0 (điện áp 0 V) và 1 (điện áp 5 V) được tạo ra bằng cách sử dụng các công tắc ấn từ các pin RB0:RB3 của PORTB , sau đó kiểm tra xem công tắc nào được ấn và bật LED tương ứng với công tắc đó (các LED này được bố trí ở các pin RB7:RB4) sáng lên. Để kiểm tra được ứng dụng này ta cần xây dựng sơ đồ mạch như sau:



Hình 4.2 Mạch test chức năng I/O cho ứng dụng 3.

Chương trình viết cho ứng dụng này như sau:

;Chương trình 4.1.7

processor 16f877a

include <p16f877a.inc>

__CONFIG __CP_OFF & __WDT_OFF & __BODEN_OFF & __PWRTE_ON & __XT_OSC & __WRT_OFF & __LVP_OFF & __CPD_OFF

;Khai báo hằng

SW1	EQU	0
SW2	EQU	1
SW3	EQU	2
SW4	EQU	3
LED1	EQU	4
LED2	EQU	5

LED3	EQU	6
LED4	EQU	7

```

        ORG      0x000
        GOTO     start

start
        BCF      STATUS,RP1
        BCF      STATUS,RP0
        CLRF     PORTB
        BSF      STATUS,RP0
        MOVLW    b'00001111'    ; thiết lập chức năng I/O cho từng pin trong
                                ;PORTB

        MOVWF    TRISB
        BCF      STATUS,RP0

loop
        BTFSS    PORTB,SW1    ; kiểm tra công tắc 1
        CALL     switch1      ; thực thi lệnh này nếu công tắc 1 được ấn
        BTFSS    PORTB,SW2    ; nếu công tắc ; 1 không được ấn, kiểm tra công
                                ; tắc 2
        CALL     switch2      ; tiếp tục quá trình đối với các công tắc còn lại
        BTFSS    PORTB,SW3
        CALL     switch3
        BTFSS    PORTB,SW4
        CALL     switch4
        GOTO     loop

switch1
        CLRF     PORTB
        BSF      PORTB,LED1
        RETURN

switch2
        CLRF     PORTB
        BSF      PORTB,LED2
        RETURN

switch3
        CLRF     PORTB
        BSF      PORTB,LED3
        RETURN

switch4
        CLRF     PORTB

```

```
BSF      PORTB,LED4
RETURN
```

```
END
```

Trong chương trình trên ta ứng dụng thuật toán hồi vòng thông qua vòng lặp loop trong phần chương trình chính. Khi công tắc không được nhấn, mức logic tại các pin nối với công tắc là mức 1. Khi công tắc được ấn, các pin trên sẽ như nối đất và mang mức logic 0. Ta chỉ việc kiểm tra liên tục trạng thái logic của các pin đó và bật LED tương ứng với công tắc thông qua các chương trình con switch1, switch2, switch3 và switch4 khi phát hiện một công tắc nào đó được ấn. Tuy nhiên cần chú ý là phải thiết lập trạng thái I/O thích hợp cho từng pin trong PORTB (thiết lập RB3:RB0 là input, RB7:RB4 là output).

Một điểm quan trọng cần lưu ý là các công tắc ấn thường bị “dội”, tức là khi ấn xuống hoặc thả ra, điện áp tại các công tắc sẽ phải trải qua một giai đoạn quá độ, điện áp sẽ dao động không ổn định trong một khoảng thời gian nào đó, ngoài ra trạng thái logic của pin cũng sẽ thay đổi do một tác động tức thời từ một trường bên ngoài mà không phải do ta ấn công tắc. Các yếu tố trên sẽ làm ảnh hưởng tới hoạt động của vi điều khiển. Để khắc phục nhược điểm trên ta có hai phương pháp:

Phương pháp chống “dội” bằng phần cứng: ta thêm các tụ điện vào các công tắc để lọc bớt các tín hiệu nhiễu gây nhiễu và các tín hiệu không ổn định trong thời gian quá độ. Phương pháp này cũng hiệu quả nhưng gây tốn kém về linh kiện và mạch nguyên lý trở nên phức tạp.

Phương pháp chống “dội” bằng phần mềm: ta cho vi điều khiển delay trong một thời gian ngắn và kiểm tra xem công tắc còn được ấn không, nếu công tắc thực sự còn được ấn thì mới tiến hành các thao tác tương ứng với công tắc đó.

Chương trình cải tiến để khắc phục nhược điểm trên có thể được viết như sau:

;Chương trình 4.1.8

```
processor    16f877a
```

```
include      <p16f877a.inc>
```

```
_CONFIG     _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```

```
;-----
```

```
;Khai báo hằng
```

```
;-----
```

```
SW1          EQU          0
```

```
SW2          EQU          1
```

```
SW3          EQU          2
```

```
SW4          EQU          3
```

LED1	EQU	4
LED2	EQU	5
LED3	EQU	6
LED4	EQU	7

;Khai báo biến

count1	EQU	0x20
counta	EQU	0x21
countb	EQU	0x22

;Các khai báo khác

SWdel	SET	del150	; gán SWdel với label del150
-------	-----	--------	------------------------------

;Chương trình

	ORG	0x000	
	GOTO	start	
start			; vị trí bắt đầu chương trình chính
	BCF	STATUS,RP1	
	BCF	STATUS,RP0	; chọn BANK0
	CLRF	PORTB	
	BSF	STATUS,RP0	; chọn BANK1
	MOVLW	b'00001111'	
	MOVWF	TRISB	
	BCF	STATUS,RP0	; chọn BANK0
loop			; vòng lặp kiểm tra công tác nào được ấn
	BTFSS	PORTB,SW1	; kiểm tra SW1
	CALL	switch1	; nhảy tới chương trình con switch1 nếu
			; SW1 được ấn
	BTFSS	PORTB,SW2	; nếu SW1 không được ấn tiếp tục kiểm tra
			; SW2
	CALL	switch2	; thao tác tương tự như SW1
	BTFSS	PORTB,SW3	
	CALL	switch3	
	BTFSS	PORTB,SW4	
	CALL	switch4	
	GOTO	loop	

switch1

CLRF	PORTB	; xóa PORTB
CALL	SWdel	; gọi chương trình delay del150
BTFSC	PORTB,SW1	; kiểm tra công tắc 1 còn nhấn hay không
RETURN		; nếu không còn nhấn thì trở về chương trình chính
led1_ON		
BSF	PORTB,LED1	; bật LED1 sáng
BTFSC	PORTB,SW1	; xác nhận lại trạng thái công tắc 1
RETURN		; trở về chương trình chính nếu công tắc không còn ấn
GOTO	led1_ON	; tiếp tục giữ LED1 sáng nếu công tắc còn được ấn
switch2		
CLRF	PORTB	
CALL	SWdel	
BTFSC	PORTB,SW2	
RETURN		
led2_ON		
BSF	PORTB,LED2	
BTFSC	PORTB,SW2	
RETURN		
GOTO	led2_ON	
switch3		
CLRF	PORTB	
CALL	SWdel	
BTFSC	PORTB,SW3	
RETURN		
led3_ON		
BSF	PORTB,LED3	
BTFSC	PORTB,SW3	
RETURN		
GOTO	led3_ON	
switch4		
CLRF	PORTB	
CALL	SWdel	
BTFSC	PORTB,SW4	
RETURN		
led4_ON		
BSF	PORTB,LED4	

```

    BTFSC    PORTB,SW4
    RETURN
    GOTO     led4_ON

```

```

;-----
;Chương trình delay cải tiến cho phép nhiều khoảng thời gian delay khác nhau
;-----

```

```
del0
```

```
    RETURN
```

```
del1
```

```
    MOVLW    d'1'
    GOTO     delay
```

```
del5
```

```
    MOVLW    d'5'
    GOTO     delay
```

```
del10
```

```
    MOVLW    d'10'
    GOTO     delay
```

```
del20
```

```
    MOVLW    d'20'
    GOTO     delay
```

```
del50
```

```
    MOVLW    d'50'
    GOTO     delay
```

```
del100
```

```
    MOVLW    d'100'
    GOTO     delay
```

```
del150
```

```
    MOVLW    d'150'
    GOTO     delay
```

```
del200
```

```
    MOVLW    d'200'
    GOTO     delay
```

```
delay
```

```
    MOVWF    count1
```

```
d1
```

```
; tạo thời gian delay 1 mS
```

```
    MOVLW    0xC7
    MOVWF    counta
    MOVLW    0x01
    MOVWF    countb
```

```
delay_0
```

```
    DECFSZ   counta,1
```



```

GOTO $+2
DECFSZ    countb,1
GOTO      delay_0
DECFSZ    count1,1
GOTO      d1
RETURN

```

END

Với chương trình trên, thời gian ấn công tắc phải lâu hơn thời gian delay được chỉ định bởi hằng số SWdel do công tắc sẽ được kiểm tra lại trạng thái sau thời gian delay . Nếu thời gian ấn công tắc không đạt yêu cầu, thao tác bật LED tương ứng với công tắc đó sáng lên sẽ không được thực hiện và vi điều khiển sẽ tiếp tục quá trình kiểm tra trạng thái các công tắc còn lại.

Thời gian delay cần được kiểm định bằng thực nghiệm và được ấn định một cách thích hợp để chống “dội” một cách hiệu quả, đồng thời cũng không được lâu quá, như vậy sẽ gây sự khó chịu trong việc sử dụng công tắc do phải ấn công tắc trong một khoảng thời gian đủ lâu.

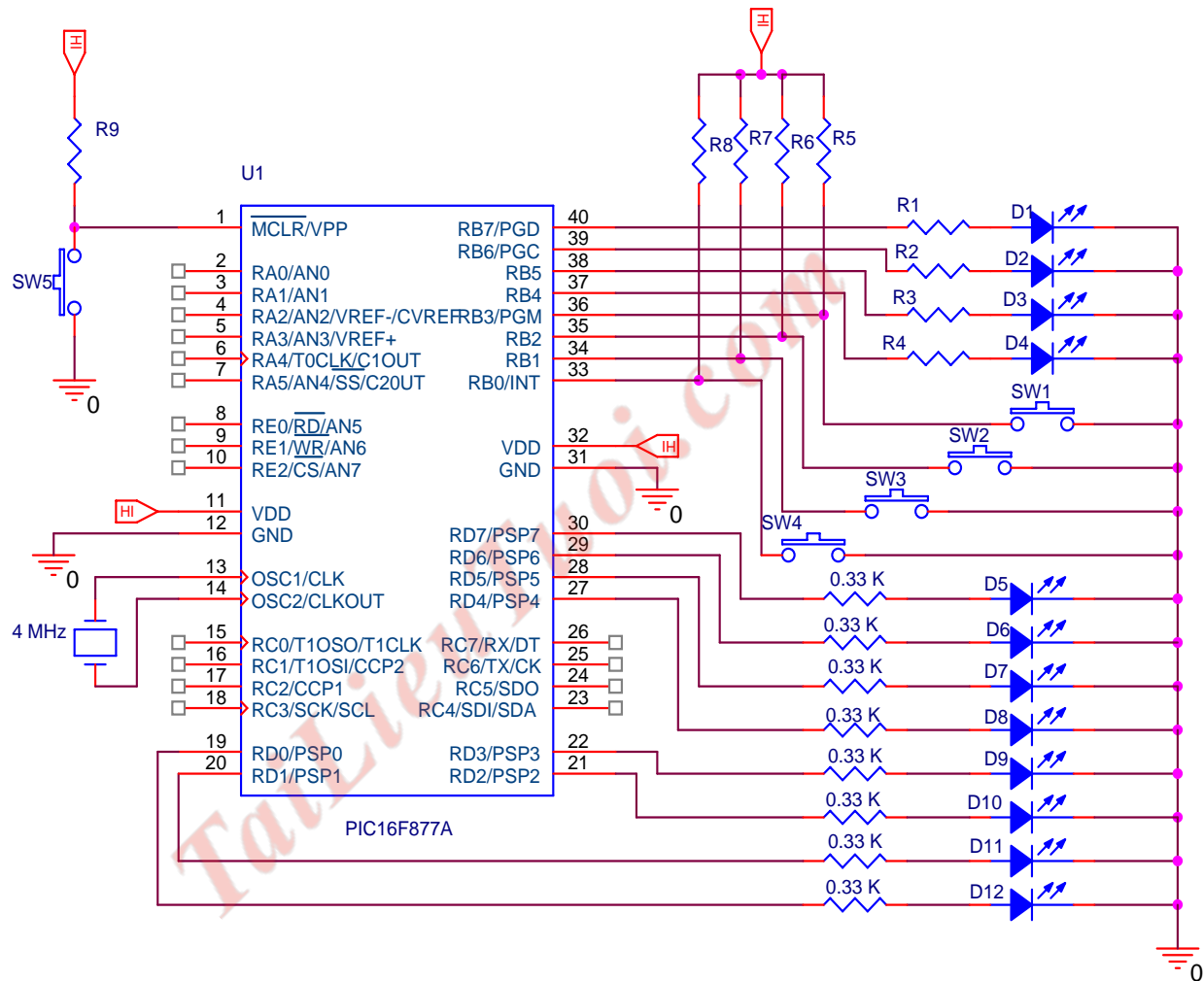
Việc thay đổi thời gian delay trong chương trình có thể được thực hiện đơn giản bằng cách thay đổi label của chương trình delay gán cho tham số SWdel. Thực ra ta có thể trực tiếp đưa tham số thời gian delay trực tiếp vào thanh ghi count1 mà không cần thông qua tham số SWdel, điều đó làm cho chương trình trở nên dài và phức tạp hơn. Tuy nhiên chương trình trên cũng đã cho ta thấy được một điểm khác biệt giữa lệnh “EQU” và lệnh “SET”, giúp ta hiểu rõ hơn và sử dụng một cách thích hợp các lệnh trên trong các ứng dụng khác.

Ứng dụng 4. 4: ứng dụng tổng hợp.

Trong ứng dụng này ta sẽ tập hợp lại tất cả các kĩ năng được sử dụng trong các ứng dụng trước. Yêu cầu đặt ra cũng như ứng dụng 3, tuy nhiên bên cạnh việc bật LED tương ứng với công tắc sáng lên, ta phải tiếp tục thực hiện một thao tác nữa là ra lệnh cho vi điều khiển hiển thị 8 LED được gán vào PORTD theo một thứ tự tương ứng. Cụ thể như sau:

Ấn SW1: LED1 sáng, 8 LED PORTD chạy từ trái sang phải (LED sáng chạy).
 Ấn SW2: LED2 sáng, 8 LED PORTD chạy từ trái sang phải (LED tắt chạy).
 Ấn SW3: LED3 sáng, 8 LED PORTD chạy từ trái sang phải (2 LED sáng chạy).
 Ấn SW4: LED4 sáng, 8 LED PORTD chạy từ trái sang phải (2 LED tắt chạy).

Để test được ứng dụng này, ta cần phát triển thêm mạch test của ứng dụng 3 bằng cách thêm vào 8 LED ở PORTD thông qua các điện trở. Cụ thể như sau:



Hình 4.3 Mạch test ứng dụng 4.

Chương trình viết cho mạch test này cũng tương tự như ứng dụng 3 nhưng được thêm vào phần hiển thị LED ở PORTD. Ta sử dụng thuật toán bảng dữ liệu để hiển thị LED. Chương trình cụ thể như sau:

;Chương trình 4.1.9

processor 16f877a

include <p16f877a.inc>

__CONFIG __CP_OFF & __WDT_OFF & __BODEN_OFF & __PWRTE_ON & __XT_OSC & __WRT_OFF & __LVP_OFF & __CPD_OFF

;Khai báo các hằng số

SW1 EQU 0

SW2 EQU 1

SW3 EQU 2

SW4	EQU	3
LED1	EQU	4
LED2	EQU	5
LED3	EQU	6
LED4	EQU	7

;Khai báo biến

count	EQU	0x20	; biến dùng cho quá trình dịch LED
count1	EQU	0x21	; các biến dùng cho chương trình delay
counta	EQU	0x22	
countb	EQU	0x23	

;Chương trình

	ORG	0x000	
	GOTO	start	
start			; vị trí bắt đầu chương trình chính
	BCF	STATUS,RP1	
	BCF	STATUS,RP0	; chọn BANK0
	CLRF	PORTB	
	CLRF	PORTD	
	BSF	STATUS,RP0	; chọn BANK1
	MOVLW	b'00001111'	
	MOVWF	TRISB	
	MOVLW	0x00	
	MOVWF	TRISD	
	BCF	STATUS,RP0	; chọn BANK0
loop1			
	CLRF	count	; reset biến count
	CALL	check_key	; gọi chương trình con check_key
loop2			
	MOVF	count,W	; đưa giá trị biến count vào thanh ghi W
	BTFSC	PORTB,LED1	; kiểm tra trạng thái bit LED1
	CALL	table1	; gọi chương trình con “table1” nếu bit “LED1” mang giá trị bằng 1
			; tiếp tục kiểm tra bit LED2 nếu bit LED1 bằng 0
	BTFSC	PORTB,LED2	; thao tác tương tự với các bit chỉ thị trạng thái các
	CALL	table2	; SW còn lại
	BTFSC	PORTB,LED3	
	CALL	table3	

BTFSC	PORTB,LED4	
CALL	table4	
MOVWF	PORTD	; đưa giá trị từ thanh ghi W sau khi quay trở về từ
		; bảng dữ liệu ra PORTD
CALL	delay	; gọi chương trình con delay
INCF	count,0	; tăng giá trị biến count để kiểm tra
XORLW	d'14'	; so sánh biến count với giá trị 14
BTFSC	STATUS,Z	; kiểm tra cờ Z (Zero)
GOTO	loop1	; nhảy tới label “loop1” nếu Z bằng 1 (giá trị
		; biến “count” bằng 14)
INCF	count,1	; tăng giá trị biến “count” nếu Z bằng 0 (giá trị
		; biến “count” không bằng 14)
GOTO	loop2	; sau đó nhảy tới label “loop2”

table1 ; các bảng dữ liệu dùng cho phần dịch LED

ADDWF	PCL,f
RETLW	b'10000000'
RETLW	b'01000000'
RETLW	b'00100000'
RETLW	b'00010000'
RETLW	b'00001000'
RETLW	b'00000100'
RETLW	b'00000010'
RETLW	b'00000001'
RETLW	b'00000010'
RETLW	b'00000100'
RETLW	b'00001000'
RETLW	b'00010000'
RETLW	b'00100000'
RETLW	b'01000000'

table2

ADDWF	PCL,f
RETLW	b'01111111'
RETLW	b'10111111'
RETLW	b'11011111'
RETLW	b'11101111'
RETLW	b'11110111'
RETLW	b'11111011'
RETLW	b'11111101'
RETLW	b'11111110'

RETLW	b'11111101'
RETLW	b'11111011'
RETLW	b'11110111'
RETLW	b'11101111'
RETLW	b'11011111'
RETLW	b'10111111'

table3

ADDWF	PCL,f
RETLW	b'11000000'
RETLW	b'01100000'
RETLW	b'00110000'
RETLW	b'00011000'
RETLW	b'00001100'
RETLW	b'00000110'
RETLW	b'00000011'
RETLW	b'00000011'
RETLW	b'00000110'
RETLW	b'00001100'
RETLW	b'00011000'
RETLW	b'00110000'
RETLW	b'01100000'
RETLW	b'11000000'

table4

ADDWF	PCL,f
RETLW	b'00111111'
RETLW	b'10011111'
RETLW	b'11001111'
RETLW	b'11100111'
RETLW	b'11110011'
RETLW	b'11111001'
RETLW	b'11111100'
RETLW	b'11111100'
RETLW	b'11111001'
RETLW	b'11110011'
RETLW	b'11100111'
RETLW	b'11001111'
RETLW	b'10011111'

check_key

; chương trình con check_key kiểm tra trạng thái

BTFSS	PORTB,SW1	; các SW, sau đó bật LED tương ứng với SW đó
CALL	switch1	; sáng nếu SW đó được ấn. Trạng thái các LED
BTFSS	PORTB,SW2	; có tác dụng như các bit cờ hiệu khi xác định thao
CALL	switch2	; tác dịch LED tương ứng với SW được ấn
BTFSS	PORTB,SW3	
CALL	switch3	
BTFSS	PORTB,SW4	
CALL	switch4	
RETURN		

switch1

CLRF	PORTB
BSF	PORTB,LED1
RETURN	

switch2

CLRF	PORTB
BSF	PORTB,LED2
RETURN	

switch3

CLRF	PORTB
BSF	PORTB,LED3
RETURN	

switch4

CLRF	PORTB
BSF	PORTB,LED4
RETURN	

delay

; chương trình delay một khoảng thời gian 250 ms

MOVLW	d'250'
MOVWF	count1

d1

MOVLW	0xC7
MOVWF	counta
MOVLW	0x01
MOVWF	countb

delay_0

DECFSZ	counta,1
GOTO	+\$+2
DECFSZ	countb,1
GOTO	delay_0

```
DECFSZ    count1,1  
GOTO      d1  
RETURN
```

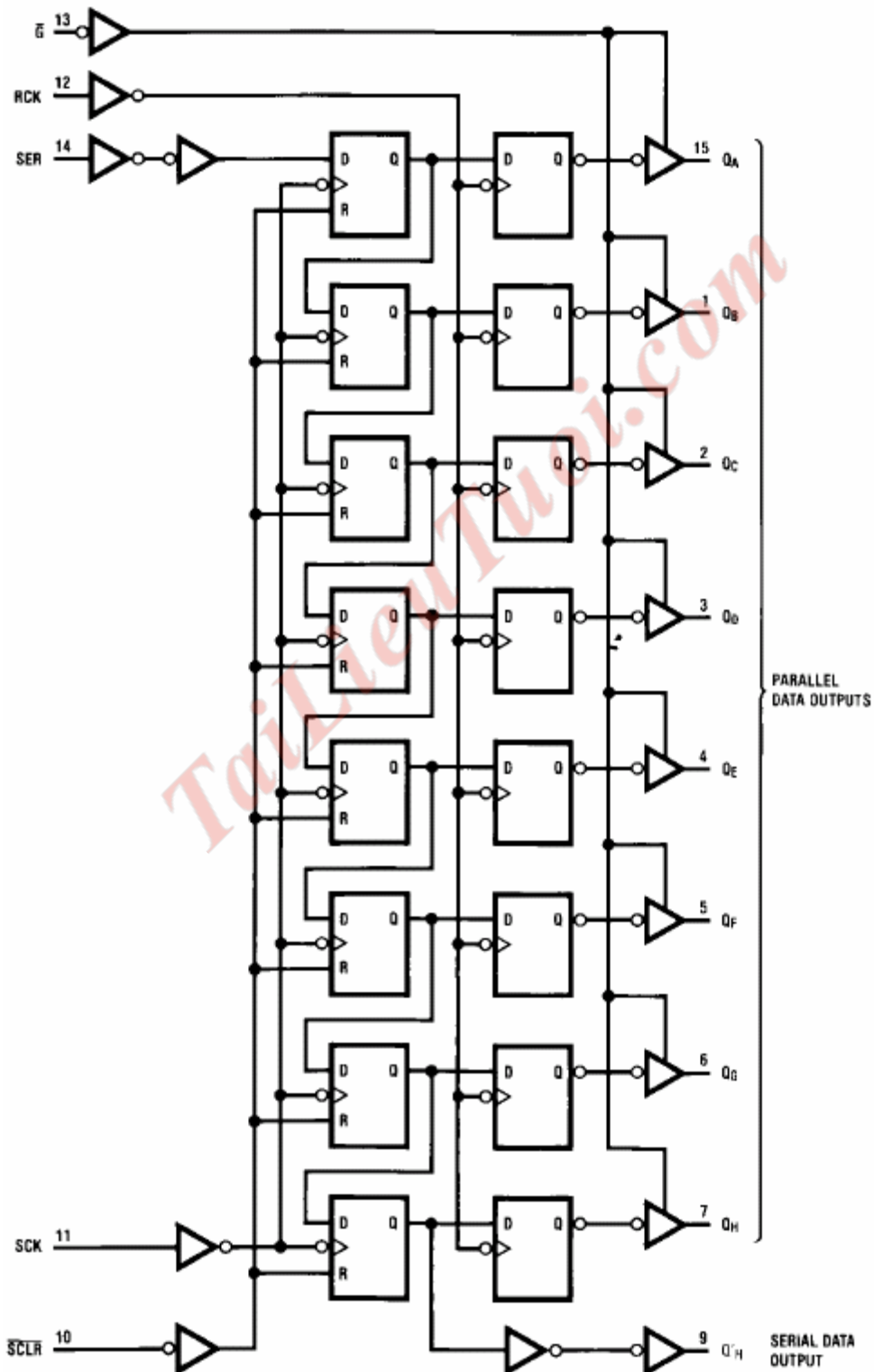
END

Trong chương trình này ta lợi dụng các bit trạng thái của các LED để dùng như các cờ hiệu để xác định thao tác dịch LED tương ứng với SW được nhấn trong vòng lặp “loop1” và “loop2”. Các thuật toán như bảng dữ liệu, kiểm tra trạng thái công tắc,... đều đã được đề cập đến ở các phần trước, vấn đề đặt ra trong chương trình này chỉ là sắp xếp và tổ chức hợp lý thứ tự các thao tác và các thuật toán. Tuy nhiên nếu đọc kỹ chương trình trên ta sẽ phát hiện một điểm bất hợp lý ở vị trí đặt lệnh “CALL check_key”. Nếu đặt ở vị trí như chương trình trên, vi điều khiển sẽ chỉ kiểm tra các SW ngay tại thời điểm kết thúc quá trình dịch LED. Như vậy muốn thay đổi thao tác quét LED ta phải ấn SW đúng ngay tại thời điểm đó, điều này gây nhiều khó khăn và tạo sự bất hợp lý so với thực tế. Để khắc phục ta chỉ việc đặt lệnh đó vào trong vòng lặp “loop2”, khi đó trạng thái các SW sẽ được cập nhật thường xuyên hơn sau mỗi lần dịch LED mà không phải chờ cho đến khi kết thúc một quá trình dịch LED.

Tới giai đoạn này xem như ta kết thúc những thao tác đơn giản nhất khi sử dụng vi điều khiển PIC16F877A. Trong phần này ta chỉ sử dụng duy nhất vi điều khiển PIC và các PORT I/O để xây dựng các ứng dụng. Kể từ phần sau ta sẽ kết hợp vi điều khiển PIC với các thiết bị ngoại vi khác để phát huy tối đa khả năng của vi điều khiển.

4.2 VI ĐIỀU KHIỂN PIC16F877A VÀ IC GHI DỊCH 74HC595

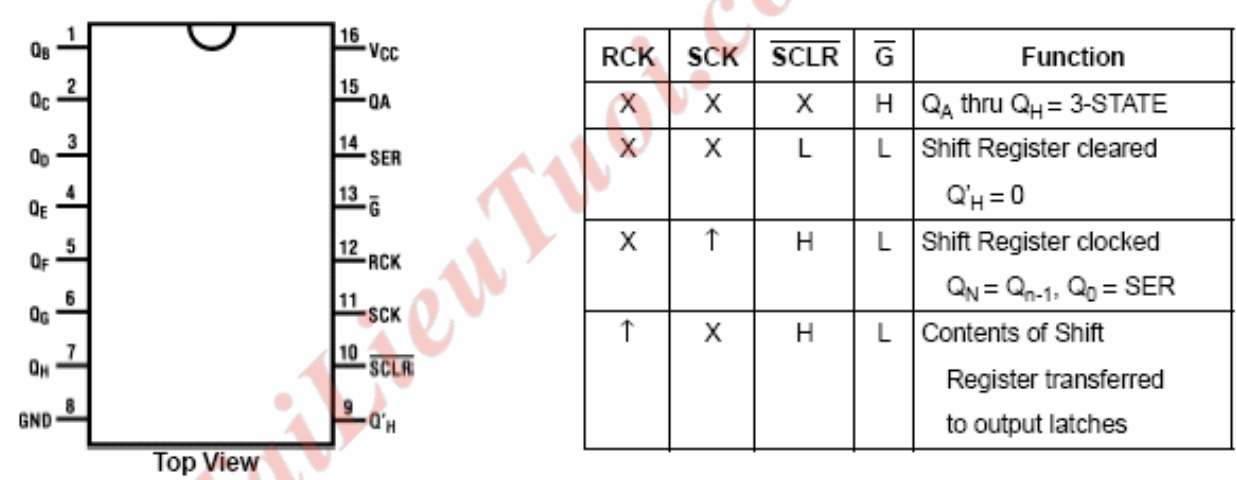
Mục đích sử dụng IC 74HC595 là nâng cao số lượng pin output của vi điều khiển. Thay vì phải truy xuất trực tiếp một giá trị nào đó ra các PORT I/O, ta có thể truy xuất gián tiếp thông qua IC 74HC595. Tuy nhiên việc trước tiên là phải tìm hiểu xem IC 74HC595 hoạt động như thế nào và cách điều khiển nó ra sao. Hình sau là sơ đồ khối của IC:



Hình 4.4 Sơ đồ khối IC 74HC595

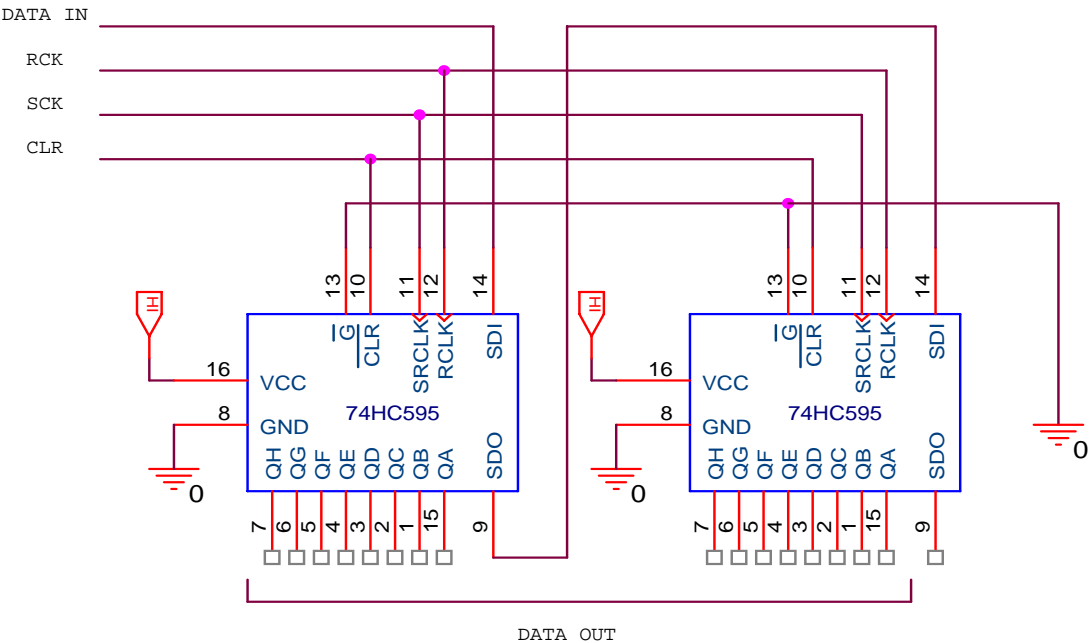
Thực chất đây là IC ghi dịch với 8 bit ngõ ra $Q_H:Q_A$ với chốt dữ liệu 8 bit. Dữ liệu chỉ được đưa vào qua 1 pin SER và được điều khiển bởi các pin RCK (pin điều khiển chốt dữ

liệu), SCK (pin điều khiển việc dịch dữ liệu vào IC thông qua các xung clock), $\overline{\text{SCLR}}$ (pin tác động mức thấp dùng để xóa dữ liệu) và pin Q_H (pin đưa dữ liệu nối tiếp ra ngoài, pin này dùng để nối nhiều IC 74HC595 lại với nhau) và pin \overline{G} (pin cho phép ngõ ra). Ta có thể điều khiển một IC 74HC595 hoặc nhiều IC ghép với nhau thông qua 4 pin RCK, SCK, SER và $\overline{\text{SCLR}}$. Điều này cho phép mở rộng một cách vô hạn số lượng pin output cho vi điều khiển, tất nhiên với một nhược điểm là thời gian truy xuất chậm do dữ liệu phải được dịch từng bit vào IC thông qua từng cạnh dương tác động vào pin SCK trước khi đưa dữ liệu ra ngoài thông qua các pin $Q_H:Q_A$. Sau đây là sơ đồ chân và bảng sự thật của IC 74HC595:



Hình 4.6 Sơ đồ chân và bảng sự thật của 74HC595

Hình sau thể hiện cách nối nhiều IC 74HC595 lại với nhau:



Hình 4.7 Cách nối nhiều IC 74HC595

Như ta thấy trong hình trên, các pin SCK, RCK và $\overline{\text{SCLR}}$ được nối chung lại với nhau, trong khi pin SDO của IC trước sẽ nối với pin SDI của IC sau. Tất cả các IC này sẽ được điều khiển thông qua 4 pin SCK, RCK, $\overline{\text{SCLR}}$ và SDI, như vậy ta có thể tiết kiệm được một số lượng đáng kể số lượng pin điều khiển của vi điều khiển.

Cách điều khiển IC được thể hiện thông qua bảng sự thật ở hình 4.6. Trước tiên đưa 1 bit dữ liệu vào pin SDI, tạo ra một cạnh dương ở pin SCK để dịch bit dữ liệu đó vào, quá trình này lặp đi lặp lại liên tục cho đến khi toàn bộ dữ liệu được dịch vào các IC 74HC595 (IC tiếp theo cùng sẽ dịch dữ liệu được đưa ra thông qua pin SDO của vi điều khiển trước). Sau đó tạo một cạnh dương ở pin RCK để đưa dữ liệu từ chốt dữ liệu ra các pin output. Ứng dụng sau giúp ta hiểu rõ hơn cách điều khiển các IC 74HC595.

Ứng dụng 4.5: IC 74HC595 và cách điều khiển.

Trong ứng dụng này ta sẽ đưa dữ liệu 8 bit bất kì ra thông qua IC 74HC595. Dữ liệu sẽ được kiểm tra thông qua các LED được gắn vào các pin output của IC. Các pin điều khiển của 74HC595 được gắn vào các pin RB3:RB0 của PORTB. Cụ thể như sau:

Pin RB0: nối với pin SDI

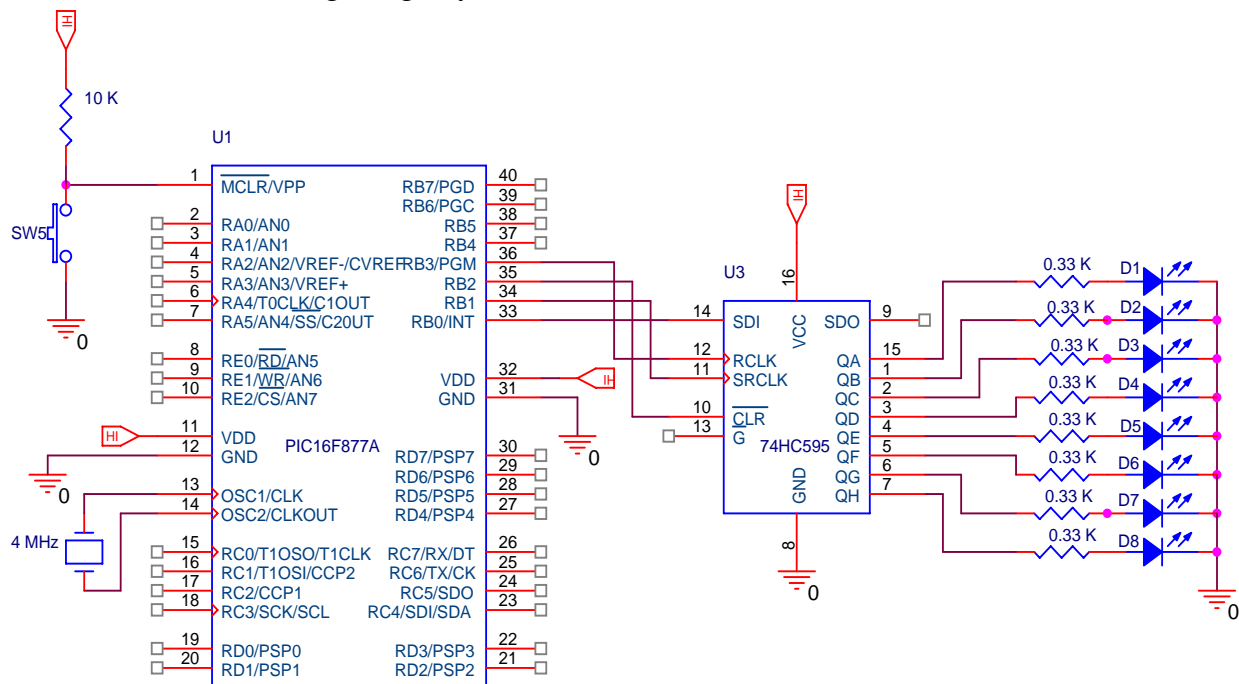
Pin RB1: nối với pin SCK

Pin RB2: nối với pin $\overline{\text{SCLR}}$

Pin RB3: nối với pin RCLK

Các thứ tự này không bắt buộc phải được tuân thủ một cách nghiêm ngặt, tùy theo mạch phần cứng mà ta có sự điều chỉnh tương ứng trong phần mềm. Ngoài ra ta có thể sử dụng bất cứ pin nào của PORT I/O nào để điều khiển IC này.

Mạch test cho ứng dụng này được thiết kế như sau:



Hình 4.8 Mạch test vi điều khiển PIC16F877A và IC 74HC595.

Sau đây là chương trình viết cho ứng dụng này:

```
; Chương trình 4.2.1
;Chương trình test IC ghi dịch 74HC595
;-----
processor    16f877a
include      <p16f877a.inc>
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
;-----
; Khai báo biến
;-----
sendreg      EQU        0X20      ; chứa dữ liệu cần xuất ra IC 74HC595
count        EQU        0X21      ; dùng để đếm số bit dữ liệu được gửi ra
;-----
;Định nghĩa phần cứng
;-----
#define       data      PORTB,0
#define       clock     PORTB,1
#define       clear     PORTB,2
#define       latch     PORTB,3
;-----
; Chương trình chính
;-----
                ORG        0x000
                GOTO       start
start           ; chương trình chính
                BCF        STATUS,RP1
                BCF        STATUS,RP0      ; chọn BANK0
                CLRF       PORTB
                BSF        STATUS,RP0      ; chọn BANK1
                MOVLW       0xF0           ; các pin RB3:RB0 là output
                MOVWF      TRISB          ; các pin RB7:RB4 là input
                BCF        STATUS,RP0      ; chọn BANK0
                MOVLW       0x04
                MOVWF      PORTB          ; đưa pin  $\overline{\text{SCLR}}$  lên mức logic cao
                BCF        clear          ; reset dữ liệu trong IC 74HC595
                NOP           ; clear tác động cạnh xuống
                BSF        clear          ; đưa pin  $\overline{\text{SCLR}}$  trở về mức logic cao
                MOVLW       0xCA          ; dữ liệu cần đưa ra IC 74HC595
```

	CALL	serout	; gọi chương trình con serout
	BSF	latch	; tạo cạnh dương tại pin RCK để đưa dữ
	NOP		; liệu ra các pin output của IC 74HC595
	BCF	latch	; đưa pin RCK trở về mức logic thấp
	GOTO	\$; chương trình bị “treo” tại đây
serout			
	MOVWF	sendreg	; đưa dữ liệu vào thanh ghi sendreg
	MOVLW	0x08	; đếm 8 bit dữ liệu
	MOVWF	count	
testbit			
	BCF	data	; dữ liệu mặc định bằng 0
	BTFSC	sendreg,7	; sendreg,7 == 0 ??
	BSF	data	; nếu không bằng 0, set dữ liệu từ 0 -> 1
	BSF	clock	
	NOP		; tạo cạnh dương tại pin SCK để dịch dữ
			; liệu vào IC 74HC595
	BCF	clock	; đưa pin SCK về lại mức logic thấp
	RLF	sendreg,0	; dịch trái thanh ghi sendreg
	MOVWF	sendreg	
	DECFSZ	count,1	; giảm biến count 1 đơn vị
	GOTO	testbit	; nếu biến count chưa bằng 0, tiếp tục quá
			; trình dịch dữ liệu
	RETURN		; trở về chương trình chính nếu count = 0
END			; kết thúc chương trình

Điểm đáng chú ý nhất của chương trình trên là thuật toán xác định giá trị bit dữ liệu cần dịch vào IC 74HC595. Ban đầu đường dữ liệu (SDI) sẽ được mặc định là mức logic 0, sau đó ta kiểm tra bit dữ liệu đó (bit thứ 7 trong thanh ghi sendreg) xem có thực sự bằng 0 hay không. Nếu bằng 1 thì ta set đường dữ liệu lên mức logic 1. Như vậy ta lần lượt kiểm tra mức logic của các bit dữ liệu cần đưa vào IC 74HC595 và set/clear đường dữ liệu SDI tương ứng với bit dữ liệu cần dịch. Việc còn lại là tạo cạnh dương tại pin SCK để đưa trạng thái logic của đường dữ liệu SDI vào trong IC 74HC595. Như vậy sau 8 lần dịch, 8 bit dữ liệu chứa trong thanh ghi sendreg đã được đưa vào thanh ghi dịch bên trong IC, và để đưa dữ liệu đó ra các pin output $Q_H:Q_A$, ta chỉ việc tạo một cạnh dương tại pin RCK, dữ liệu trong thanh ghi sendreg sẽ được thể hiện bằng các trạng thái sáng/tắt của các LED gắn vào IC 74HC595, tất nhiên với điều kiện pin \overline{G} phải được nối mass hoặc được đưa về mức logic 0.

Một điều cần lưu ý nữa là cạnh tác động của pin \overline{SCLR} . Do cạnh tác động của pin này là cạnh âm nên cần có sự điều chỉnh thích hợp để có thể điều khiển IC 74HC595 một cách đúng đắn.

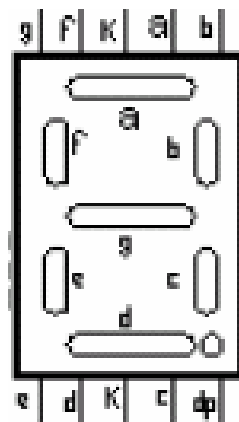
Trong trường hợp nối nhiều IC 74HC595 lại với nhau thì thuật toán hoàn toàn tương tự, tuy nhiên dữ liệu sẽ lần lượt đưa vào thanh ghi sendreg và gọi chương trình con serout. Quá trình này được lặp lại cho đến khi toàn bộ dữ liệu đã được đưa vào các IC, sau đó mới đưa dữ liệu ra ngoài bằng cách tạo một cạnh dương tại pin RCK.

4.3 PIC16F877A VÀ LED 7 ĐOẠN

LED 7 đoạn là một công cụ thông dụng được dùng để hiển thị các thông số dưới dạng các số từ 0 đến 9. Mặc dù công cụ LCD giúp ta thể hiện các thông số một cách linh động hơn nhưng LED 7 đoạn vẫn được sử dụng nhiều trong công nghiệp do các ưu thế của nó như ít chịu sự ảnh hưởng của nhiệt độ, dễ nhận ra và góc nhìn rộng.

LED 7 đoạn bao gồm 7 đoạn LED được đánh dấu là các kí tự a,b,c,d,e,f,g và một dấu chấm thập phân kí hiệu là dp. Như vậy ta có thể xem LED 7 đoạn là một tổ hợp gồm 8 LED được bố trí theo một qui tắc nhất định dùng để hiển thị các chữ số thập phân.

Có hai loại LED 7 đoạn, đó là loại Anode chung (cực Anode của các LED được nối chung với nhau) và loại Cathode chung (Cực Cathode của các LED được nối chung với nhau). Tùy theo từng loại mà ta có thể điều khiển các LED trong tổ hợp đó sáng tắt một cách thích hợp. Đối với loại Anode chung, một LED sẽ sáng nếu mức logic đưa vào pin điều khiển LED đó là mức 0. . Đối với loại Cathode chung, một LED sẽ sáng nếu mức logic đưa vào pin điều khiển LED đó là mức 1.



Hình 4.9 LED 7 đoạn.

Hình vẽ trên là một LED 7 đoạn loại Cathode chung. Thực ra cấu trúc các pin của LED 7 đoạn có thể thay đổi tùy theo loại chữ không cố định, và cách duy nhất để xác định chính xác các pin điều khiển của LED 7 đoạn là phải kiểm tra từng pin của LED đó. Dựa vào hình vẽ ta có thể hiểu được một phần nào cách hiển thị của LED 7 đoạn. Ví dụ, muốn hiển thị số 6 ta sẽ cho các đoạn LED a, c, d, e, g, f sáng và đoạn LED b tắt. Việc điều khiển sáng tắt được thực hiện bằng cách đưa dữ liệu thích hợp vào các pin a, b, c, d, e, f, g và dp của LED 7 đoạn. Đó là cách hiển thị theo từng LED, tuy nhiên trong thực tế để tiết kiệm số pin cần thiết để điều khiển một lúc nhiều LED 7 đoạn, các pin a, b, c, d, e, f, g và dp sẽ được nối song song với nhau, các pin Anode chung hoặc Cathode chung được dùng để cho phép LED 7

đoạn đó sáng hay tắt. Sở dĩ ta nối chung các pin a, b, c, d, e, f, g và dp lại với nhau được là dựa vào hiện tượng lưu ảnh của mắt. Mắt người chỉ có khả năng nhận được 24 hình ảnh trong một giây, do đó khi các LED 7 đoạn chớp tắt với một tốc độ quá nhanh như tốc độ xử lý của một vi điều khiển thì mắt người không có khả năng phát hiện ra. Bằng cách đó nếu ta lần lượt cho từng LED 7 đoạn sáng trong một khoảng thời gian rất ngắn nào đó thì mắt người sẽ bị “đánh lừa” rằng tất cả các LED đang sáng cùng một lúc.

Để hiểu thêm về cách hiển thị và thuật toán dùng để hiển thị LED 7 đoạn, ta sẽ thực hiện một ứng dụng đơn giản là hiển thị 2 LED 7 đoạn.

Ứng dụng 4.6: Hiển thị LED 7 đoạn.

Trong ứng dụng này ta sẽ hiển thị một số có 2 chữ số trên 2 LED.

Loại LED 7 đoạn ta sẽ sử dụng là loại Anode chung. Trước hết ta cần xác định trước sơ đồ nối chân giữa vi điều khiển và các LED 7 đoạn để từ đó xác định được dữ liệu cần đưa vào để điều khiển LED 7 đoạn hiển thị một chữ số thập phân nào đó. Giả sử ta nối các pin dữ liệu của LED 7 đoạn vào PORTD của PIC16F877A theo thứ tự như sau:

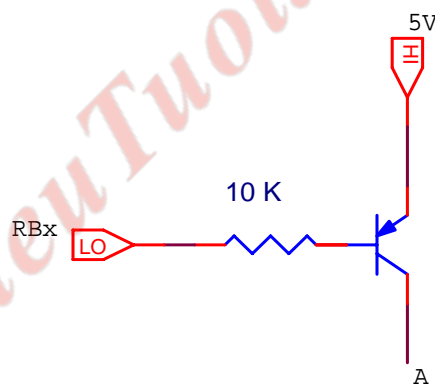
Pin dp nối vào pin RD7
 Pin g nối vào pin RD6
 Pin f nối vào pin RD5
 Pin e nối vào pin RD4
 Pin d nối vào pin RD3
 Pin c nối vào pin RD2
 Pin b nối vào pin RD1
 Pin a nối vào pin RD0

Các pin Anode chung của LED 7 đoạn sẽ được nối vào các pin RB0 và RB1 của PORTB. Như vậy muốn điều khiển một đoạn LED nào đó sáng đưa pin điều khiển đoạn LED tương ứng về mức logic 0. Với cách nối chân như vậy ta có bảng dữ liệu tương ứng với các chữ số cần hiển thị trên LED 7 đoạn như sau:

Chữ số	RB7 (dp)	RB6 (g)	RB5 (f)	RB4 (e)	RB3 (d)	RB2 (c)	RB1 (b)	RB0 (a)	Mã Hex
0	1	1	0	0	0	0	0	0	C0h
1	1	1	1	1	1	0	0	1	F9h
2	1	0	1	0	0	1	0	0	A4h
3	1	0	1	1	0	0	0	0	B0h
4	1	0	0	1	1	0	0	1	99h
5	1	0	0	1	0	0	1	0	92h
6	1	0	0	0	0	0	1	0	82h
7	1	1	1	1	1	0	0	0	F8h
8	1	0	0	0	0	0	0	0	80h
9	1	0	0	1	0	0	0	0	90h

Dựa vào bảng dữ liệu trên, muốn hiển thị một chữ số thập phân nào đó ra LED 7 đoạn, ta chỉ việc đưa mã hex của chữ số đó ra PORTD của vi điều khiển. Một điểm cần lưu ý là bảng mã trên không cố định mà nó phụ thuộc nhiều vào cấu trúc phần cứng của mạng điều khiển, do đó tùy theo cách kết nối phần cứng mà ta có được bảng mã tương ứng.

Đến đây xem như ta đã hoàn tất quá trình chuyển đổi dữ liệu từ dạng thập phân sang dạng mã của LED 7 đoạn. Việc còn lại là làm sao để cho phép một LED nào đó trong dãy LED 7 đoạn mắc song song tắt hoặc sáng lên. Một giải pháp đơn giản là sử dụng các BJT hoạt động với chức năng như là các công tắc đóng mở để cho phép hoặc không cho phép nguồn cung cấp đưa vào LED 7 đoạn, các “công tắc” này sẽ được điều khiển bởi các pin trong PORTB. Sơ đồ nguyên lý của các “công tắc” này khi dùng để điều khiển LED 7 đoạn loại Anode chung như sau:



Hình 4.9 Sơ đồ nguyên lý “công tắc” điều khiển cấp nguồn cho LED 7 đoạn Anode chung

Ta nối pin của Port điều khiển vào cực B của BJT loại pnp thông qua một điện trở, giá trị điện trở này phụ thuộc vào khả năng chịu dòng tối đa của LED 7 đoạn. Cực E của BJT được nối lên nguồn 5V và cực C được đưa vào pin V_{CC} của LED 7 đoạn. Khi pin điều khiển ở mức logic 1, do V_{EB} của BJT bằng 0 nên BJT không dẫn, do đó không có dòng điện đi qua LED. Khi Port điều khiển ở mức logic 0, dòng đi từ cực E sang cực B của BJT làm cho BJT dẫn bão hòa, trở kháng của BJT xem như bằng 0 và LED 7 đoạn xem như được nối trực tiếp với nguồn cung cấp.

Với ứng dụng trên, chẳng hạn ta sử dụng 2 pin RB0 (điều khiển LED hàng đơn vị) và RB1 (điều khiển LED hàng chục) để điều khiển cấp nguồn cho các LED. Khi đó chương trình được viết như sau:

;Chương trình 4.3.1

; Chương trình hiển thị một số có hai chữ số cho trước ra LED 7 đoạn

```
processor    16f877a
include      <p16f877a.inc>
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```



```

count1    EQU        0x20        ;các tham số sử dụng cho chương trình con
counta    EQU        0x21        ; delay_1ms
countb    EQU        0x22

ORG       0x000
GOTO      start

start
BCF        STATUS, RP1
BSF        STATUS, RP0        ; chọn BANK1
MOVLW     0x00
MOVWF     TRISD                ; PORTD <- output
MOVLW     0x00
MOVWF     TRISB                ; PORTB <- output
BCF        STATUS, RP0        ; chọn BANK0
CLRF      PORTB
CLRF      PORTD

loop
MOVLW     0x99                ; hiển thị số 4
MOVWF     PORTD
MOVLW     b'11111101'        ; cấp nguồn cho LED hàng chục
MOVWF     PORTB
CALL      delay_1ms          ; gọi chương trình con delay_1ms

MOVLW     0x92                ; hiển thị số 5
MOVWF     PORTD
MOVLW     b'11111110'        ; cấp nguồn cho LED hàng đơn vị
MOVWF     PORTB
CALL      delay_1ms          ; gọi chương trình con delay_1ms

GOTO      loop                ; lặp lại các thao tác trên

delay_1ms                                ; chương trình con delay_1ms
MOVLW     d'1'
MOVWF     count1
d1    MOVLW     0xC7
MOVWF     counta
MOVLW     0x01
MOVWF     countb
delay_0
DECFSZ    counta, 1
GOTO      $+2

```

```

DECFSZ    countb,1
GOTO      delay_0
DECFSZ    count1,1
GOTO      d1
RETURN

```

END

Như vậy trong chương trình trên, mỗi LED 7 đoạn sẽ lần lượt được bật sáng trong khoảng thời gian 1 ms, sau đó LED khác được bật lên trong khoảng thời gian 1 ms. Thao tác này được lặp đi lặp lại trong vòng lặp “loop”. Thực chất là các LED sẽ chớp tắt liên tục mỗi khoảng thời gian 1ms, nhưng do thời gian chớp tắt quá nhanh nên mắt người bị “đánh lừa” là cả hai LED đang sáng cùng một lúc. Hiện tượng này có thể nhận biết rõ ràng hơn bằng cách tăng thời gian delay đến một mức nào đó mà mắt người có thể nhận biết được, khi đó ta sẽ thấy rõ ràng là từng LED một được bật tắt một cách tuần tự.

Tương tự ta có thể mở rộng số lượng LED bằng cách nối song song tất cả chúng lại với nhau và áp dụng thuật toán trên để hiển thị.

Bây giờ ta thử giải quyết một trường hợp phức tạp hơn. Trong ví dụ trên ta tự ấn định chữ số hiển thị, tuy nhiên nếu chữ số cần hiển thị (chữ số hàng chục và hàng đơn vị) được chứa trong một thanh ghi nào đó thì sẽ có một số vấn đề phát sinh như sau:

Thứ nhất, do dữ liệu trong thanh ghi được lưu dưới dạng dữ liệu 8 bit nên chữ số hàng chục sẽ được chứa trong 4 bit cao và chữ số hàng đơn vị sẽ được chứa trong 4 bit thấp. Vậy làm thế nào để tách được chữ số hàng chục và hàng đơn vị chứa trong thanh ghi?? Một thuật toán đơn giản là sử dụng phép toán AND. Dữ liệu dạng nhị phân sẽ giữ nguyên giá trị khi ta thực hiện phép toán AND với 1 và sẽ được xóa về 0 nếu thực hiện phép toán AND với giá trị 0. Bằng cách đó muốn tách 4 bit thấp, ta “AND” 4 bit cao với 0, 4 bit thấp với 1 và ngược lại đối với trường hợp cần tách 4 bit cao.

Thứ hai, do dữ liệu được lưu dưới dạng mã HEX, bao gồm các chữ số từ 0 đến 9 và các kí tự từ 0 đến A. Vậy làm thế nào để chuyển đổi dữ liệu từ dạng mã HEX về dạng thập phân?? (cần lưu ý là tập lệnh dành cho PIC không có phép toán chia lấy phần dư hay phần nguyên DIV và MOD cũng như các phép toán so sánh). Phương pháp đơn giản nhất là so sánh với từng chữ số HEX và áp dụng phép chuyển đổi đối với từng trường hợp.

Và cuối cùng, làm sao hiển thị chữ số thập phân trên ra LED 7 đoạn?? Như vậy ta cần tiến hành một bước nữa là chuyển đổi từ mã thập phân sang mã LED 7 đoạn, và một phương pháp ta vẫn thường sử dụng trong các ứng dụng trước là phương pháp bảng dữ liệu sẽ tiếp tục được sử dụng trong chương trình này. Giá trị của chữ số cần chuyển đổi sẽ được cộng vào thanh ghi PCL để mang mã chuyển đổi tương ứng từ bảng dữ liệu trở về.

Bây giờ ta thử viết chương trình thực hiện thao tác trên theo các giải thuật đã đề ra. Chương trình sẽ được viết như sau:

; Chương trình 4.3.2

; Chương trình hiển thị số có hai chữ số được lưu trong một thanh ghi

```
processor    16f877a
include      <p16f877a.inc>
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```

```
count1      EQU      0x20      ; dùng cho chương trình delay_1ms
counta      EQU      0x21      ; dùng cho chương trình delay_1ms
countb      EQU      0x22      ; dùng cho chương trình delay_1ms
display_reg EQU      0x23      ; chứa giá trị cần hiển thị
hang_chuc   EQU      0x24      ; chứa hàng chục của thanh ghi display_reg
hang_don_vi EQU      0x25      ; chứa hàng đơn vị của thanh ghi
                                ; display_reg
xx          EQU      0x26      ; dùng cho chương trình con “chuyen_ma”
xx1         EQU      0x27
```

```
ORG         0x000
```

```
GOTO        start
```

```
start      ; bắt đầu chương trình(
```

```
BCF         STATUS, RP1
```

```
BSF         STATUS, RP0      ; chọn BANK1
```

```
MOVLW      0x00
```

```
MOVWF       TRISD            ; PORTD <- output
```

```
MOVLW      0x00
```

```
MOVWF       TRISB            ; PORTB <- output
```

```
BCF         STATUS, RP0      ; chọn BANK0
```

```
MOVLW      b'11111111'
```

```
MOVWF       PORTB            ; tắt tất cả các LED
```

```
CLRF        PORTD
```

```
MOVLW      0x5F              ; giá trị cần hiển thị
```

```
MOVWF       display_reg
```

```
ANDLW      0x0F              ; tách chữ số hàng đơn vị
```

```
MOVWF       hang_don_vi
```

```
MOVLW      0xF0
```

```
ANDWF       display_reg, 0
```

```
MOVWF       hang_chuc        ; tách chữ số hàng chục
```

```
SWAPF       hang_chuc, 1
```

	MOVF	hang_don_vi,0	
	CALL	chuyenma	; gọi chương trình con dùng để chuyển từ ; mã HEX sang mã thập phân
	MOVWF	hang_don_vi	; lưu giá trị sau khi chuyển đổi
	BTFSC	xx1,0	; kiểm tra xem giá trị cần chuyển đổi có ; lớn hơn 10 hay không
	INCF	hang_chuc,1	; nếu đúng, tăng hàng chục 1 đơn vị
	MOVF	hang_chuc,0	; nếu không tiếp tục chuyển mã chữ số ; hàng chục
	CALL	chuyen_ma	
Loop	MOVWF	hang_chuc	; lưu lại giá trị sau khi chuyển đổi ; đoạn chương trình hiển thị kết quả
	MOVF	hang_don_vi,0	; chuyển đổi ra LED 7 đoạn
	CALL	table	
	MOVWF	PORTD	
	MOVLW	b'11111110'	
	MOVWF	PORTB	
	CALL	delay_1ms	
	MOVF	hang_chuc,0	
	CALL	table	
	MOVWF	PORTD	
	MOVLW	b'11111101'	
	MOVWF	PORTB	
	CALL	delay_1ms	
	GOTO	loop	
chuyen_ma			; chương trình con chuyển từ mã HEX sang ; dạng mã thập phân
	MOVWF	xx	; lưu số cần chuyển đổi vào thanh ghi xx
	MOVLW	0x00	; so sánh với số 0
	XORWF	xx,0	
	BTFSC	STATUS,Z	
	GOTO	nho_hon_10	; nếu bằng 0, nhảy tới label “nho_hon_10”
	MOVLW	0x01	; nếu không bằng 0, tiếp tục so sánh với 1
	XORWF	xx,0	; tiếp tục tiến hành so sánh với các chữ
	BTFSC	STATUS,Z	; số tiếp theo
	GOTO	nho_hon_10	
	MOVLW	0x02	

XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x03
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x04
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x05
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x06
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x07
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x08
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x09
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x0A
XORWF xx,0

	BTFSC	STATUS,Z	
	GOTO	bang_10	
	MOVLW	0x0B	
	XORWF	xx,0	
	BTFSC	STATUS,Z	
	GOTO	bang_11	
	MOVLW	0x0C	
	XORWF	xx,0	
	BTFSC	STATUS,Z	
	GOTO	bang_12	
	MOVLW	0x0D	
	XORWF	xx,0	
	BTFSC	STATUS,Z	
	GOTO	bang_13	
	MOVLW	0x0E	
	XORWF	xx,0	
	BTFSC	STATUS,Z	
	GOTO	bang_14	
	MOVLW	0x0F	
	XORWF	xx,0	
	BTFSC	STATUS,Z	
	GOTO	bang_15	
nho_hon_10			; xử lí trường hợp nhỏ hơn 10
	MOVLW	0x00	; bit 0 của thanh ghi xx1 mang giá trị 0
	MOVWF	xx1	
	MOVF	xx,0	; lưu giá trị sau chuyển đổi chứa trong
			; thanh ghi xx vào thanh ghi W
	RETURN		; trở về chương trình chính
bang_10			
	MOVLW	0x01	; bit 0 của thanh ghi xx1 mang giá trị 1
	MOVWF	xx1	; để báo hiệu cần tăng giá trị hàng tiếp theo
	RETLW	0x00	; mang giá trị chuyển đổi tương ứng trở về
			; chương trình chính thông qua thanh ghi W
bang_11			; thao tác tương tự với các trường hợp còn lại
	MOVLW	0x01	
	MOVWF	xx1	

```
    RETLW    0x01
bang_12
```

```
    MOVLW    0x01
    MOVWF    xx1
    RETLW    0x02
```

```
bang_13
    MOVLW    0x01
    MOVWF    xx1
    RETLW    0x03
```

```
bang_14
    MOVLW    0x01
    MOVWF    xx1
    RETLW    0x04
```

```
bang_15
    MOVLW    0x01
    MOVWF    xx1
    RETLW    0x05
```

```
Table                                ; tra bảng dữ liệu để chuyển đổi từ mã thập phân
    ADDWF    PCL,1                  ; sang mã LED 7 đoạn
    RETLW    0xC0
    RETLW    0xF9
    RETLW    0xA4
    RETLW    0xB0
    RETLW    0x99
    RETLW    0x92
    RETLW    0x82
    RETLW    0xF8
    RETLW    0x80
    RETLW    0x90
```

```
delay_1ms                            ; chương trình con tạo thời gian delay 1ms
```

```
    MOVLW    d'1'
    MOVWF    count1
```

```
d1    MOVLW    0xC7
    MOVWF    counta
    MOVLW    0x01
    MOVWF    countb
```

```
delay_0
    DECFSZ   counta,1
    GOTO $+2
    DECFSZ   countb,1
```

```

GOTO delay_0
DECFSZ     count1,1
GOTO d1
RETURN

```

END

Trong chương trình con “chuyen_ma”, ta lần lượt so sánh giá trị sau khi tách từ thanh ghi “display_reg” thành hàng chục (chứa trong thanh ghi “hang_chuc”) và hàng đơn vị (chứa trong thanh ghi “hang_don_vi”) so sánh với từng giá trị từ 0 đến 15. Nếu số cần chuyển mã nhỏ hơn 10, ta chỉ việc giữ nguyên giá trị và trở về chương trình chính. Nếu số cần chuyển mã có giá trị lớn hơn hoặc bằng 10, ta đưa giá trị cần chuyển vào thanh ghi W thông qua lệnh RETLW và thiết lập một “cờ hiệu” nào đó do ta tự tạo để báo hiệu rằng chữ số cần chuyển đổi có giá trị lớn hơn 10 (ở đây là bit 0 chứa trong thanh ghi “xx1” để báo hiệu rằng cần tăng giá trị hàng tiếp theo lên 1 đơn vị). Chương trình chính sẽ có đoạn chương trình xử lý “cờ hiệu” này để cho ra các chữ số thập phân thích hợp ứng với các chữ số HEX. Công việc còn lại là chuyển đổi từ số thập phân sang mã LED 7 đoạn thông qua bảng dữ liệu và hiển thị kết quả ra các LED.

Như vậy trong mục này ta đã thực hiện được một số thao tác, chương trình và giải thuật cơ bản đối với LED 7 đoạn và cách hiển thị trên LED. Các thao tác bao gồm cách hình thành bảng dữ liệu, cách kết nối LED 7 đoạn và phương pháp hiển thị. Các giải thuật bao gồm các cách chuyển đổi từ mã HEX sang mã thập phân, từ mã thập phân sang mã LED 7 đoạn và cách tách chữ số hàng chục và hàng đơn vị chứa trong một thanh ghi bất kì. Từ các thao tác cơ bản này ta có thể phát triển thành nhiều ứng dụng phức tạp hơn cho vi điều khiển khi làm việc với LED 7 đoạn, đặc biệt là các ứng dụng cần hiển thị kết quả dưới dạng số. Ta sẽ tiếp tục bàn đến các ứng dụng này trong phần tiếp theo khi đề cập đến các TIMER.

4.4 NGẮT VÀ CẤU TRÚC CỦA MỘT CHƯƠNG TRÌNH NGẮT

Ngắt và các loại ngắt đã được trình bày cụ thể trong chương 2. Ở đây ta chỉ tóm tắt lại một số đặc điểm quan trọng của ngắt và thông tin mang tính ứng dụng.

Có thể nói đây là một khái niệm mang tính trừu tượng cao nhưng cũng được thiết lập dựa trên các hiện tượng và tình huống có thực trong thực tế. Chẳng hạn như trong cuộc sống hằng ngày, đôi khi ta phải tạm ngưng một công việc nào đó để làm một công việc khác cần thiết hơn, chẳng hạn như tạm ngưng một công việc nào đó đang làm để nghe điện thoại. Sự tạm ngưng này cần được báo hiệu bởi một tín hiệu (trong trường hợp trên là chuông điện thoại chẳng hạn) và phải được ta cho phép trước đó (nếu ta không cho phép điện thoại reo thì điện thoại sẽ không reo). Từ ví dụ thực tế trên ta có thể liên tưởng đến ngắt và cách xử lý ngắt của một vi điều khiển. Một ngắt là một tín hiệu điều khiển bắt buộc vi điều khiển tạm ngưng công việc đang làm để tiến hành các thao tác mà ngắt đó qui định thông qua chương trình ngắt. Tín hiệu điều khiển này được báo hiệu bởi cờ ngắt (tương ứng với chuông điện

thoại ở ví dụ trên) và phải được ta cho phép trước đó thông qua các bit điều khiển cho phép hoặc không cho phép ngắt. Một chương trình ngắt thông thường sẽ được tách riêng với chương trình chính để bảo đảm tính độc lập của chương trình ngắt.

Đối với vi điều khiển PIC16F877A, khi một ngắt (đã được cho phép trước đó) xảy ra thì “phản ứng” của nó là quay về địa chỉ 0004h và thực hiện các lệnh bắt đầu tại địa chỉ này. Thông thường đối với chương trình viết cho vi điều khiển PIC, chương trình ngắt sẽ được đặt tại đây và chương trình chính sẽ được bắt đầu ở một địa chỉ cách đó một đoạn “an toàn” sao cho chương trình chính và chương trình ngắt không bị chồng lên nhau. Nếu ta sử dụng trình biên dịch MPLAB, trình biên dịch sẽ báo lỗi khi hiện tượng trên xảy ra và ta có thể khắc phục bằng cách dời chương trình chính đi một đoạn xa hơn.

Một điểm cần lưu ý nữa là trong quá trình thực hiện chương trình ngắt, nội dung của một số thanh ghi quan trọng có khả năng bị thay đổi (thanh ghi W chẳng hạn). Do đó trước khi thực hiện chương trình ngắt ta cần thực hiện một thao tác là “cất” một số thanh ghi quan trọng vào một vài ô nhớ nào đó và phải trả lại giá trị ban đầu cho các thanh ghi đó trước khi thoát khỏi chương trình ngắt bằng lệnh RETFIE.

Nếu sử dụng trình biên dịch MPLAB, cấu trúc chương trình này đã được viết sẵn, ta chỉ việc đưa chương trình ngắt và chương trình chính vào các vị trí thích hợp được chú thích trong chương trình, tuy nhiên dựa vào các nhận định như trên ta hoàn toàn có thể tự định ra một cấu trúc chương trình cho riêng mình như sau:

```
;-----  
; Một số thông tin cần ghi chú về chương trình  
;-----  
TITLE "tên chương trình"  
processor    16f877a  
include      <p16f877a.inc>  
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &  
_WRT_OFF & _LVP_OFF & _CPD_OFF  
  
; Ví dụ về cách khai báo một vi điều khiển  
;-----  
; Định nghĩa phần cứng  
;-----  
; định nghĩa các chân xuất nhập để dễ dàng sử dụng  
; ví dụ  
#DEFINE     DEN1          PORTB,0  
#DEFINE     DEN2          PORTB,1  
#DEFINE     DEN3          PORTB,2
```

```

;-----
; Định nghĩa các biến, các thanh ghi các tham số
;-----
;Nên đặt tất cả các biến trong BANK0
; ví dụ

ORG 0x020

REGAD1      RES      1      ; biến này có độ lớn 1 byte, địa chỉ
                        ; bắt đầu là 0x20
Variables    RES      32     ; biến này có độ lớn 32 byte, địa chỉ
                        ; bắt đầu là 0x21
;-----
; Chương trình ngắt
;-----
; Chú ý là tuyệt đối không thay đổi cấu trúc đoạn chương trình bắt đầu và thoát ra
; khởi chương trình ngắt
; Nếu thay đổi chương trình sẽ chạy không đúng

ORG 0X0004
;-----
; Bắt đầu chương trình ngắt
; Đoạn chương trình bắt buộc và không được thay đổi
; có tác dụng lưu lại một số thanh ghi quan trọng
;-----
MOVWF    W_SAVE      ; W_SAVE(bank unknown!) = W
SWAPF    STATUS,W
CLRF     STATUS      ; force bank 0 for remainder of handler
MOVWF    STAT_SV     ; STAT_SV = swap_nibbles( STATUS )
                        ; STATUS = 0

MOVF     PCLATH,W
MOVWF    PCH_SV      ; PCH_SV = PCLATH
CLRF     PCLATH      ; PCLATH = 0
MOVF     FSR,W
MOVWF    FSR_SV      ; FSR_SV = FSR
                        ; 12 cycles from interrupt to here!
;-----
; Đoạn chương trình ngắt bắt đầu tại đây
;-----
; Kiểm tra xem ngắt nào đã xảy ra

```

; Xóa cờ ngắt trước khi thực hiện các lệnh trong ngắt
; Bắt đầu các lệnh cho chương trình ngắt

; Kết thúc chương trình ngắt
; Đoạn chương trình bắt buộc và không được thay đổi
; có tác dụng phục hồi giá trị ban đầu cho một số thanh ghi quan trọng

ENDINT

MOVF FSR_SV,W
MOVWF FSR ; FSR = FSR_SV
MOVF PCH_SV,W
MOVWF PCLATH ; PCLATH = PCH_SV
SWAPF STAT_SV,W
MOVWF STATUS ; STATUS = swap_nibbles(STAT_SV)
SWAPF W_SAVE,F
SWAPF W_SAVE,W ; W = swap(swap(W_SAVE)) (no change Z bit)

RETFIE ; RETURN!
; 1 cycle to resumption of code (branch penalty)

; Chấm dứt chương trình ngắt
; Bắt đầu các bước khởi tạo cho toàn bộ chương trình

ORG 0X0000
GOTO START
ORG 0X0050
; Phải cách ra một đoạn để tránh đè lên chương trình ngắt
START

; Khởi tạo các PORT
; Khởi tạo các biến
; Khởi tạo các khối chức năng (Timer, CCP, PWM,.....)

; Bắt đầu vòng lặp chính

MAIN

; Các thao tác trong vòng lặp chính
; Các chương trình con

END

So với các chương trình trước đây thì bắt đầu từ giai đoạn này, các chương trình sẽ trở nên phức tạp hơn về cấu trúc cũng như chức năng do có thêm chương trình ngắt. Tuy nhiên ta sẽ dễ dàng làm quen với cấu trúc mới này sau một vài chương trình đơn giản có liên quan đến ngắt. Ta sẽ bắt đầu với Timer và các ngắt của Timer.

4.5 TIMER VÀ ỨNG DỤNG

Như ta đã biết PIC16F877A có 3 bộ định thời là Timer0, Timer1 và Timer2. Mỗi Timer có một cấu trúc và chức năng riêng tùy thuộc vào mục đích sử dụng. Có thể phân chia một cách tương đối mục đích sử dụng của một Timer như sau:

Tác dụng định thời (Timing): các Timer sẽ sử dụng xung clock đồng bộ được cung cấp bởi oscillator của vi điều khiển hoặc từ một oscillator cố định RC0/T1OSO/T1CKI và RC1/T1OSICCP2 đối với Timer1. Giá trị đếm chứa trong thanh ghi của các Timer sẽ tăng tuần tự sau một khoảng thời gian tuần tự được định trước dựa vào các thông số của prescaler, postscaler, chu kỳ lệnh và các giá trị định trước được đưa vào các thanh ghi chứa giá trị đếm của các Timer. Đây cũng là lý do tại sao ta nói Timer có tác dụng định thời vì dựa vào giá trị đếm của các Timer, ta có thể xác định một cách tương đối chính xác thời gian thực.

Tác dụng đếm (Counting): các Timer sẽ lấy xung đếm từ bên ngoài. Các xung đếm này có tác dụng phản ánh một hiện tượng nào đó từ thế giới bên ngoài và thông qua việc đếm các xung clock đó, ta có thể xác định được số lần một hiện tượng nào đó xảy ra, từ đó ấn định các thao tác tương ứng đối với hiện tượng đó.

Thông thường các thao tác đối với Timer dựa vào các ngắt và chương trình ngắt. Ta cần xem lại cấu trúc một chương trình ngắt được trình bày ở phần trước để quá trình viết chương trình cho Timer trở nên thuận lợi hơn. Bên cạnh đó cách thiết lập các chế độ hoạt động đối với mỗi Timer cũng khác nhau. Vấn đề này sẽ được trình bày cụ thể trong từng chương trình ứng dụng, ngoài ra có thể tham khảo thêm một số tài liệu của nhà sản xuất Microchip để biết thêm chi tiết.

4.5.1 TIMER VÀ HOẠT ĐỘNG ĐỊNH THỜI

Trong phần này ta sẽ làm bước đầu làm quen với các Timer của vi điều khiển PIC16F877A và các thao tác cơ bản đối với các Timer, bao gồm thao tác khởi tạo và xử lý ngắt. Để cụ thể hơn ta sẽ đi sâu vào ứng dụng sau:

Ứng dụng 4.7: Hiển thị các giá trị định thời của Timer ra LED 7 đoạn.

Ứng dụng này được phát triển dựa trên ứng dụng 4.6 về hiển thị trên LED 7 đoạn. Ở ứng dụng 4.6 ta đã làm quen với các thao tác cơ bản đối với LED 7 đoạn. Trong ứng dụng này ta sẽ dùng các Timer để hiển thị các giá trị tăng dần từ 0 đến 99 sau một khoảng thời

gian định trước trên 2 LED hàng chục và hàng đơn vị. Cấu trúc phần cứng vẫn không có gì thay đổi, tuy nhiên về chương trình sẽ có những thay đổi đáng kể.

Trước hết là giải thuật cho ứng dụng trên. Ta sẽ khởi tạo Timer để hình thành thời gian delay cố định. Thời gian delay sẽ kết thúc bằng một tín hiệu từ ngắt Timer, chương trình ngắt có nhiệm vụ cập nhật giá trị đếm mỗi khi ngắt xảy ra, chương trình chính có tác dụng hiển thị các giá trị đã được cập nhật ra LED 7 đoạn. Trước tiên ta sẽ sử dụng Timer0 cho ứng dụng trên và chương trình cụ thể như sau:

```
;-----  
; Ghi chú về chương trình  
;-----  
; Chương trình 4.5.1  
; Chương trình hiển thị số đếm trên hai LED 7 đoạn theo thứ tự tăng dần  
; Timer sử dụng: Timer2  
;-----  
; Khai báo vi điều khiển  
;-----  
processor    16f877a  
include      <p16f877a.inc>  
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &  
_WRT_OFF & _LVP_OFF & _CPD_OFF  
;-----  
; Khai báo biến  
;-----  
count1      EQU      0x20      ; Các thanh ghi sử dụng cho chương  
counta      EQU      0x21      ; trình delay  
countb      EQU      0x22  
  
hang_don_vi EQU      0x23      ; Các thanh ghi chứa giá trị cần  
hang_chuc   EQU      0x24      ; hiển thị ra LED 7 đoạn  
  
W_save      EQU      0x25      ; Các thanh ghi dùng để cất các  
PCLATH_save EQU      0x26      ; thanh ghi quan trọng khi thực thi  
STATUS_save EQU      0x27      ; chương trình ngắt  
FSR_save     EQU      0x28  
;-----  
; Chương trình ngắt  
;-----  
ORG         0x0004  
GOTO        ISR  
ISR
```

```

;-----
; Đoạn chương trình bắt buộc đầu chương trình ngắt
;-----

    MOVWF    W_save
    SWAPF    STATUS,W
    CLRF     STATUS
    MOVWF    STATUS_save
    MOVF     PCLATH,W
    MOVWF    PCLATH_save
    CLRF     PCLATH
    MOVF     FSR,W
    MOVWF    FSR_save

    BTFSS    INTCON,TMR0IF    ; Kiểm tra cờ ngắt Timer0
    GOTO     exit_int         ; Nếu cờ ngắt chưa được set, thoát khỏi
                                ; chương trình ngắt

    BCF      INTCON,TMR0IF    ; nếu cờ ngắt đã được set, xóa cờ ngắt để
                                ; cho phép nhận biết thời điểm tiếp theo
                                ; xảy ra ngắt

;-----
; Các thao tác chính của chương trình ngắt
;-----

    INCF     hang_don_vi,1    ; tăng hàng đơn vị
    MOVLW    0x0A
    XORWF    hang_don_vi,0    ; so sánh hàng đơn vị với 10
    BTFSS    STATUS,Z
    GOTO     exit_int         ; thoát chương trình ngắt nếu chưa bằng 10
    CLRF     hang_don_vi      ; nếu bằng 10, xóa hàng đơn vị
    INCF     hang_chuc,1      ; tăng hàng chục
    MOVLW    0x0A
    XORWF    hang_chuc,0      ; so sánh hàng chục với 10
    BTFSS    STATUS,Z
    GOTO     exit_int         ; thoát chương trình ngắt nếu chưa bằng 10
    CLRF     hang_chuc        ; nếu bằng 10, xóa hàng chục, bắt đầu đếm
                                ; lại từ giá trị 00

    GOTO     exit_int         ; thoát chương trình ngắt

;-----
; Đoạn chương trình bắt buộc trước khi thoát khỏi chương trình ngắt
;-----
exit_int

```

```

MOVF    FSR_save,W
MOVWF   FSR
MOVF    PCLATH_save,W
MOVWF   PCLATH
SWAPF   STATUS_save,W
MOVWF   STATUS
SWAPF   W_save,1
SWAPF   W_save,0
RETFIE

```

```

;-----
; Kết thúc chương trình ngắt
;-----

```

```

ORG      0x0000
GOTO     start
ORG      0x050

```

```

;-----
; Bắt đầu chương trình chính
;-----
start

```

```

;-----
; Khởi tạo các PORT
;-----

```

```

BCF      STATUS,RP1
BSF      STATUS,RP0      ; Chọn BANK1

```

```

MOVLW    0x00
MOVWF    TRISD            ; PORTD <- output
MOVLW    b'11111100'
MOVWF    TRISB            ; PORTB<1:0> <- output

```

```

BCF      STATUS,RP0      ; chọn BANK0

```

```

CLRF     PORTD
MOVLW    b'00000011'     ; tắt các LED hàng chục và hàng đơn vị
MOVWF    PORTB

```

```

;-----
; Khởi tạo Timer0
;-----

```

```

CLRF     TMR0            ; xóa thanh ghi TMR0
CLRF     INTCON          ; xóa thanh ghi INTCON
BSF      STATUS,RP0      ; chọn BANK0

```

```

    MOVLW    b'10000001'    ; tắt chức năng điện trở kéo lên ở PORTB,
    MOVWF    OPTION_REG    ; chọn xung đếm là xung lệnh, gán
                                ; prescaler cho Timer0 và chọn tỉ số chia
                                ; tần số prescaler là 1:4
    BCF      STATUS,RP0    ; chọn BANK0
    BSF      INTCON,TMR0IE ; cho phép ngắt Timer0
    BSF      INTCON,PEIE    ; cho phép ngắt ngoại vi
    BSF      INTCON,GIE    ; cho phép toàn bộ các ngắt
;-----
; Khởi tạo các biến
;-----
    CLRF     hang_chuc
    CLRF     hang_don_vi
;-----
; Vòng lặp chính
;-----
main
    CALL     hien_thi        ; gọi chương trình con
    GOTO     main
;-----
; Chương trình con hiển thị các giá trị chứa trong các thanh ghi hang_chuc và hang_don_vi ra
; các LED hàng chục và LED hàng đơn vị
;-----
hien_thi
    MOVF     hang_chuc,0    ; hiển thị LED hàng chục
    CALL     table
    MOVWF    PORTD
    MOVLW    b'11111101'
    MOVWF    PORTB
    CALL     delay_1ms

    MOVF     hang_don_vi,0  ; hiển thị LED hàng đơn vị
    CALL     table
    MOVWF    PORTD
    MOVLW    b'11111110'
    MOVWF    PORTB
    CALL     delay_1ms
    RETURN        ; kết thúc chương trình con hien_thi
;-----
; Các chương trình con dùng cho chương trình con hien_thi
;-----

```


table ; bảng dữ liệu chuyển từ mã thập phân sang mã
; LED 7 đoạn

```
ADDWF    PCL,1
RETLW    0xC0
RETLW    0xF9
RETLW    0xA4
RETLW    0xB0
RETLW    0x99
RETLW    0x92
RETLW    0x82
RETLW    0xF8
RETLW    0x80
RETLW    0x90
```

delay_1ms

```
MOVLW    d'1'
MOVWF    count1
d2    MOVLW    0xC7
MOVWF    counta
MOVLW    0x01
MOVWF    countb
```

delay_1 ; chương trình con tạo thời gian delay 1 ms

```
DECFSZ    counta,1
GOTO      $+2
DECFSZ    countb,1
GOTO      delay_1
DECFSZ    count1,1
GOTO      d2
RETURN
```

END ; chương trình kết thúc tại đây

Ta nhận thấy rằng cấu trúc chương trình trên hoàn toàn tương tự như cấu trúc của chương trình mẫu, các giải thuật về hiển thị LED đã được đề cập cụ thể ở ứng dụng 4.6, do đó vẫn đề còn lại chỉ là các vấn đề liên quan đến Timer0. Các bước khởi tạo Timer0 đã được đề cập cụ thể trong các tài liệu của nhà sản xuất, ta chỉ việc dựa theo “sườn bài” có sẵn đó và thêm vào các thông số thích hợp đặt vào các thanh ghi điều khiển (đối với Timer0 là các thanh ghi OPTION_REG, thanh ghi INTCON và thanh ghi TMR0) để khởi tạo các điều kiện ban đầu cho Timer0 sao cho phù hợp với mục đích sử dụng.

Với chương trình trên, mỗi lần ngắt Timer0 xảy ra, vi điều khiển sẽ từ vòng lặp của chương trình chính quay trở về chương trình ngắt. Chương trình ngắt sẽ thực hiện công việc tăng giá trị đếm một cách thích hợp ở các thanh ghi hang_chuc và thanh ghi hang_don_vi.

Thuật toán dành cho chương trình ngắt cũng tương đối đơn giản, giá trị đếm sẽ được lưu trực tiếp dưới dạng mã thập phân nên ta không cần phải chuyển đổi từ mã HEX sang mã thập phân. Khi quá trình cập nhật giá trị đếm kết thúc, vi điều khiển quay trở về vòng lặp chính và tiếp tục quá trình hiển thị các giá trị đã được cập nhật từ chương trình ngắt.

Bây giờ ta thử tính thời gian định thời do Timer0 tạo ra. Do ta khởi tạo Timer0 sử dụng xung đếm là xung lệnh nên mỗi xung có thời gian là 1 μ S (đối với oscillator 4 MHz), xung lệnh được chia 4 bởi prescaler nên giá trị của thanh ghi TMR0 sẽ tăng lên 1 đơn vị sau khoảng thời gian $(4 \times 1 \mu\text{S}) = 4 \mu\text{S}$. Như vậy ngắt sẽ xảy ra sau mỗi quãng thời gian $(256 \times 4 \mu\text{S}) = 1024 \mu\text{S}$ (Timer0 là bộ đếm 8 bit và ngắt xảy ra khi TMR0 bị tràn).

Dựa vào chương trình trên ta có thể kiểm tra được tác động của prescaler bằng cách thay đổi giá trị đưa vào thanh ghi OPTION_REG. Sự thay đổi thời gian định thời khi ta thay đổi tỉ số chia của prescaler là tương đối rõ ràng.

Cũng dựa vào chương trình trên ta có thể thay Timer0 bằng Timer1 hoặc Timer2 để làm quen với các Timer của vi điều khiển. Sự thay đổi duy nhất so với chương trình trên là ở các bước khởi tạo, do mỗi Timer đều có một cấu trúc và hoạt động độc lập với nhau. Các bước khởi tạo có thể được tham khảo trong các tài liệu của nhà sản xuất. Chương trình sau có tác dụng như chương trình 4.5.1 nhưng lần này ta sẽ sử dụng Timer2 làm bộ định thời.

```
-----
; Ghi chú về chương trình
;-----
; Chương trình 4.5.2
; Chương trình hiển thị số đếm trên hai LED 7 đoạn theo thứ tự tăng dần
; Timer sử dụng: Timer2
;-----
; Khai báo vi điều khiển
;-----
processor    16f877a
include      <p16f877a.inc>
__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
;-----
; Khai báo biến
;-----
count1      EQU      0x20
counta      EQU      0x21
countb      EQU      0x22
hang_don_vi EQU      0x23
hang_chuc   EQU      0x24
```

W_save	EQU	0x25
PCLATH_save	EQU	0x26
STATUS_save	EQU	0x27
FSR_save	EQU	0x28

```
ORG      0x0004
GOTO     ISR
```

```
;-----
; Chương trình ngắt
;-----
```

```
ISR
```

```
;-----
; Đoạn chương trình bắt buộc đầu chương trình ngắt
;-----
```

```
MOVWF    W_save
SWAPF    STATUS,W
CLRF     STATUS
MOVWF    STATUS_save
MOVF     PCLATH,W
MOVWF    PCLATH_save
CLRF     PCLATH
MOVF     FSR,W
MOVWF    FSR_save
```

```
;-----
; Các thao tác chính của chương trình ngắt
;-----
```

```
BTFSS    PIR1,TMR2IF
GOTO     exit_int

BCF       PIR1,TMR2IF
INCF     hang_don_vi,1
MOVLW    0x0A
XORWF    hang_don_vi,0
BTFSS    STATUS,Z
GOTO     exit_int
CLRF     hang_don_vi
INCF     hang_chuc,1
MOVLW    0x0A
XORWF    hang_chuc,0
BTFSS    STATUS,Z
GOTO     exit_int
```

CLRF hang_chuc
GOTO exit_int

; Đoạn chương trình bắt buộc trước khi thoát khỏi chương trình ngắt

exit_int

MOVF FSR_save,W
MOVWF FSR
MOVF PCLATH_save,W
MOVWF PCLATH
SWAPF STATUS_save,W
MOVWF STATUS
SWAPF W_save,1
SWAPF W_save,0
RETFIE

; Kết thúc chương trình ngắt

ORG 0x0000
GOTO start
ORG 0x050

; Bắt đầu chương trình chính

start

; Khởi tạo các PORT

BCF STATUS,RP1
BSF STATUS,RP0

MOVLW 0x00
MOVWF TRISD
MOVLW b'11111100'
MOVWF TRISB

BCF STATUS,RP0

CLRF PORTD
MOVLW b'00000011'
MOVWF PORTB

```

;-----
; Khởi tạo Timer2
;-----
        MOVLW    b'11111111'    ; postscaler 1:16, prescaler 1:16, Timer2 ON
        MOVWF    T2CON

        BSF      STATUS,RP0
        MOVLW    .249            ; đặt trước giá trị cần số sánh với thanh ghi TMR2
        MOVWF    PR2            ; vào thanh ghi PR2
        BSF      PIE1,TMR2IE    ; cho phép ngắt Timer2

        BCF      STATUS,RP0

        BSF      INTCON,PEIE    ; cho phép các ngắt ngoại vi
        BSF      INTCON,GIE     ; cho phép toàn bộ các ngắt
;-----
; Khởi tạo các biến
;-----
        CLRF     hang_chuc
        CLRF     hang_don_vi
;-----
; Vòng lặp chính
;-----
main
        CALL     hien_thi
        GOTO     main
;-----
; Chương trình con hiển thị các giá trị chứa trong các thanh ghi hang_chuc và hang_don_vi ra
; các LED hàng chục và LED hàng đơn vị
;-----
hien_thi
        MOVF     hang_chuc,0
        CALL     table
        MOVWF    PORTD
        MOVLW    b'11111101'
        MOVWF    PORTB
        CALL     delay_1ms

        MOVF     hang_don_vi,0
        CALL     table
        MOVWF    PORTD

```

```

        MOVLW    b'11111110'
        MOVWF    PORTB
        CALL     delay_1ms
        RETURN

;-----
; Các chương trình con dùng cho chương trình con hien_thi
;-----
table
    ADDWF    PCL,1
    RETLW    0xC0
    RETLW    0xF9
    RETLW    0xA4
    RETLW    0xB0
    RETLW    0x99
    RETLW    0x92
    RETLW    0x82
    RETLW    0xF8
    RETLW    0x80
    RETLW    0x90

delay_1ms
    MOVLW    d'1'
    MOVWF    count1
d2    MOVLW    0xC7
    MOVWF    counta
    MOVLW    0x01
    MOVWF    countb
delay_1
    DECFSZ   counta,1
    GOTO     $+2
    DECFSZ   countb,1
    GOTO     delay_1
    DECFSZ   count1,1
    GOTO     d2
    RETURN
END

```

Timer2 cũng là bộ đếm 8 bit được hỗ trợ thêm thanh ghi so sánh PR2 và hai bộ chia tần số postscaler prescaler giúp ta linh động hơn trong việc tạo ra khoảng thời gian delay thích hợp cho ứng dụng. Thanh ghi điều khiển Timer2 là thanh ghi T2CON. Chương trình trên không có gì mới, nó chỉ giúp ta ôn lại một số đặc điểm của Timer2 và cách khởi tạo nó.

Ứng dụng 4.8: Ứng dụng PIC16F877A và các LED 7 đoạn để làm đồng hồ.

Với hai ví dụ trên ta có thể nắm bắt được các khái niệm cơ bản về tác dụng định thời dùng Timer, và một trong những ứng dụng phổ biến nhất của chế độ định thời là làm đồng hồ điện tử. Ta có thể sử dụng bất cứ Timer nào của vi điều khiển để phục vụ cho ứng dụng này, tuy nhiên để có một cách nhìn tổng quát hơn về các Timer, lần này ta sẽ sử dụng Timer1. Bây giờ ta sẽ tiến hành từng bước để thực hiện thành công ứng dụng này.

Trước tiên là vấn đề về cấu trúc phần cứng, để hiển thị được giờ, phút, giây ta cần đến 6 LED 7 đoạn, cách kết nối hoàn toàn tương tự như các ứng dụng sử dụng 2 LED ở ví dụ 4.7, chỉ việc nối thêm 4 LED 7 đoạn mắc song song với hai LED trước đó và kết nối thêm 4 “công tắc” dùng BJT vào PORTB để điều khiển quét LED.

Tiếp theo là vấn đề về chương trình viết cho vi điều khiển. Cách “phân công” đối với chương trình sẽ không có gì thay đổi, tức là chương trình chính sẽ làm nhiệm vụ hiển thị LED và chương trình ngắt sẽ thực hiện công việc cập nhật các giá trị cần hiển thị. Tuy nhiên có một số vấn đề phát sinh như sau:

Thứ nhất, làm sao tạo ra thời gian định thời 1 giây?? Timer ta sử dụng là Timer1 16 bit với bộ chia tần số prescaler có các tỉ số chia là 1:1, 1:2, 1:4, 1:8 và được điều khiển bởi thanh ghi T1CON (xem lại Timer1 để biết thêm chi tiết). Giá trị đếm tối đa của Timer1 sẽ là 65534, trong khi nếu ta sử dụng oscillator 4 MHz (mỗi xung lệnh có thời gian 1 μ S) thì Timer1 cần phải đếm đến giá trị 1 000 000, và nếu ta có huy động tối đa khả năng chia tần số của prescaler (1:8) thì giá trị đếm cũng phải đạt đến $1\,000\,000/8 = 125\,000$ (vẫn còn lớn hơn rất nhiều so với giá trị đếm tối đa của Timer1. Một giải pháp cho vấn đề này là dùng thêm một thanh ghi đếm phụ (thanh ghi count). Cụ thể như sau: ta cho Timer1 đếm từ 0 đến 25000, do đó ta cần 5 lần đếm như vậy (5 lần ngắt Timer1 xảy ra) để đạt được giá trị đếm 125 000. Như vậy trước khi cập nhật giá trị giây, ta cần kiểm tra xem biến phụ count đã bằng 5 hay chưa, nếu bằng rồi thì mới tăng giá trị giây và reset lại biến count.

Thứ hai, làm sao cập nhật giá trị giờ??? Các giá trị phút và giây tăng từ 0 đến 60 nên thuật toán dùng để cập nhật là tương đối đơn giản (tương tự như thuật toán ở ứng dụng 4.7, chỉ có điều ta không so sánh hàng chục với 10 mà so sánh với 6), còn giá trị giờ chỉ tăng từ 0 đến 24. Giải thuật đề ra là ta không cập nhật từng hàng đơn vị và hàng chục của giá trị giờ như đối với phút và giây, thay vào đó giá trị giờ sẽ được cập nhật vào một thanh ghi, sau đó dùng thuật toán tách hàng chục và hàng đơn vị của giờ như ở ứng dụng 4.6 (chương trình 4.3.2) để hiển thị các giá trị thanh ghi chứa giá trị giờ ra LED 7 đoạn.

Đến đây ta đã có thể viết chương trình cho ứng dụng theo các giải thuật đề ra ở trên. Chương trình cụ thể sẽ được viết như sau:

; Ghi chú về chương trình

; Chương trình 4.5.3

; Chương trình ứng dụng PIC16F877A và LED 7 đoạn để làm đồng hồ điện tử

; Timer sử dụng: Timer1

; Khai báo vi điều khiển

processor 16f877a

include <p16f877a.inc>

_CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _XT_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

; Khai báo biến

count1	EQU	0x20	; Các thanh ghi dùng cho
counta	EQU	0x21	; chương trình con delay_1ms
countb	EQU	0x22	

hang_don_vi_giay	EQU	0x23	; Các thanh ghi chứa các giá trị
hang_chuc_giay	EQU	0x24	; giờ, phút, giây cần hiển thị
hang_don_vi_phut	EQU	0x25	
hang_chuc_phut	EQU	0x26	
gio	EQU	0x27	
hang_don_vi_gio	EQU	0x28	
hang_chuc_gio	EQU	0x29	

count	EQU	0x30	; Các thanh ghi phụ
display_reg	EQU	0x31	
xx	EQU	0x32	
xx1	EQU	0x33	

W_save	EQU	0x34	; Các thanh ghi dùng để lưu lại giá
PCLATH_save	EQU	0x35	; trị các thanh ghi quan trọng khi
STATUS_save	EQU	0x36	; thực thi chương trình ngắt
FSR_save	EQU	0x37	

ORG 0x0004

GOTO ISR


```

;-----
; Chương trình ngắt
;-----
ISR
;-----
; Đoạn chương trình bắt buộc khi bắt đầu chương trình ngắt
;-----
    MOVWF    W_save
    SWAPF    STATUS,W
    CLRF     STATUS
    MOVWF    STATUS_save
    MOVF     PCLATH,W
    MOVWF    PCLATH_save
    CLRF     PCLATH
    MOVF     FSR,W
    MOVWF    FSR_save
;-----
; Kiểm tra các cờ ngắt
;-----
    BTFSS    PIR1,TMR1IF    ; kiểm tra cờ ngắt của Timer1
    GOTO     exit_int
    BCF      T1CON,TMR1ON    ; tạm thời tắt Timer1 để khởi tạo lại
;-----
; Các thao tác chính của chương trình ngắt
;-----
    CLRF     TMR1L           ; Khởi tạo lại các giá trị chứa trong thanh
    CLRF     TMR1H           ; ghi TMRH và TMRL
    MOVLW    0x61            ; Đưa vào các thanh ghi đếm của Timer1
    MOVWF    TMR1H           ; giá trị 25000 (25000 -> 61A8h)
    MOVLW    0xA8
    MOVWF    TMR1L

    BSF      T1CON,TMR1ON    ; Bật Timer1
    BCF      PIR1,TMR1IF    ; xóa cờ ngắt để tiếp tục nhận biết thời điểm tiếp
                                ; theo ngắt xảy ra
    INCF     count           ; biến đếm phụ
    MOVLW    d'5'            ; so sánh count với giá trị 5
    XORWF    count,0
    BTFSS    STATUS,Z
    GOTO     exit_int        ; nếu chưa bằng 5, thoát khỏi ngắt
    CLRF     count           ; nếu đã bằng 5, reset lại biến count

```

```

INCF      hang_don_vi_giay,1 ; tăng hàng đơn vị của biến giây
MOVLW    0x0A                ; so sánh với 10
XORWF    hang_don_vi_giay,0 ; cập nhật hàng chục của giá trị giây
BTSS     STATUS,Z
GOTO     exit_int
CLRF     hang_don_vi_giay
INCF     hang_chuc_giay,1
MOVLW    0x06                ; so sánh giá trị hàng chục giây với 6
XORWF    hang_chuc_giay,0
BTSS     STATUS,Z
GOTO     exit_int

```

```

CLRF     hang_chuc_giay      ; cập nhật giá trị phút
INCF     hang_don_vi_phut,1
MOVLW    0x0A                ; so sánh hàng đơn vị của giá trị phút với 10
XORWF    hang_don_vi_phut,0
BTSS     STATUS,Z
GOTO     exit_int
CLRF     hang_don_vi_phut
INCF     hang_chuc_phut,1
MOVLW    0x06                ; so sánh hàng chục của giá trị phút với 6
XORWF    hang_chuc_phut,0
BTSS     STATUS,Z
GOTO     exit_int

```

```

CLRF     hang_chuc_phut
INCF     gio,1                ; cập nhật giá trị giờ
MOVLW    0x18
XORWF    gio,0
BTSS     STATUS,Z
GOTO     exit_int
CLRF     gio
GOTO     exit_int

```

```

;-----
; Đoạn chương trình bắt buộc dùng để kết thúc chương trình ngắt
;-----

```

exit_int

```

MOVF     FSR_save,W
MOVWF    FSR
MOVF     PCLATH_save,W
MOVWF    PCLATH

```

```

SWAPF    STATUS_save,W
MOVWF    STATUS
SWAPF    W_save,1
SWAPF    W_save,0
RETFIE

```

```

ORG      0x0000
GOTO     start
ORG      0x050

```

```

;-----
; Chương trình chính
;-----

```

```

start

```

```

;-----
; Khởi tạo các PORT điều khiển
;-----

```

```

BCF      STATUS,RP1
BSF      STATUS,RP0
MOVLW    0x00          ; PORTD <-output
MOVWF    TRISD
MOVLW    b'11000000'   ; PORTB <5:0> <- output
MOVWF    TRISB         ; Ta cần 6 pin ở PORTB để điều khiển quét LED
BCF      STATUS,RP0
CLRF     PORTD
MOVLW    b'00111111'   ; Tắt tất cả các LED
MOVWF    PORTB

```

```

;-----
; Khởi tạo Timer1
;-----

```

```

CLRF     T1CON
CLRF     INTCON
CLRF     TMR1H
CLRF     TMR1L
BSF      STATUS,RP0    ; Chọn BANK1
CLRF     PIE1
BSF      PIE1,TMR1IE    ; Cho phép ngắt Timer1
BCF      STATUS,RP0    ; Chọn BANK0
CLRF     PIR1          ; xóa tất cả các cờ ngắt
MOVLW    0X30          ; prescaler 1:8, xung đếm là xung lệnh, tạm thời
MOVWF    T1CON         ; tắt Timer1

```

```

MOVLW    0x61                ; Khởi tạo các giá trị trong thanh ghi TMR1H
MOVWF    TMR1H                ; và TMR1L (TMR1H:TMR1L = 25000)
MOVLW    0xA8
MOVWF    TMR1L

```

```

BSF      T1CON,TMR1ON        ; Bật Timer1
BSF      INTCON,TMR1IE        ; Cho phép ngắt Timer1
BSF      INTCON,PEIE          ; Cho phép ngắt ngoại vi
BSF      INTCON,GIE           ; Cho phép toàn bộ các ngắt

```

```

;-----
; Khởi tạo các biến
;-----

```

```

CLRF     gio
CLRF     hang_chuc_gio
CLRF     hang_don_vi_gio
CLRF     hang_don_vi_phut
CLRF     hang_chuc_phut
CLRF     hang_chuc_giay
CLRF     hang_don_vi_giay
CLRF     count

```

```

;-----
; Vòng lặp chính
;-----

```

main

```

CALL     hien_thi
GOTO     main

```

hien_thi

```

CALL     chuyen_ma_gio        ; gọi chương trình con chuyen_ma_gio
MOVF     hang_chuc_gio,0       ; Hiển thị giá trị giờ ra LED
CALL     table
MOVWF    PORTD
MOVLW    b'11011111'
MOVWF    PORTB
CALL     delay_1ms
MOVF     hang_don_vi_gio,0
CALL     table
MOVWF    PORTD
MOVLW    b'11101111'
MOVWF    PORTB
CALL     delay_1ms

```

```

MOVF    hang_chuc_phut,0    ; Hiển thị giá trị phút ra LED
CALL    table
MOVWF   PORTD
MOVLW   b'11110111'
MOVWF   PORTB
CALL    delay_1ms
MOVF    hang_don_vi_phut,0
CALL    table
MOVWF   PORTD
MOVLW   b'1111011'
MOVWF   PORTB
CALL    delay_1ms

```

```

MOVF    hang_chuc_giay,0    ; Hiển thị giá trị giây ra LED
CALL    table
MOVWF   PORTD
MOVLW   b'11111101'
MOVWF   PORTB
CALL    delay_1ms
MOVF    hang_don_vi_giay,0
CALL    table
MOVWF   PORTD
MOVLW   b'11111110'
MOVWF   PORTB
CALL    delay_1ms
RETURN

```

```

table                                         ; Bảng dữ liệu dùng để chuyển đổi
ADDWF   PCL,1                               ; từ mã thập phân sang mã LED 7 đoạn
RETLW   0xC0
RETLW   0xF9
RETLW   0xA4
RETLW   0xB0
RETLW   0x99
RETLW   0x92
RETLW   0x82
RETLW   0xF8
RETLW   0x80
RETLW   0x90

```

```

delay_1ms                                   ; Chương trình con tạo thời gian delay 1ms
MOVLW   d'1'

```

```

    MOVWF    count1
d2    MOVLW    0xC7
    MOVWF    counta
    MOVLW    0x01
    MOVWF    countb

```

```

delay_1
    DECFSZ    counta,1
    GOTO      $+2
    DECFSZ    countb,1
    GOTO      delay_1
    DECFSZ    count1,1
    GOTO      d2
    RETURN

```

```

chuyen_ma_gio    ; chương trình con dùng để tách rời giá trị hàng
    MOVF          gio,0    ; chục và hàng đơn vị của thanh ghi chứa giá trị
    MOVWF         display_reg    ; giờ và chuyển sang mã thập phân
    ANDLW         0x0F    ; Kết quả chuyển đổi được lưu trong thanh ghi
    MOVWF         hang_don_vi_gio    ; hang_don_vi_gio và hang_phut_gio
    MOVLW         0xF0
    ANDWF         display_reg,0
    MOVWF         hang_chuc_gio
    SWAPF         hang_chuc_gio,1
    MOVF          hang_don_vi_gio,0
    CALL          chuyen_ma
    MOVWF         hang_don_vi_gio
    BTFSC         xx1,0
    INCF          hang_chuc_gio,1
    MOVF          hang_chuc_gio,0
    CALL          chuyen_ma
    MOVWF         hang_chuc_gio
    RETURN

```

```

chuyen_ma    ; chương trình con chuyển từ mã HEX sang
              ; mã thập phân
    MOVWF         xx
    MOVLW         0x00
    XORWF         xx,0
    BTFSC         STATUS,Z
    GOTO          nho_hon_10

    MOVLW         0x01

```

XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x02
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x03
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x04
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x05
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x06
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x07
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x08
XORWF xx,0
BTFSK STATUS,Z
GOTO nho_hon_10

MOVLW 0x09
XORWF xx,0

BTFSC STATUS,Z
GOTO nho_hon_10

MOVLW 0x0A
XORWF xx,0
BTFSC STATUS,Z
GOTO bang_10

MOVLW 0x0B
XORWF xx,0
BTFSC STATUS,Z
GOTO bang_11

MOVLW 0x0C
XORWF xx,0
BTFSC STATUS,Z
GOTO bang_12

MOVLW 0x0D
XORWF xx,0
BTFSC STATUS,Z
GOTO bang_13

MOVLW 0x0E
XORWF xx,0
BTFSC STATUS,Z
GOTO bang_14

MOVLW 0x0F
XORWF xx,0
BTFSC STATUS,Z
GOTO bang_15

nho_hon_10

MOVLW 0x00
MOVWF xx1
MOVF xx,0
RETURN

bang_10

MOVLW 0x01
MOVWF xx1

RETLW	0x00
bang_11	
MOVLW	0x01
MOVWF	xx1
RETLW	0x01
bang_12	
MOVLW	0x01
MOVWF	xx1
RETLW	0x02
bang_13	
MOVLW	0x01
MOVWF	xx1
RETLW	0x03
bang_14	
MOVLW	0x01
MOVWF	xx1
RETLW	0x04
bang_15	
MOVLW	0x01
MOVWF	xx1
RETLW	0x05
END	; Kết thúc chương trình

Thực ra ta có nhiều phương pháp khác để tạo thời gian định thời 1s bằng cách sử dụng các đặc tính của Timer1, chẳng hạn ta có thể sử dụng oscillator ngoại vi khác cho Timer1 mà không cần dùng chung với oscillator của vi điều khiển. Ta cũng có thể sử dụng các Timer khác cho ứng dụng này và tùy theo đặc điểm cấu tạo của từng Timer ta có thể xác định được các giá trị thích hợp để tạo thời gian định thời 1s.

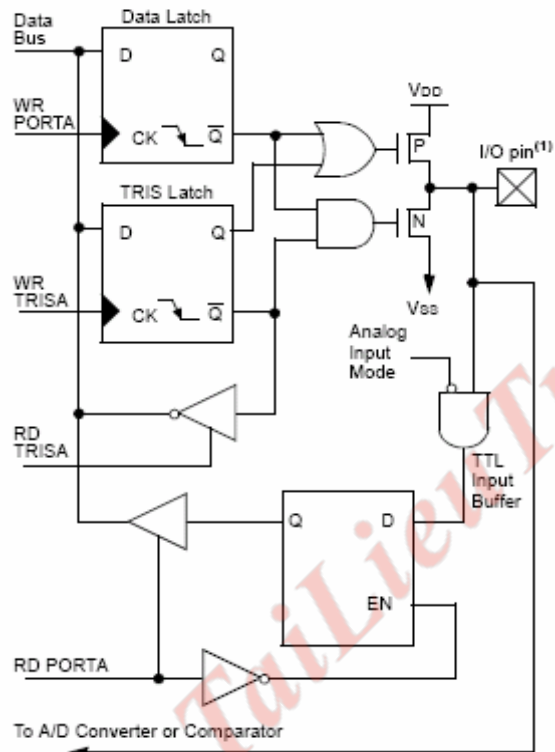
Tuy nhiên dù sử dụng phương pháp nào đi nữa ta cũng không thể tạo ra đồng hồ điện tử có độ chính xác tuyệt đối khi sử dụng vi điều khiển do thời gian thực thi lệnh của vi điều khiển sau mỗi thời gian định thời không thể được xác định một cách chính xác. Tuy nhiên đây cũng là ứng dụng hoàn chỉnh nhất và mang tính thực tiễn nhiều nhất so với các ứng dụng trước.

TaiLieuTuoi.com

PHỤ LỤC 1 SƠ ĐỒ KHỐI CÁC PORT CỦA VI ĐIỀU KHIỂN PIC16F877A

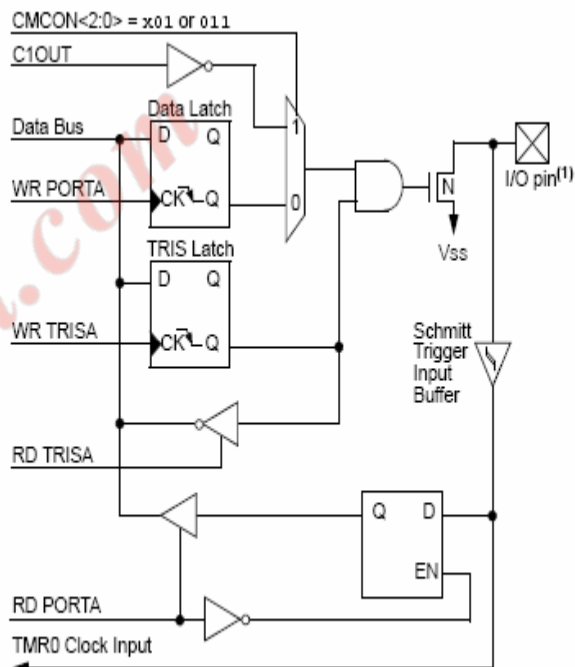
P1.1 PORTA

Sơ đồ khối RA3:RA0



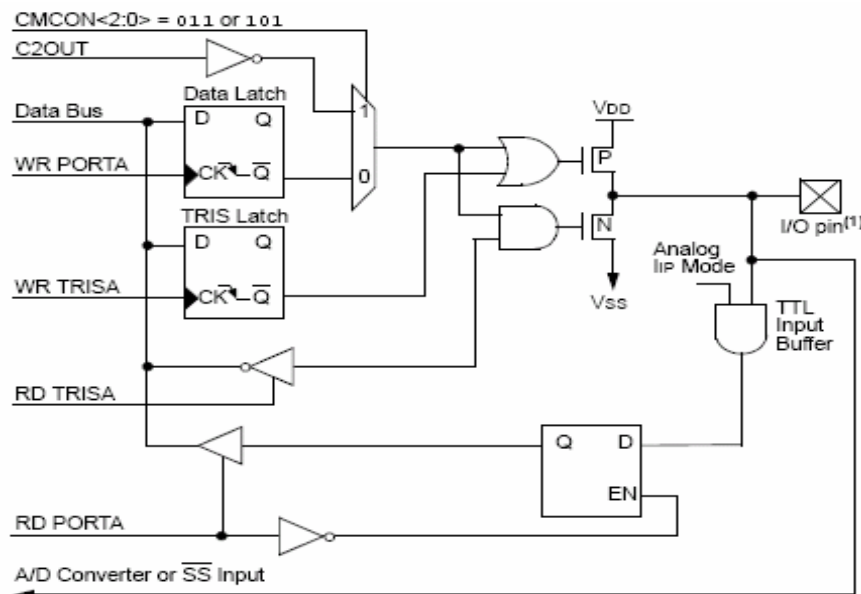
Note 1: I/O pins have protection diodes to VDD and VSS.

Sơ đồ khối RA4.



Note 1: I/O pin has protection diodes to Vss only.

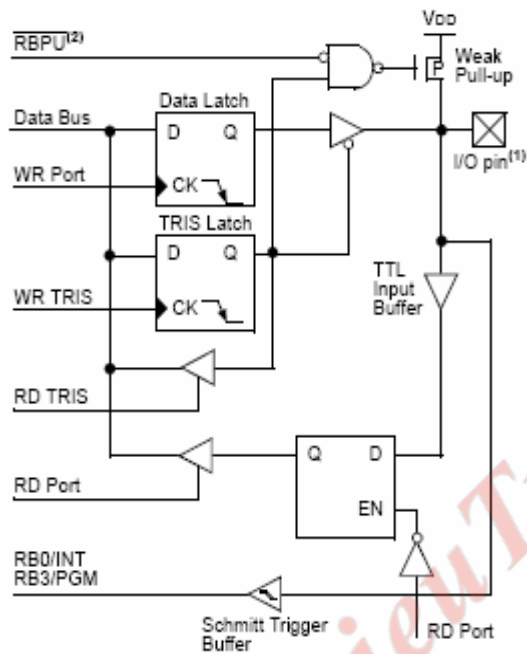
Sơ đồ khối RA5



Note 1: I/O pin has protection diodes to VDD and VSS.

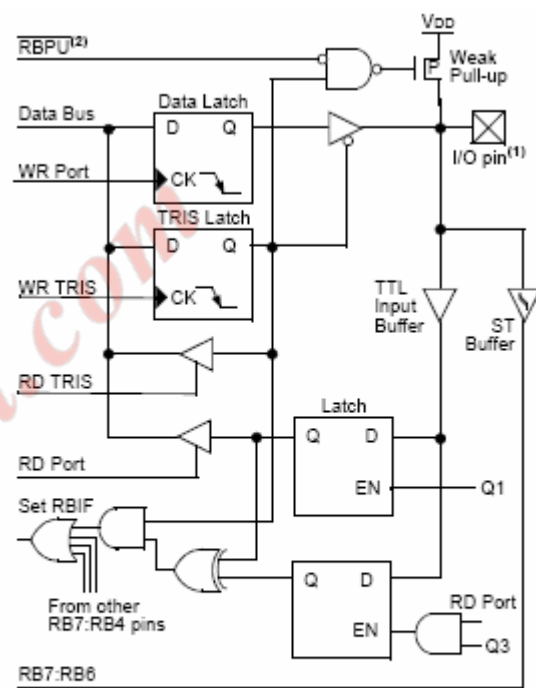
P1.2 PORTB

Sơ đồ khối RB3:RB0



- Note 1:** I/O pins have diode protection to VDD and VSS.
Note 2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

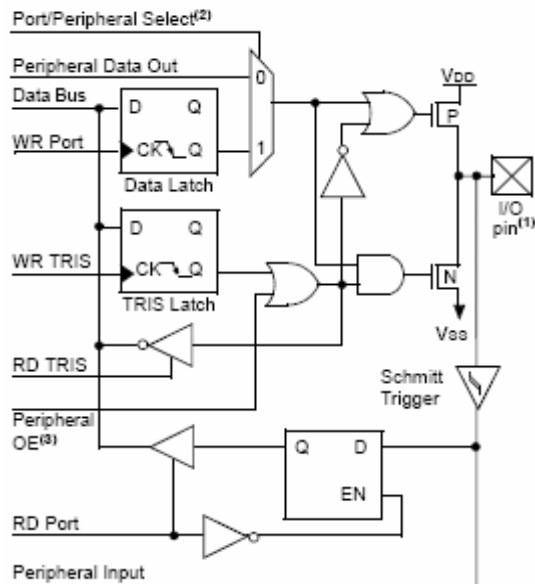
Sơ đồ khối RB7:RB4



- Note 1:** I/O pins have diode protection to VDD and VSS.
Note 2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

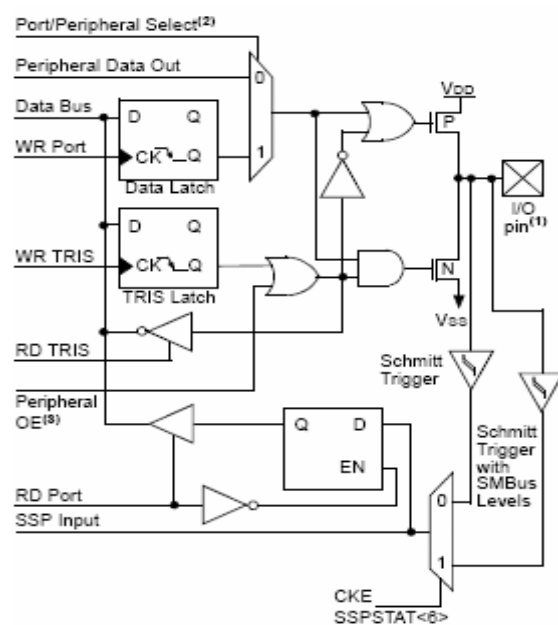
P1.3 PORTC

Sơ đồ khối RC7:RC5 và RC2:RC0



- Note 1:** I/O pins have diode protection to VDD and VSS.
Note 2: Port/Peripheral Select signal selects between port data and peripheral output.
Note 3: Peripheral OE (Output Enable) is only activated if Peripheral Select is active.

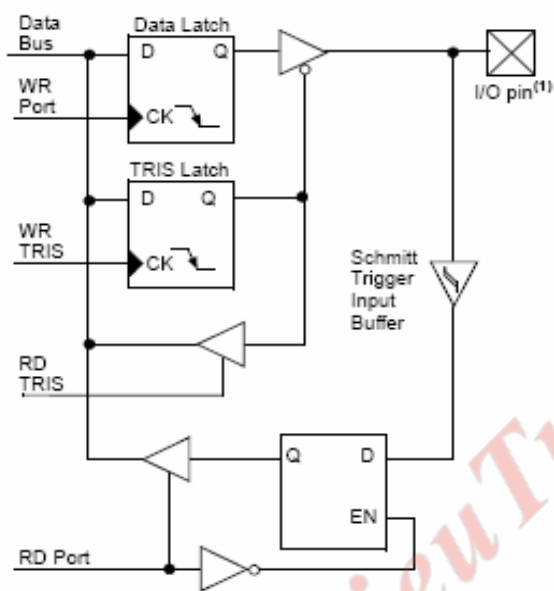
Sơ đồ khối RC4:RC3



- Note 1:** I/O pins have diode protection to VDD and VSS.
Note 2: Port/Peripheral Select signal selects between port data and peripheral output.
Note 3: Peripheral OE (Output Enable) is only activated if Peripheral Select is active.

P1.4 PORTD

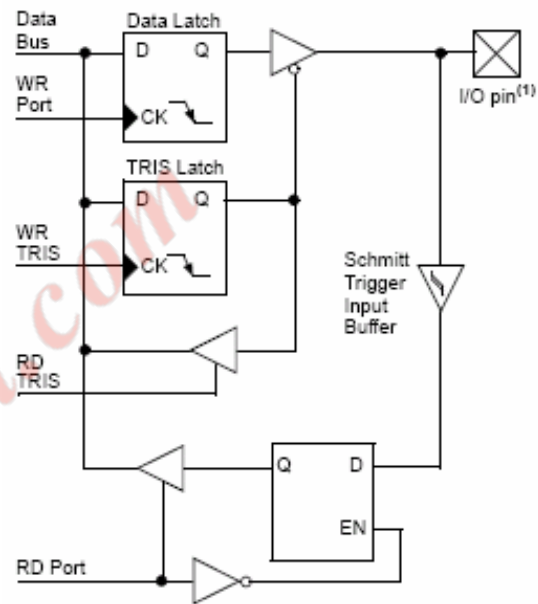
Sơ đồ khối RD7:RD0



Note 1: I/O pins have protection diodes to V_{DD} and V_{SS}.

P1.5 PORTE

Sơ đồ khối RE2:RE0



Note 1: I/O pins have protection diodes to V_{DD} and V_{SS}.

PHỤ LỤC 2 THANH GHI SFR (SPECIAL FUNCTION REGISTER)

P2.1 Thanh ghi TMR0: địa chỉ 01h, 101h.

Thanh ghi 8 bit chứa giá trị của bộ định thời Timer0.

P2.2 Thanh ghi PCL: địa chỉ 02h, 82h, 102h, 182h.

Thanh ghi chứa 8 bit thấp của bộ đếm chương trình (PC).

P2.3 Thanh ghi STATUS: địa chỉ 03h, 83h, 103h, 183h

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C
bit 7							bit 0

Bit 7: IRP bit chọn bank bộ nhớ dữ liệu cần truy xuất (dùng cho địa chỉ gián tiếp).

IRP = 0: bank 2,3 (từ 100h đến 1FFh)

IRP = 1: bank 0,1 (từ 00h đến FFh)

Bit 6,5:RP1:RP0 hai bit chọn bank bộ nhớ dữ liệu cần truy xuất (dùng cho địa chỉ trực tiếp)

RP1:RP0	Bank
00	0
01	1
10	2
11	3

Bit 4: \overline{TO} : bit chỉ thị trạng thái của WDT(Watch Dog Timer)

\overline{TO} =1 khi vi điều khiển vừa được cấp nguồn, hoặc sau khi lệnh CLRWDT hay SLEEP được thực thi.

\overline{TO} =0 khi WDT bị tràn

Bit 3: \overline{PD} bit chỉ thị trạng thái nguồn

\overline{PD} = 1 khi vi điều khiển được cấp nguồn hoặc sau lệnh CLRWDT

\overline{PD} = 0 sau khi lệnh SLEEP được thực thi

Bit 2: Z bit Zero

Z =1 khi kết quả của phép toán hay logic bằng 0

Z = 0 khi kết quả của phép toán hay logic khác 0

Bit 1: DC Digit carry/Borrow

DC = 1 khi kết quả phép toán tác động lên 4 bit thấp có nhớ.

DC = 0 khi kết quả phép toán tác động lên 4 bit thấp không có nhớ.

Bit 0 C Carry/borrow

C =1 khi kết quả phép toán tác động lên bit MSB có nhớ.

C=0 khi kết quả phép toán tác động lên bit MSB không có nhớ.

P2.4 Thanh ghi SFR: địa chỉ 04h.

Thanh ghi chứa con trỏ địa chỉ gián tiếp của bộ nhớ dữ liệu.

P2.5 Thanh ghi PORTA: địa chỉ 05h.

Thanh ghi chứa giá trị nhận vào hay xuất ra PORTA.

P2.6 Thanh ghi PORTB: địa chỉ 06h, 106h.

Thanh ghi chứa giá trị nhận vào hay xuất ra PORTB.

P2.7 Thanh ghi PORTC: địa chỉ 07h.

Thanh ghi chứa giá trị nhận vào hay xuất ra PORTC

P2.8 Thanh ghi PORTD: địa chỉ 08h.

Thanh ghi chứa giá trị nhận vào hay xuất ra PORTD.

P2.9 Thanh ghi PORTE: địa chỉ 09h.

Thanh ghi chứa giá trị nhận vào hay xuất ra PORTE.

P2.10 Thanh ghi PCLATCH: địa chỉ 0Ah, 8Ah, 10Ah, 18Ah.

Thanh ghi đóng vai trò là buffer đệm trong quá trình ghi giá trị lên 5 bit cao của bộ đếm chương trình PC.

P2.11 Thanh ghi INTCON: địa chỉ 0Bh, 8Bh, 10Bh, 18Bh.

Thanh ghi chứa các bit điều khiển và các bit cờ hiệu khi timer0 bị tràn, ngắt ngoại vi RB0/INT và ngắt interrupt-on-change tại các chân của PORTB.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

- Bit 7 GIE Global Interrupt Enable bit
GIE = 1 cho phép tất cả các ngắt.
GIE = 0 không cho phép tất cả các ngắt.
- Bit 6 PEIE Pheripheral Interrupt Enable bit
PEIE = 1 cho phép tất cả các ngắt ngoại vi
PEIE = 0 không cho phép tất cả các ngắt ngoại vi
- Bit 5 TMR0IE Timer0 Overflow Interrupt Enable bit
TMR0IE = 1 cho phép ngắt Timer0
TMR0IE = 0 không cho phép ngắt Timer0
- Bit 4 RBIE RB0/INT External Interrupt Enable bit
RBIE = 1 cho phép ngắt ngoại vi RB0/INT
RBIE = 0 không cho phép ngắt ngoại vi RB0/INT
- Bit 3 RBIE RB Port change Interrupt Enable bit
RBIE = 1 cho phép ngắt RB Port change
RBIE = 0 không cho phép ngắt RB Port change

- Bit 2 TMR0IF Timer0 Interrupt Flag bit
TMR0IF = 1 thanh ghi TMR0 bị tràn (phải xóa bằng chương trình).
TMR0IF = 0 thanh ghi TMR0 chưa bị tràn.
- Bit 1 INTF BR0/INT External Interrupt Flag bit
INTF = 1 ngắt RB0/INT xảy ra (phải xóa cờ hiệu bằng chương trình).
INTF = 0 ngắt RB0/INT chưa xảy ra.
- Bit 0 RBIF RB Port Change Interrupt Flag bit
RBIF = 1 ít nhất có một chân RB7:RB4 có sự thay đổi trạng thái. Bit này phải được xóa bằng chương trình sau khi đã kiểm tra lại các giá trị của các chân tại PORTB.
RBIF = 0 không có sự thay đổi trạng thái các chân RB7:RB4.

P2.12 Thanh ghi PIR1: địa chỉ 0Ch

Thanh ghi chứa cờ ngắt của các khối ngoại vi.

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

- Bit 7 PSPIF Parallel Slave Port Read/Write Interrupt Flag bit
PSPIF = 1 vừa hoàn tất thao tác đọc hoặc ghi PSP (phải xóa bằng chương trình).
PSPIF = 0 không có thao tác đọc ghi PSP nào diễn ra.
- Bit 6 ADIF ADC Interrupt Flag bit
ADIF = 1 hoàn tất chuyển đổi ADC.
ADIF = 0 chưa hoàn tất chuyển đổi ADC.
- Bit 5 RCIF USART Receive Interrupt Flag bit
RCIF = 1 buffer nhận qua chuẩn giao tiếp USART đã đầy.
RCIF = 0 buffer nhận qua chuẩn giao tiếp USART rỗng.
- Bit 4 TXIF USART Transmit Interrupt Flag bit
TXIF = 1 buffer truyền qua chuẩn giao tiếp USART rỗng.
TXIF = 0 buffer truyền qua chuẩn giao tiếp USART đầy.
- Bit 3 SSPIF Synchronous Serial Port (SSP) Interrupt Flag bit
SSPIF = 1 ngắt truyền nhận SSP xảy ra.
SSPIF = 0 ngắt truyền nhận SSP chưa xảy ra.
- Bit 2 CCP1IF CCP1 Interrupt Flag bit
Khi CCP1 ở chế độ Capture
CCP1IF=1 đã cập nhật giá trị trong thanh ghi TMR1.
CCP1IF=0 chưa cập nhật giá trị trong thanh ghi TMR1.
Khi CCP1 ở chế độ Compare

CCP1IF=1 giá trị cần so sánh bằng với giá trị chứa trong TMR1
 CCP1IF=0 giá trị cần so sánh không bằng với giá trị trong TMR1.

- Bit 1 TMR2IF TMR2 to PR2 Match Interrupt Flag bit
 TRM2IF = 1 giá trị chứa trong thanh ghi TMR2 bằng với giá trị chứa trong thanh ghi PR2.
 TRM2IF = 0 giá trị chứa trong thanh ghi TMR2 chưa bằng với giá trị chứa trong thanh ghi PR2.
- Bit 0 TMR1IF TMR1 Overflow Interrupt Flag bit
 TMR1IF = 1 thanh ghi TMR1 bị tràn (phải xóa bằng chương trình).
 TMR1IF = 0 thanh ghi TMR1 chưa bị tràn.

P2.13 Thanh ghi PIR2: địa chỉ 0Dh

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

Bit 7, 5, 2, 1: không quan tâm và mặc định mang giá trị 0.

- Bit 6 CMIF Comparator Interrupt Flag bit
 CMIF = 1 tín hiệu ngõ vào bộ so sánh thay đổi.
 CMIF = 0 tín hiệu ngõ vào bộ so sánh không thay đổi.
- Bit 4 EEIF EEPROM Write Operation Interrupt Flag bit
 EEIF = 1 quá trình ghi dữ liệu lên EEPROM hoàn tất.
 EEIF = 0 quá trình ghi dữ liệu lên EEPROM chưa hoàn tất hoặc chưa bắt đầu.
- Bit 3 BCLIF Bus Collision Interrupt Flag bit
 BCLIF = 1 Bus truyền nhận đang bận khi (đang có dữ liệu truyền đi trong bus) khi SSP hạt động ở chế độ I2C Master mode.
 BCLIF = 0 Bus truyền nhận chưa bị tràn (không có dữ liệu truyền đi trong bus).
- Bit 0 CCP2IF CCP2 Interrupt Flag bit
 Ở chế độ Capture
 CCP2IF = 1 đã cập nhật giá trị trong thanh ghi TMR1.
 CCP2IF = 0 chưa cập nhật giá trị trong thanh ghi TMR1.
 Ở chế độ Compare
 CCP2IF = 1 giá trị cần so sánh bằng với giá trị chứa trong TMR1.
 CCP2IF = 0 giá trị cần so sánh chưa bằng với giá trị chứa trong TMR1.

P2.14 Thanh ghi TMR1L: địa chỉ 0Eh

Thanh ghi chứa 8 bit thấp của bộ định thời TMR1.

P2.15 Thanh ghi TMR1H: địa chỉ 0Fh

Thanh ghi chứa 8 bit cao của bộ định thời TMR2.

P2.16 Thanh ghi T1CON: địa chỉ 10h

Thanh ghi điều khiển Timer1.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON
bit 7							bit 0

- Bit 7,6 Không quan tâm và mang giá trị mặc định bằng 0.
- Bit 5,4 T1CKPS1:T1CKPS0 Timer1 Input Clock Prescaler Select bit
- 11 tỉ số chia tần số của prescaler là 1:8
 10 tỉ số chia tần số của prescaler là 1:4
 01 tỉ số chia tần số của prescaler là 1:2
 00 tỉ số chia tần số của prescaler là 1:1
- Bit 3 T1OSCEN Timer1 Oscillator Enable Control bit
 T1OSCEN = 1 cho phép Timer1 hoạt động với xung do oscillator cung cấp.
 T1OSCEN = 0 không cho phép Timer1 hoạt động với xung do oscillator cung cấp (tắt bộ chuyển đổi xung bên trong Timer1).
- Bit 2 $\overline{T1SYNC}$ Timer1 ternal Clock Input Synchronization Control bit
 Khi TMR1CS = 1:
 $\overline{T1SYNC}$ = 1 không đồng bộ xung clock ngoại vi đưa vào Timer1.
 $\overline{T1SYNC}$ = 0 đồng bộ xung clock ngoại vi đưa vào Timer1.
 Khi TMR1CS = 0
 Bit $\overline{T1SYNC}$ không được quan tâm do Timer1 sử dụng xung clock bên trong.
- Bit 1 TMR1CS Timer1 Clock Source Select bit
 TMR1CS = 1 chọn xung đếm là xung ngoại vi lấy từ pin RC0/T1OSC/T1CKI (cạnh tác động là cạnh lên).
 TMR1CS = 0 chọn xung đếm là xung clock bên trong ($F_{osc}/4$).
- Bit 0 TMR1ON Timer1 On bit
 TMR1ON = 1 cho phép Timer1 hoạt động.
 TMR1ON = 0 Timer1 ngưng hoạt động.

P2.17 Thanh ghi TMR2: địa chỉ 11h

Thanh ghi chứa giá trị bộ đếm Timer2.

P2.18 Thanh ghi T2CON: địa chỉ 12h

Thanh ghi điều khiển Timer2.

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

- Bit 7 Không quan tâm và mặc định mang giá trị 0
- Bit 6-3 TOUTPS3:TOUTPS0 Timer2 Output Postscaler Select bit
 Các bit này điều khiển việc lựa chọn tỉ số chia tần số cho postscaler.

0101 Slave mode, xung clock lấy từ pin SCK, không cho phép pin điều khiển \overline{SS} (\overline{SS} là pin I/O bình thường).

0100 SPI Slave mode, xung clock lấy từ pin SCK, cho phép pin điều khiển \overline{SS} .

0011 SPI Master mode, xung clock bằng (ngõ ra TMR2)/2.

0010 SPI Master mode, xung clock bằng ($F_{OSC}/64$).

0001 SPI Master mode, xung clock bằng ($F_{OSC}/16$).

0000 SPI Master mode, xung clock bằng ($F_{OSC}/4$).

Các trạng thái không được liệt kê hoặc không có tác dụng điều khiển hoặc chỉ có tác dụng đối với chế độ I2C mode.

Khi MSSP ở chế độ I2C

Bit 7 WCOL Write Collision Detect bit

Khi truyền dữ liệu ở chế độ I2C Master mode:

WCOL = 1 đưa dữ liệu truyền đi vào thanh ghi SSPBUF trong khi chế độ truyền dữ liệu của I2C chưa sẵn sàng.

WCOL = 0 không xảy ra hiện tượng trên.

khi truyền dữ liệu ở chế độ I2C Slave mode:

WCOL = 1 dữ liệu mới được đưa vào thanh ghi SSPBUF trong khi dữ liệu cũ chưa được truyền đi.

WCOL = 0 không có hiện tượng trên xảy ra.

Ở chế độ nhận dữ liệu (Master hoặc Slave):

Bit này không có tác dụng chỉ thị các trạng thái.

Bit 6 SSPOV Receive Overflow Indicator Flag bit.

Khi nhận dữ liệu:

SSPOV = 1 dữ liệu mới được nhận vào thanh ghi SSPBUF trong khi dữ liệu cũ chưa được đọc.

SSPOV = 0 không có hiện tượng trên xảy ra.

Khi truyền dữ liệu:

Bit này không có tác dụng chỉ thị các trạng thái.

Bit 5 SSPEN Synchronous Serial Port Enable bit

SSPEN = 1 cho phép cổng giao tiếp MSSP (các pin SDA và SCL).

SSPEN = 0 không cho phép cổng giao tiếp MSSP.

Cần chú ý là các pin SDA và SCL phải được điều khiển trạng thái bằng các bit tương ứng trong thanh ghi TRISC trước đó).

Bit 4 CKP SCK Release Control bit

Ở chế độ Slave mode:

CKP = 1 cho xung clock tác động.

CKP = 0 giữ xung clock ở mức logic thấp (để bảo đảm thời gian thiết lập dữ liệu).

Bit 3,0 SSPM3:SSPM0

Các bit này đóng vai trò lựa chọn các chế độ hoạt động của MSSP.

1111 I2C Slave mode 10 bit địa chỉ và cho phép ngắt khi phát hiện bit Start và bit Stop.

1110 I2C Slave mode 7 bit địa chỉ và cho phép ngắt khi phát hiện bit Start và bit Stop.

1011 I2C Firmwave Controlled Master mode (không cho phép chế độ Slave).

1000 I2C Master mode, xung clock = $F_{OSC}/(4*(SSPADD+1))$.

0111 I2C Slave mode 10 bit địa chỉ.

Các trạng thái không được liệt kê hoặc không có tác dụng điều khiển hoặc chỉ có tác dụng đối với chế độ SPI mode.

P2.21 Thanh ghi CCPR1L: địa chỉ 15h

Thanh ghi chứa 8 bit thấp của khối CCP1.

P2.22 Thanh ghi CCPR1H: địa chỉ 16h

Thanh ghi chứa 8 bit cao của khối CCP1.

P2.23 Thanh ghi CCP1CON và thanh ghi CCP2CON: địa chỉ 17h (CCP1CON) và 1Dh (SSP2CON)

Thanh ghi điều khiển khối CCP1.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

Bit 7,6 Không có tác dụng và mặc định mang giá trị 0.

Bit 5,4 CCPxX:CCPxY: PWM least Significant bits (các bit này không có tác dụng ở chế độ Capture và Compare)

Ở chế độ PWM, đây là 2 bit MSB chứa giá trị tính độ rộng xung (duty cycle) của khối PWM (8 bit còn lại được chứa trong thanh ghi CCPRxL).

Bit 3-0 CCPxM3:CCPxM0 CCPx Mode Select bit

Các bit dùng để xác lập các chế độ hoạt động của khối CCPx

0000 không cho phép CCPx (hoặc dùng để reset CCPx)

0100 CCPx hoạt động ở chế độ Capture, “hiện tượng” được thiết lập là mỗi cạnh xuống tại pin dùng cho khối CCPx.

0101 CCPx hoạt động ở chế độ Capture, “hiện tượng” được thiết lập là mỗi cạnh lên tại pin dùng cho khối CCPx.

0110 CCPx hoạt động ở chế độ Capture, “hiện tượng” được thiết lập là mỗi cạnh lên thứ 4 tại pin dùng cho khối CCPx.

0111 CCPx hoạt động ở chế độ Capture, “hiện tượng” được thiết lập là mỗi cạnh lên thứ 16 tại pin dùng cho khối CCPx.

1000 CCPx hoạt động ở chế độ Compare, ngõ ra được đưa lên mức cao và bit CCPxIF được set khi các giá trị cần so sánh bằng nhau.

1001 CCPx hoạt động ở chế độ Compare, ngõ ra được xuống mức thấp và bit CCPxIF được set khi các giá trị cần so sánh bằng nhau.

1010 CCPx hoạt động ở chế độ Compare, khi các giá trị cần so sánh bằng nhau, ngắt xảy ra, bit CCPxIF được set và trạng thái pin output không bị ảnh hưởng.

1011 CCPx hoạt động ở chế độ Compare, khi các giá trị cần so sánh bằng nhau, xung trigger đặc biệt (Trigger Special Event) sẽ được tạo ra, khi đó cờ ngắt CCPxIF được set, các pin output không thay đổi trạng thái, CCP1 reset Timer1, CCP2 reset Timer1 và khởi động khối ADC.

11xx CCPx hoạt động ở chế độ PWM.

P2.24 Thanh ghi RCSTA: địa chỉ 18h

Thanh ghi chứa các bit trạng thái và các bit điều khiển quá trình nhận dữ liệu qua chuẩn giao tiếp USART.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Bit 7 SPEN Serial Port Enable bit

SPEN = 1 Cho phép cổng giao tiếp USART (pin RC7/RX/DT và RC6/TX/CK).

SPEN = 0 không cho phép cổng giao tiếp USART.

Bit 6 RX9 9-bit Receive Enable bit

RX9 = 1 nhận 9 bit dữ liệu.

RX9 = 0 nhận 8 bit dữ liệu.

Bit 5 SREN Single Receive Enable bit

Ở chế độ USART bất đồng bộ: bit này không cần quan tâm.

Ở chế độ USART Master đồng bộ:

SREN = 1 cho phép chức năng nhận 1 byte dữ liệu (8 bit hoặc 9 bit).

SREN = 0 không cho phép chức năng nhận 1 byte dữ liệu.

Bit 4 CREN Continuous Receive Enable bit

Ở chế độ bất đồng bộ:

CREN = 1 cho phép nhận 1 chuỗi dữ liệu liên tục.

CREN = 0 không cho phép nhận 1 chuỗi dữ liệu liên tục.

Ở chế độ đồng bộ:

CREN = 1 cho phép nhận dữ liệu cho tới khi xóa bit CREN.

CREN = 0 không cho phép nhận chuỗi dữ liệu.

Bit 3 ADDEN Address Detect Enable bit

Ở chế độ USART bất đồng bộ 9 bit

ADDEN = 1 cho phép xác nhận địa chỉ, khi bit RSR<8> được set thì ngắt được cho phép thực thi và giá trị trong buffer được nhận vào.

ADDEN = 0 không cho phép xác nhận địa chỉ, các byte dữ liệu được nhận vào và bit thứ 9 có thể được sử dụng như là bit parity.

- Bit 2 FERR Framing Error bit
FERR = 1 xuất hiện lỗi “Framing” trong quá trình truyền nhận dữ liệu.
FERR = 0 không xuất hiện lỗi “Framing” trong quá trình truyền nhận dữ liệu.
- Bit 1 OERR Overrun Error bit,
OERR = 1 xuất hiện lỗi “Overrun”
OERR = 0 không xuất hiện lỗi “Overrun”
- Bit 0 RX9D
Bit này chứa bit dữ liệu thứ 9 của dữ liệu truyền nhận.

P2.25 Thanh ghi XTREG: địa chỉ 19h

Thanh ghi đóng vai trò là buffer đệm 8 bit trong quá trình truyền dữ liệu thông qua chuẩn giao tiếp USART.

P2.26 Thanh ghi RCREG: địa chỉ 1Ah

Thanh ghi đóng vai trò là buffer đệm trong quá trình nhận dữ liệu qua chuẩn giao tiếp USART.

P2.27 Thanh ghi CCPR2L: địa chỉ 1Bh

Thanh ghi chứa 8 bit thấp của khối CCP2.

P2.28 Thanh ghi CCPR2H: địa chỉ 1Ch

Thanh ghi chứa 8 bit cao của khối CCP2.

P2.29 Thanh ghi ADRESH: địa chỉ 1Eh

Thanh ghi chứa byte cao của kết quả quá trình chuyển đổi ADC.

P2.30 Thanh ghi ADCON0: địa chỉ 1Fh

Đây là một trong hai thanh ghi điều khiển khối chuyển đổi ADC. Thanh ghi còn lại là thanh ghi ADCON1 (địa chỉ 9Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

Bit 7,6 ADCS1:ADCS0 A/D Conversion Clock Select bit

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	Frc (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	Frc (clock derived from the internal A/D RC oscillator)

Bit 5-3 CHS2:CHS0 Analog Channel Select bit

Các bit này dùng để chọn kênh chuyển đổi ADC

000 kênh 0 (AN0)

001 kênh 1 (AN1)

010 kênh 2 (AN2)

011 kênh 3 (AN3)

100 kênh 4 (AN4)

101 kênh 5 (AN5)

110 kênh 6 (AN6)

111 kênh 7 (AN7)

Bit 2 $\overline{GO/DONE}$ A/D Conversion Status bit

Khi $\overline{ADON} = 1$

$\overline{GO/DONE} = 1$ A/D đang hoạt động (set bit này sẽ làm khởi động ADC và tự xóa khi quá trình chuyển đổi kết thúc).

$\overline{GO/DONE} = 0$ A/D không hoạt động.

Bit 1 Không cần quan tâm và mặc định mang giá trị 0.

Bit 0 \overline{ADON} A/D On bit

$\overline{ADON} = 1$ bật A/D

$\overline{ADON} = 0$ tắt A/D

P2.31 Thanh ghi $\overline{OPTION_REG}$: địa chỉ 81h, 181h

Thanh ghi này cho phép điều khiển chức năng pull-up của các pin trong PORTB, xác lập các tham số về xung tác động, cạnh tác động của ngắt ngoại vi và bộ đếm Timer0.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
\overline{RBPUP}	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
bit 7							bit 0

Bit 7 \overline{RBPUP} PORTB pull-up enable bit

$\overline{RBPUP} = 1$ không cho phép chức năng pull-up của PORTB

$\overline{RBPUP} = 0$ cho phép chức năng pull-up của PORTB

Bit 6 INTEDG Interrupt Edge Select bit

INTEDG = 1 ngắt xảy ra khi cạnh dương chân RB0/INT xuất hiện.

INTEDG = 0 ngắt xảy ra khi cạnh âm chân BR0/INT xuất hiện.

Bit 5 TOCS Timer0 Clock Source select bit

TOCS = 1 clock lấy từ chân RA4/TOCK1.

TOCS = 0 dùng xung clock bên trong (xung clock này bằng với xung clock dùng để thực thi lệnh).

Bit 4 TOSE Timer0 Source Edge Select bit

TOSE = 1 tác động cạnh lên.

TOSE = 0 tác động cạnh xuống.

Bit 3 PSA Prescaler Assignment Select bit

PSA = 1 bộ chia tần số (prescaler) được dùng cho WDT

PSA = 0 bộ chia tần số được dùng cho Timer0

Bit 2:0 PS2:PS0 Prescaler Rate Select bit

Các bit này cho phép thiết lập tỉ số chia tần số của Prescaler.

Bit Value	TMR0 Rate	WDT Rate
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

P2.32 Thanh ghi TRISA: địa chỉ 85h

Thanh ghi điều khiển xuất nhập của các pin trong PORTA.

P2.33 Thanh ghi TRISB: địa chỉ 86h, 186h

Thanh ghi điều khiển xuất nhập của các pin trong PORTB.

P2.34 Thanh ghi TRISC: địa chỉ 87h

Thanh ghi điều khiển xuất nhập của các pin trong PORTC.

P2.35 Thanh ghi TRISD: địa chỉ 88h

Thanh ghi điều khiển xuất nhập của các pin trong PORTD.

P2.36 Thanh ghi TRISE: địa chỉ 89h

Thanh ghi điều khiển xuất nhập của các pin trong PORTE, điều khiển cổng giao tiếp song song PSP (Parallel Slave Port).

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	Bit 2	Bit 1	Bit 0
bit 7							bit 0

Bit 7 BIF Input Buffer Full Status bit

BIF = 1 một Word dữ liệu vừa được nhận và đang chờ CPU đọc vào.

BIF = 0 chưa có Word dữ liệu nào được nhận.

Bit 6 OBF Output Buffer Full Status bit

OBF = 1 Buffer truyền dữ liệu vẫn còn chứa dữ liệu cũ và vẫn chưa được đọc.

OBF = 0 Buffer truyền dữ liệu đã được đọc.

Bit 5 IBOV Input Buffer Overflow Detect bit

IBOV = 1 dữ liệu được ghi lên buffer trong khi dữ liệu cũ vẫn chưa được đọc.

IBOV = 0 buffer chưa bị tràn.

Bit 4 PSPMODE Parallel Slave Port Mode Select bit

PSPMODE = 1 Cho phép PSP, PORTD đóng vai trò là cổng giao tiếp song song PSP.

PSPMODE = 0 Không cho phép PSP.

Bit 3 Không cần quan tâm và mặc định mang giá trị 0.

Bit 2 Bit2 Direction Control for pin $\overline{RE2}/\overline{CS}/AN7$.

Bit2 = 1 Input

Bit2 = 0 Output

Bit 1 Bit1 Direction Control for pin $\overline{RE1}/\overline{WR}/AN6$

Bit1 = 1 Input

Bit1 = 0 Output

Bit 0 Bit0 Direction Control for pin $\overline{RE0}/\overline{RD}/AN5$

Bit0 = 1 Input

Bit0 = 0 Output

P2.37 Thanh ghi PIE1: địa chỉ 8Ch

Thanh ghi chứa các bit cho phép các ngắt ngoại vi.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Bit 7 PSPIE Parallel Slave Port Read/Write Interrupt Enable bit

PSPIE = 1 cho phép ngắt PSP read/write.

PSPIE = 0 không cho phép ngắt PSP read/write.

Bit 6 ADIE ADC (A/D converter) Interrupt Enable bit

ADIE = 1 cho phép ngắt ADC.

ADIE = 0 không cho phép ngắt ADC.

Bit 5 RCIE USART Receive Interrupt Enable bit

RCIE = 1 cho phép ngắt nhận USART

RCIE = 0 không cho phép ngắt nhận USART

Bit 4 TXIE USART Transmit Interrupt Enable bit

TXIE = 1 cho phép ngắt truyền USART

TXIE = 0 không cho phép ngắt truyền USART

Bit 3 SSPIE Synchronous Serial Port Interrupt Enable bit

SSPIE = 1 cho phép ngắt SSP

SSPIE = 0 không cho phép ngắt SSP

Bit 2 CCP1IE CCP1 Interrupt Enable bit

CCP1IE = 1 cho phép ngắt CCP1

CCP1IE = 0 không cho phép ngắt CCP1

Bit 1 TMR2IE TMR2 to PR2 Match Interrupt Enable bit

TMR2IE = 1 cho phép ngắt.

TMR2IE = 0 không cho phép ngắt.

Bit 0 TMR1IE TMR1 Overflow Interrupt Enable bit

TMR1IE = 1 cho phép ngắt.

TMR1IE = 0 không cho phép ngắt.

P2.38 Thanh ghi PIE2: địa chỉ 8Dh

Thanh ghi chứa các bit cho phép các ngắt ngoại vi.

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

Bit 7, 5, 2, 1 Không cần quan tâm và mặc định mang giá trị 0.

Bit 6 CMIE Comparator Interrupt Enable bit

CMIE = 1 Cho phép ngắt của bộ so sánh.

CMIE = 0 Không cho phép ngắt.

Bit 4 EEIE EEPROM Write Operation Interrupt Enable bit

EEIE = 1 Cho phép ngắt khi ghi dữ liệu lên bộ nhớ EEPROM.

EEIE = 0 Không cho phép ngắt khi ghi dữ liệu lên bộ nhớ EEPROM.

Bit 3 BCLIE Bus Collision Interrupt Enable bit

BCLIE = 1 Cho phép ngắt.

BCLIE = 0 Không cho phép ngắt.

Bit 0 CCP2IE CCP2 Interrupt Enable bit

CCP2IE = 1 Cho phép ngắt.

CCP2IE = 0 Không cho phép ngắt.

P2.39 Thanh ghi PCON: địa chỉ 8Eh

Thanh ghi điều khiển

chứa các cờ hiệu cho biết trạng thái các chế độ reset của vi điều khiển.

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-1
—	—	—	—	—	—	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

Bit 7, 6, 5, 4, 3, 2 Không cần quan tâm và mặc định mang giá trị 0.

Bit 1 $\overline{\text{POR}}$ Power-on Reset Status bit

$\overline{\text{POR}}$ = 1 không có sự tác động của Power-on Reset.

$\overline{\text{POR}}$ = 0 có sự tác động của Power-on reset.

Bit 0 $\overline{\text{BOR}}$ Brown-out Reset Status bit

$\overline{\text{BOR}}$ = 1 không có sự tác động của Brown-out reset.

$\overline{\text{BOR}}$ = 0 có sự tác động của Brown-out reset.

P2.40 Thanh ghi SSPCON2: địa chỉ 91h

Thanh ghi điều khiển các chế độ hoạt động của chuẩn giao tiếp I2C.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

Bit 7 GCEN General Call Enable bit

GCEN = 1 Cho phép ngắt khi địa chỉ 0000h được nhận vào thanh ghi SSPSR (địa chỉ của chế độ General Call Address).

- GCEN = 0 Không cho phép chế độ địa chỉ trên.
- Bit 6 ACKSTAT Acknowledge Status bit (bit này chỉ có tác dụng khi truyền dữ liệu ở chế độ I2C Master mode).
 ACKSTAT = 1 nhận được xung \overline{ACK} từ I2C Slave.
 ACKSTAT = 0 chưa nhận được xung \overline{ACK} .
- Bit 5 ACKDT Acknowledge Data bit (bit này chỉ có tác dụng khi nhận dữ liệu ở chế độ I2C Master mode).
 ACKDT = 1 chưa nhận được xung \overline{ACK} .
 ACKDT = 0 Đã nhận được xung \overline{ACK} .
- Bit 4 ACKEN Acknowledge Sequence Enable bit (bit này chỉ có tác dụng khi nhận dữ liệu ở chế độ I2C Master mode)
 ACKEN = 1 cho phép xung \overline{ACK} xuất hiện ở 2 pin SDA và SCL khi kết thúc quá trình nhận dữ liệu.
 ACKEN = 0 không cho phép tác động trên.
- Bit 3 RCEN Receive Enable bit (bit này chỉ có tác dụng ở chế độ I2C Master mode).
 RCEN = 1 Cho phép nhận dữ liệu ở chế độ I2C Master mode.
 RCEN = 0 Không cho phép nhận dữ liệu.
- Bit 2 PEN Stop Condition Enable bit
 PEN = 1 cho phép thiết lập điều kiện Stop ở 2 pin SDA và SCL.
 PEN = 0 không cho phép tác động trên.
- Bit 1 RSEN Repeated Start Condition Enable bit
 RSEN = 1 cho phép thiết lập điều kiện Start lặp lại liên tục ở 2 pin SDA và SCL.
 RSEN = 0 không cho phép tác động trên.
- Bit 0 SEN Start Condition Enable/Stretch Enable bit
 Ở chế độ Master mode:
 SEN = 1 cho phép thiết lập điều kiện Start ở 2 pin SDA và SCL.
 SEN = 0 không cho phép tác động trên.
 Ở chế độ Slave mode:
 SEN = 1 cho phép khóa xung clock từ pin SCL của I2C Master.
 Không cho phép tác động trên.

P2.41 Thanh ghi PR2: địa chỉ 92h

Thanh ghi dùng để ấn định trước giá trị đếm cho Timer2. Khi vi điều khiển được reset, PR2 mang giá trị FFh. Khi ta đưa một giá trị vào thanh ghi PR2, Timer2 sẽ đếm từ 00h cho đến khi giá trị bộ đếm của Timer2 bằng với giá trị của bộ đếm trong thanh ghi PR2. Như vậy mặc định Timer2 sẽ đếm từ 00h đến FFh.

P2.42 Thanh ghi SSPADD: địa chỉ 93h

Thanh ghi chứa địa chỉ của vi điều khiển khi hoạt động ở chuẩn giao tiếp I2C Slave mode. Khi không dùng để chứa địa chỉ (I2C Master mode) SSPADD được dùng để chứa giá trị tạo ra xung clock đồng bộ tại pin SCL.

P2.43 Thanh ghi SSPSTAT: địa chỉ 94h

Thanh ghi chứa các bit trạng thái của chuẩn giao tiếp MSSP.

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/A	P	S	R/W	UA	BF
bit 7							bit 0

Khi MSSP hoạt động ở chế độ SPI:

Bit 7 SMP Sample bit

SPI Master mode:

SMP = 1 dữ liệu được lấy mẫu (xác định trạng thái logic) tại thời điểm cuối xung clock.

SMP = 0 dữ liệu được lấy mẫu tại thời điểm giữa xung clock.

SPI Slave mode: bit này phải được xóa về 0.

Bit 6 CKE SPI Clock Select bit

CKE = 1 SPI Master truyền dữ liệu khi xung clock chuyển từ trạng thái tích cực đến trạng thái chờ.

CKE = 0 SPI Master truyền dữ liệu khi xung clock chuyển từ trạng thái chờ đến trạng thái tích cực.

(trạng thái chờ được xác định bởi bit CKP (SSPCON<4>).

Bit 5 $\overline{D/A}$ Data/Address bit.

Bit này chỉ có tác dụng ở chế độ I2C mode.

Bit 4 P Stop bit

Bit này chỉ sử dụng khi MSSP ở chế độ I2C.

Bit 3 S Start bit

Bit này chỉ có tác dụng khi MSSP ở chế độ I2C.

Bit 2 $\overline{R/W}$ Read/Write bit information

Bit này chỉ có tác dụng khi MSSP ở chế độ I2C.

Bit 1 UA Update Address bit

Bit này chỉ có tác dụng khi MSSP ở chế độ I2C.

Bit 0 BF Buffer Status bit

BF = 1 thanh ghi đệm SSPBUF đã có dữ liệu.

BF = 0 thanh ghi đệm SSPBUF chưa có dữ liệu.

Khi hoạt động ở chế độ I2C

Bit 7 SPM Slew Rate Control bit

SPM = 1 dùng tốc độ chuẩn (100 KHz và 1 MHz).

SPM = 0 dùng tốc độ cao (400 KHz).

Bit 6 CKE MSBus Select bit

CKE = 1 cho phép MSBus.

CKE = 0 không cho phép MSBus.

Bit 5 $\overline{D/A}$ Data/Address bit

I2C Master mode: không quan tâm.

- $\overline{D/A} = 1$ byte vừa truyền đi hoặc nhận được là dữ liệu.
 $\overline{D/A} = 0$ byte vừa truyền đi hoặc nhận được là địa chỉ.
- Bit 4 P Stop bit
 $P = 1$ vừa nhận được bit Stop.
 $P = 0$ chưa nhận được bit Stop.
- Bit 3 S Start bit
 $S = 1$ vừa nhận được bit Start.
 $S = 0$ chưa nhận được bit Start.
- Bit 2 $\overline{R/W}$ Read/Write bit information
I2C Slave mode:
 $\overline{R/W} = 1$ đọc dữ liệu.
 $\overline{R/W} = 0$ ghi dữ liệu.
I2C Master mode:
 $\overline{R/W} = 1$ đang truyền dữ liệu.
 $\overline{R/W} = 0$ không truyền dữ liệu.
- Bit 1 UA Update Address
Bit này chỉ có tác dụng đối với chế độ I2C Slave mode 10 bit địa chỉ.
 $UA = 1$ vi điều khiển cần cập nhật thêm địa chỉ từ thanh ghi SSPADD.
 $UA = 0$ không cần cập nhật thêm địa chỉ.
- Bit 0 BF Buffer Full Status bit
 $BF = 1$ Thanh ghi SSPBUF đang chứa dữ liệu truyền đi hoặc nhận được.
 $BF = 0$ thanh ghi SSPBUF không có dữ liệu.

P2.44 Thanh ghi TXSTA: địa chỉ 98h

Thanh ghi chứa các bit trạng thái và điều khiển việc truyền dữ liệu thông qua chuẩn giao tiếp USART.

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

- Bit 7 CSRC Clock Source Select bit
Ở chế độ bất đồng bộ: không cần quan tâm.
Ở chế độ đồng bộ:
 $CSRC = 1$ Master mode (xung clock được lấy từ bộ tạo xung BRG).
 $CSRC = 0$ Slave mode (xung clock được nhận từ bên ngoài).
- Bit 6 TX-9 9-bit Transmit Enable bit
 $TX-9 = 1$ truyền dữ liệu 9 bit.
 $TX-9 = 0$ truyền dữ liệu 8 bit.
- Bit 5 TXEN Transmit Enable bit
 $TXEN = 1$ cho phép truyền.
 $TXEN = 0$ không cho phép truyền.
- Bit 4 SYNC USART Mode Select bit

- SYNC = 1 dạng đồng bộ
 SYNC = 0 dạng bất đồng bộ.
- Bit 3 Không cần quan tâm và mặc định mang giá trị 0.
- Bit 2 BRGH High Baud Rate Select bit
 Bit này chỉ có tác dụng ở chế độ bất đồng bộ.
 BRGH = 1 tốc độ cao.
 BRGL = 0 tốc độ thấp.
- Bit 1 TRMT Transmit Shift Register Status bit
 TRMT = 1 thanh ghi TSR không có dữ liệu.
 TRMT = 0 thanh ghi TSR có chứa dữ liệu.
- Bit 0 TX9D
 Bit này chứa bit dữ liệu thứ 9 khi dữ liệu truyền nhận là 9 bit.

P2.45 Thanh ghi SPBRG: địa chỉ 99h

Thanh ghi chứa giá trị tạo xung clock cho bộ tạo xung BRG (Baud Rate Generator).
 Tần số xung clock do BRG tạo ra được tính theo các công thức trong bảng sau:

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = $F_{osc}/(64 (X + 1))$	Baud Rate = $F_{osc}/(16 (X + 1))$
1	(Synchronous) Baud Rate = $F_{osc}/(4 (X + 1))$	N/A

Trong đó X là giá trị chứa trong thanh ghi SRBRG.

Thanh ghi CMCON: địa chỉ 9Ch

Thanh ghi điều khiển và chỉ thị các trạng thái cũng như kết quả của bộ so sánh.

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

- Bit 7 C2OUT Comparator 2 (C2) Output bit
 Khi C2INV = 0
 C2OUT = 1 khi (pin V_{IN+} của C2) > (pin V_{IN-} của C2).
 C2OUT = 0 khi (pin V_{IN+} của C2) < (pin V_{IN-} của C2).
 Khi C2INV = 1
 C2OUT = 1 khi (pin V_{IN+} của C2) < (pin V_{IN-} của C2).
 C2OUT = 0 khi (pin V_{IN+} của C2) > (pin V_{IN-} của C2).
- Bit 6 C1OUT Comparator 1 (C1) Output bit
 Khi C1INV = 0
 C1OUT = 1 khi (pin V_{IN+} của C1) > (pin V_{IN-} của C1).
 C1OUT = 0 khi (pin V_{IN+} của C1) < (pin V_{IN-} của C1).
 Khi C1INV = 1
 C1OUT = 1 khi (pin V_{IN+} của C1) < (pin V_{IN-} của C1).
 C1OUT = 0 khi (pin V_{IN+} của C1) > (pin V_{IN-} của C1).
- Bit 5 C2INV Comparator 2 Output Conversion bit

C2INV = 1 ngõ ra C2 được đảo trạng thái.
 C2INV = 0 ngõ ra C2 không đảo trạng thái.

Bit 4 C1INV Comparator 1 Output Conversion bit

C1INV = 1 ngõ ra C1 được đảo trạng thái.
 C1INV = 0 ngõ ra C1 không đảo trạng thái.

Bit 3 CIS Comparator Input Switch bit

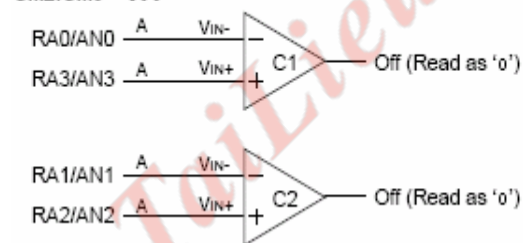
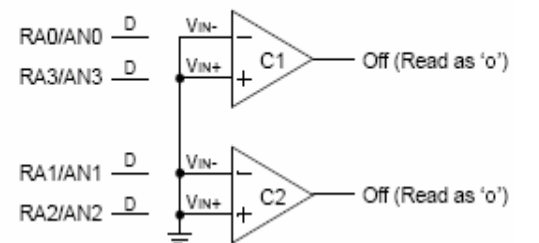
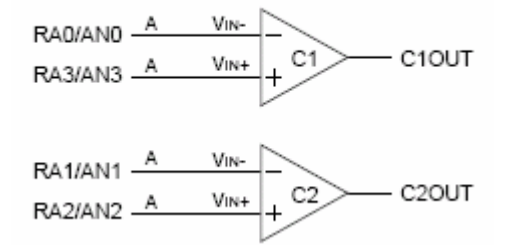
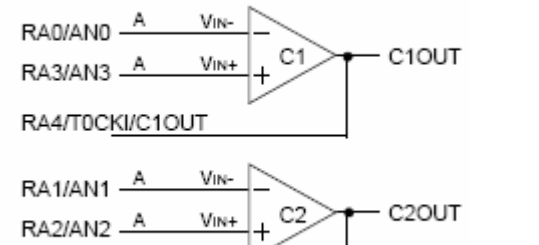
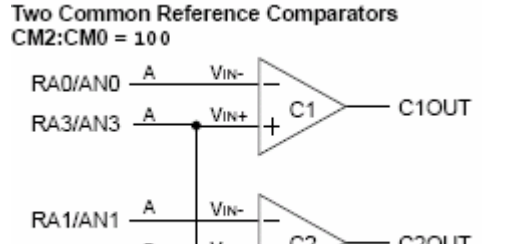
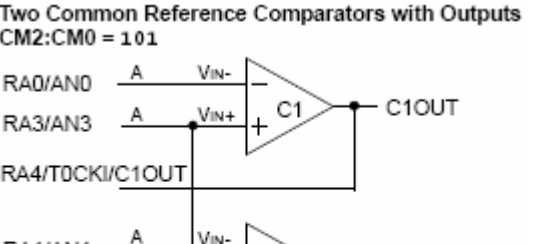
Bit này chỉ có tác dụng khi CM2:CM0 = 110

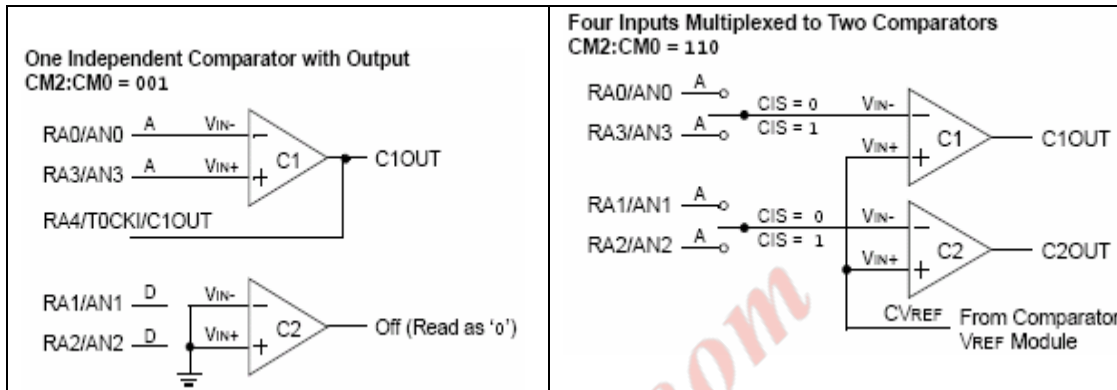
CIS = 1 khi pin V_{IN-} của C1 nối với RA3/AN3 và
 pin V_{IN-} của C2 nối với RA2/AN2

CIS = 0 khi pin V_{IN-} của C1 nối với RA0/AN0 và
 pin V_{IN-} của C2 nối với RA1/AN1

Bit 2-0 CM2:CM0 Comparator Mode bit

Các bit này đóng vai trò trong việc thiết lập các cấu hình hoạt động của bộ Comparator. Các dạng cấu hình của bộ Comparator được trình bày trong bảng sau:

Comparators Reset CM2:CM0 = 000 	Comparators Off (POR Default Value) CM2:CM0 = 111 
Two Independent Comparators CM2:CM0 = 010 	Two Independent Comparators with Outputs CM2:CM0 = 011 
Two Common Reference Comparators CM2:CM0 = 100 	Two Common Reference Comparators with Outputs CM2:CM0 = 101 



Trong đó: A là ngõ vào Analog, khi đó giá trị của các pin này đọc từ các PORT luôn bằng 0.
B là ngõ vào Digital.

P2.46 Thanh ghi CVRCON: địa chỉ 9Dh

Thanh ghi điều khiển bộ tạo điện áp so sánh khi bộ Comparator hoạt động với cấu hình '110'.

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0
bit 7							bit 0

Bit 7 CVREN Comparator Voltage Reference Enable bit.

CVREN = 1 bộ tạo điện áp so sánh được cấp điện áp hoạt động.

CVREN = 0 bộ tạo điện áp so sánh không được cấp điện áp hoạt động.

Bit 6 CVROE Comparator V_{REF} Output Enable bit

CVROE = 1 điện áp do bộ tạo điện áp so sánh tạo ra được đưa ra pin RA2.

CVROA = 0 điện áp do bộ tạo điện áp so sánh tạo ra không được đưa ra ngoài.

Bit 5 CVRR Comparator V_{REF} Range Selection bit

CVRR = 1 một mức điện áp có giá trị $V_{DD}/24$ (điện áp do bộ tạo điện áp so sánh tạo ra có giá trị từ 0 đến $0.75V_{DD}$).

CVRR = 0 một mức điện áp có giá trị $V_{DD}/32$ (điện áp do bộ tạo điện áp so sánh tạo ra có giá trị từ 0.25 đến $0.75V_{DD}$).

Bit 4 Không cần quan tâm và mặc định mang giá trị 0.

Bit 3-0 CVR3:CVR0 Các bit chọn điện áp ngõ ra của bộ tạo điện áp so sánh.

Khi CVRR = 1:

Điện áp tại pin RA2 có giá trị $CV_{REF} = (CVR<3:0>/24) * V_{DD}$.

Khi CVRR = 0

Điện áp tại pin RA2 có giá trị $CV_{REF} = (CVR<3:0>/32) * V_{DD} + 1/4V_{DD}$.

P2.47 Thanh ghi ADRESL: địa chỉ 9Eh

Thanh ghi chứa các bit thấp của kết quả bộ chuyển đổi A/D (8 bit cao chứa trong thanh ghi ADRESH địa chỉ 1Eh).

P2.48 Thanh ghi ADCON1: địa chỉ 9Fh

Thanh ghi chứa các bit điều khiển bộ chuyển đổi ADC (ADC có hai thanh ghi điều khiển là ADCON1 và ADCON0).

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Bit 7 ADFMA/D Result Format Select bit

ADFM = 1 Kết quả được lưu về phía bên phải 2 thanh ghi ADRESH:ADRESL (6 bit cao mang giá trị 0).

ADFM = 0 Kết quả được lưu về phía bên trái 2 thanh ghi ADRESH:ADRESL (6 bit thấp mang giá trị 0).

Bit 6 ADCS2 A/D Conversion Clock Select bit

ADCS2 kết hợp với 2 bit ADCS1:ADCS0 trong thanh ghi ADCON0 để điều khiển việc chọn xung clock cho khối chuyển đổi ADC.

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

Bit 5,4 Không cần quan tâm và mặc định mang giá trị 0.

Bit 3-0 PCFG3:PCFG0 A/D Port Configuration Control bit

Các bit này điều khiển việc chọn cấu hình hoạt động các cổng của bộ chuyển đổi ADC.

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

Trong đó A là ngõ vào Analog.

D là ngõ vào Digital.

C/R là số ngõ vào Analog/số điện áp mẫu.

P2.49 Thanh ghi EEDATA: địa chỉ 10Ch

Thanh ghi chứa byte thấp của dữ liệu trong quá trình ghi đọc trên bộ nhớ dữ liệu EEPROM.

P2.50 Thanh ghi EEADR: địa chỉ 10Dh

Thanh ghi chứa byte thấp của địa chỉ trong quá trình ghi đọc trên bộ nhớ dữ liệu EEPROM.

P2.51 Thanh ghi EEDATH: địa chỉ 10Eh

Thanh ghi chứa byte cao của dữ liệu trong quá trình ghi đọc trên bộ nhớ dữ liệu EEPROM (thanh ghi này chỉ sử dụng 6 bit thấp).

P2.52 Thanh ghi EEADRH: địa chỉ 10Fh

Thanh ghi chứa byte cao của địa chỉ trong quá trình ghi đọc trên bộ nhớ dữ liệu EEPROM (thanh ghi này chỉ sử dụng 4 bit thấp).

P2.53 Thanh ghi EECON1: địa chỉ 18Ch

Thanh ghi điều khiển bộ nhớ EEPROM.

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

- Bit 7 EEPGD Program/Data EEPROM Select bit
EEPGD = 1 truy xuất bộ nhớ chương trình.
EEPGD = 0 truy xuất bộ nhớ dữ liệu.
- Bit 6-4 Không cần quan tâm và mặc định mang giá trị 0.
- Bit 3 WRERR EEPROM Error Flag bit
WRERR = 1 quá trình ghi lên bộ nhớ bị gián đoạn và không thể tiếp tục (do các chế độ Reset WDT hoặc \overline{MCLR}).
WRERR = 0 quá trình ghi lên bộ nhớ hoàn tất.
- Bit 2 WREN EEPROM Write Enable bit
WREN = 1 cho phép ghi.
WREN = 0 không cho phép ghi.
- Bit 1 WR Write Control bit
WR = 1 ghi dữ liệu. Bit này chỉ được set bằng chương trình và tự động xóa về 0 khi quá trình ghi dữ liệu hoàn tất.
WR = 0 hoàn tất quá trình ghi dữ liệu.
- Bit 0 RD Read Control bit
RD = 1 đọc dữ liệu. Bit này chỉ được set bằng chương trình và tự động xóa về 0 khi quá trình đọc dữ liệu hoàn tất.
RD = 0 quá trình đọc dữ liệu không xảy ra.

P2.54 Thanh ghi EECON2: địa chỉ 18Dh.

Đây là một trong 2 thanh ghi điều khiển bộ nhớ EEPROM. Tuy nhiên đây không phải là thanh ghi vật lý thông thường và không cho phép người sử dụng truy xuất dữ liệu trên thanh ghi.

TÀI LIỆU THAM KHẢO

PIC16F877A DATASHEET WWW.MICROCHIP.COM
MIDRANGE PICmicro FAMILY WWW.MICROCHIP.COM

CÁC TRANG WEB

WWW.MICROCHIP.COM
WWW.DIENDANDIENTU.COM
WWW.PICVIETNAM.NET
VÀ CÁC TRANG WEB KHÁC

LỜI KẾT

Hi vọng sau khi đọc quyển sách này các đồng chí có những kiến thức cơ bản về vi điều khiển. Trên đó mình trình bày chủ yếu là bằng ngôn ngữ lập trình ASEMBLY còn các đồng chí nào đã nghiên cứu song môn nghiên cứu sang ngôn ngữ bậc cao như: C++, ... Thì gặp mình sẽ hỗ trợ và cung cấp các chương trình ví dụ để các đồng chí nghiên cứu.

TRÂN TRỌNG CẢM ƠN.

TaiLieuTuoi.com