LỜI MỞ ĐẦU

Ngày nay, cùng với sự phát triển không ngừng của cuộc cách mạng khoa học và kĩ thuật, ngành kĩ thuật điện tử đang dần khẳng định vai trò ngày càng lớn của mình đưa con người bước sang kỉ nguyên mới:kỉ nguyên số. Trong số đó ta không thể không nói đến kĩ thuật vi điều khiển. Kỹ thuật vi điều khiển đang có ứng dụng rộng rãi và thâm nhập vào nhiều lĩnh vực kỹ thuật và đời sống xã hội. Hầu hết các thiết bị kỹ thuật từ đơn giản cho đến phức tạp như thiết bị điều khiển tự động, thiết bị văn phòng, các thiết bị trong gia đình đều dùng các bộ vi điều khiển. Cùng với nó con người cũng ngày càng hoàn thiện các chuẩn giao tiếp để kết nối các thiết bị điện tử với nhau thực hiện việc trao đổi thông tin, điều khiển các cơ cấu chấp hành một cách thuận lợi hơn.

Với những kiến thức đã được học và những kiến thức cập nhập, nghiên cứu cùng với sự hướng dẫn của thầy giáo hướng dẫn. Em đã chọn đề tài: Thiết kế hệ thống giao tiếp I2C giữa hai vi điều khiển PIC.

Đồ án của em gồm 2 phần:

Chương một :Tổng quan về pic và giao tiếp I2C.

Chương hai :Thiết kế hệ thống giao tiếp I2C giữa 2 pic 16F877A

Trong quá trình làm đồ án tốt nghiệp, do hạn chế về thời gian, tài liệu nên không tránh khỏi những thiếu sót. Em rất mong được sự góp ý của thầy cô trong hội đồng và các bạn để đồ án tốt nghiệp của em được hoàn thiện hơn.

Em xin gửi lời cảm ơn chân thành đến nhà trường cùng thầy cô trong khoa Điện tử, đặc biệt là thầy Đoàn Hữu Chức đã giúp đỡ em hoàn thành đồ án này.

Hải Phòng, ngày 30 tháng 10 năm 2010

Sinh viên

Bùi Văn Nguyên

CHƯƠNG 1. TỔNG QUAN

1. Sơ lược về vi xử lý và vi điều khiển.

Trong những thập niên cuối thế kỉ XX, từ sự ra đời của công nghệ bán dẫn, kĩ thuật điện tử đã có sự phát triển vượt bậc. Các thiết bị điện tử sau đó đã được tích hợp với mật độ cao và rất cao trong các diện tích nhỏ, nhờ vậy các thiết bị điện tử nhỏ hơn và nhiều chức năng hơn. Các thiết bị điện tử ngày càng nhiều chức năng trong khi giá thành ngày càng rẻ hơn, chính vì vậy điện tử có mặt khắp mọi nơi.

Bước đột phá mới trong công nghệ điện tử, công ty Intel cho ra đời bộ vi xử lý đầu tiên, tức là phần cứng chỉ đóng vai trò thứ yếu, phần mềm (chương trình) đóng vai trò chủ đạo đối với các chức năng cần thực hiện. Nhờ vậy vi xử lý có sự mềm dẻo hóa trong các chức năng của mình. Ngày nay vi xử lý có tốc độ tính toán rất cao và khả năng xử lý rất lớn.

Vi xử lý có các khối chức năng cần thiết để lấy dữ liệu, xử lý dữ liệu và xuất dữ liệu ra ngoài sau khi đã xử lý. Và chức năng chính của Vi xử lý chính là xử lý dữ liệu, chẳng hạn như cộng, trừ, nhân, chia, so sánh.v.v....Vi xử lý không có khả năng giao tiếp trực tiếp với các thiết bị ngoại vi, nó chỉ có khả năng nhận và xử lý dữ liệu mà thôi.

Để vi xử lý hoạt động cần có chương trình kèm theo, các chương trình này điều khiển các mạch logic và từ đó vi xử lý xử lý các dữ liệu cần thiết theo yêu cầu.Chương trình là tập hợp các lệnh để xử lý dữ liệu thực hiện từng lệnh được lưu trữ trong bộ nhớ, công việc thực hành lệnh bao gồm: nhận lệnh từ bộ nhớ, giải mã lệnh và thực hiện lệnh sau khi đã giải mã.

Để thực hiện các công việc với các thiết bị cuối cùng, chẳng hạn điều khiển động cơ, hiển thị kí tự trên màn hình .... đòi hỏi phải kết hợp vi xử lý với các mạch điện giao tiếp với bên ngoài được gọi là các thiết bị I/O (nhập/xuất) hay còn gọi là các thiết bị ngoại vi. Bản thân các vi xử lý khi đứng một mình

không có nhiều hiệu quả sử dụng, nhưng khi là một phần của một máy tính, thì hiệu quả ứng dụng của Vi xử lý là rất lớn. Vi xử lý kết hợp với các thiết bị khác được sử trong các hệ thống lớn, phức tạp đòi hỏi phải xử lý một lượng lớn các phép tính phức tạp,có tốc độ nhanh. Chẳng hạn như các hệ thống sản xuất tự động trong công nghiệp, các tổng đài điện thoại, hoặc ở các robot có khả năng hoạt động phức tạp v.v..

Bộ Vi xử lý có khả năng vượt bậc so với các hệ thống khác về khả năng tính toán, xử lý, và thay đổi chương trình linh hoạt theo mục đích người dùng, đặc biệt hiệu quả đối với các bài toán và hệ thống lớn. Tuy nhiên đối với các ứng dụng nhỏ, tầm tính toán không đòi hỏi khả năng tính toán lớn thì việc ứng dụng vi xử lý cần cân nhắc. Bởi vì hệ thống dù lớn hay nhỏ, nếu dùng vi xử lý thì cũng đòi hỏi các khối mạch điện giao tiếp phức tạp như nhau. Các khối này bao gồm bộ nhớ để chứa dữ liệu và chương trình thực hiện, các mạch điện giao tiếp ngoại vi để xuất nhập và điều khiển trở lại, các khối này cùng liên kết với vi xử lý thì mới thực hiện được công việc. Để kết nối các khối này đòi hỏi người thiết kế phải hiểu biết tinh tường về các thành phần vi xử lý, bộ nhớ, các thiết bị ngoại vi. Hệ thống được tạo ra khá phức tạp,chiếm nhiều không gian, mạch in phức tạp và vấn đề chính là trình độ người thiết kế. Kết quả là giá thành sản phẩm cuối cùng rất cao, không phù hợp để áp dụng cho các hệ thống nhỏ.

Vì một số nhược điểm trên nên các nhà chế tạo tích hợp một ít bộ nhớ và một số mạch giao tiếp ngoại vi cùng với vi xử lý vào một IC duy nhất được gọi là Microcontroller-Vi điều khiển. Vi điều khiển có khả năng tương tự như khả năng của vi xử lý, nhưng cấu trúc phần cứng dành cho người dùng đơn giản hơn nhiều.Vi điều khiển ra đời mang lại sự tiện lợi đối với người dùng,họ không cần nắm vững một khối lượng kiến thức quá lớn như người dùng vi xử lý, kết cấu mạch điện dành cho người dùng cũng trở nên đơn giản hơn nhiều và có khả năng giao tiếp trực tiếp với các thiết bị bên ngoài.Vi điều khiển tuy được xây dựng với phần cứng dành cho người sử dụng đơn giản hơn, nhưng thay vào lợi

điểm này là khả năng xử lý bị giới hạn(tốc độ xử lý chậm hơn và khả năng tính toán ít hơn,dung lượng chương trình bị giới hạn). Thay vào đó, vi điều khiển có giá thành rẻ hơn nhiều so với vi xử lý, việc sử dụng đơn giản,do đó nó được ứng dụng rộng rãi vào nhiều ứng dụng có chức năng đơn giản, không đòi hỏi tính toán phức tạp.

Vi điều khiển được ứng dụng trong các dây chuyền tự động loại nhỏ, các robot có chức năng đơn giản, trong máy giặt, ôtô v.v...

Năm 1976 Intel giới thiệu bộ vi điều khiển (microcontroller) 8748, một chip tương tự như các bộ vi xử lý và là chip đầu tiên trong họ MCS-48. Độ phức tạp, kích thước và khả năng của Vi điều khiển tăng thêm một bậc quan trọng vào năm 1980 khi intel cho ra chip 8051,bộ Vi điều khiển đầu tiên của họ MCS-51 và là chuẩn công nghệ cho nhiều họ vi điều khiển được sản xuất sau này.Sau đó rất nhiều họ vi điều khiển của nhiều nhà chế tạo khác nhau lần lượt được đưa ra thị trường với tính năng được cải tiến ngày càng mạnh.

Các vi điều khiển thông dụng:

+ Họ vi điều khiển AMCC: do tập đoàn “Applied Micro Circuits Corporation” sản xuất. Tháng 5/2004, họ vi điều khiển này được phát triển và đưa ra thị trường bởi IBM, bao gồm:

403 PowerPC CPU

PPC 403GCX

405 PowerPC CPU

PPC 405EP

PPC 405GP/CR

PPC 405GPr

PPC NPe405H/L

440 PowerPC Book-E CPU

PPC 440GP

PPC 440GX

PPC 440EP/EPx/GRx

PPC 440SP/SPe

+ Họ vi điều khiển Atmel:

Dòng Atmel AT91 (Kiến trúc ARM THUMB)

Dòng AT90, Tiny & Mega – AVR (Atmel Norway design)

Dòng Atmel AT89 (Kiến trúc Intel 8051/MCS51)

Dòng MARC4

+ Họ vi điều khiển Freescale Semiconductor:

Năm 2004, những vi điều khiển này được phát triển và tung ra thị trường bởi Motorola.

Dòng 8-bit

68HC05 (CPU05)

68HC08 (CPU08)

68HC11 (CPU11)

Dòng 16-bit

68HC12 (CPU12)

68HC16 (CPU16)

Freescale DSP56800 (DSPcontroller)

Dòng 32-bit

Freescale 683XX (CPU32)

MPC500

MPC 860 (PowerQUICC)

MPC 8240/8250 (PowerQUICC II)

MPC 8540/8555/8560 (PowerQUICC III)

+ Họ vi điều khiển Intel

Dòng 8-bit

8XC42

MCS48

MCS51

8061

8xC251

Dòng 16-bit

80186/88

MCS96

MXS296

Dòng 32-bit

386EX

i960

+ Họ vi điều khiển Microchip

12-bit instruction PIC

14-bit instruction PIC

PIC16F84

16-bit instruction PIC

Trong đó họ vi điều khiển Microchip được ứng dụng phổ biến nhất, đặc biệt là PIC16F877A được tích hợp thêm những thành phần mới như bộ chuyển đổi A/D 10 bits, và lập trình phần mềm điều khiển cũng đơn giản hơn.

2.Tổng quan về vi điều khiển PIC

2.1.PIC là gì?

PIC là viết tắt của thuật ngữ "Programable Interlligent Compurter", có thể tạm dịch là "máy tính thông minh khả trình" do hãng Gnenral Intrusment đặt tên cho vi điều khiển đầu tiên đầu tiên của họ PIC 1650 được thiết kế 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 như ngày nay.

2.2 Đặc điểm của PIC so với các loại vi điều khiển khác

Hiện nay trên thị trường có rất nhiều loại vi điều khiển khác nhau như: 8051. Motorola 68HC, AVR, ARM…Tuy vậy PIC vẫn được sử dụng ngày càng được phổ biến bởi:

- Dễ dàng mua được ở thị trường Viêt Nam.

- Giá thành không quá đắt.

- Có đầy đủ tính năng của 1 vi điều khiển khi hoạt động độc lập.

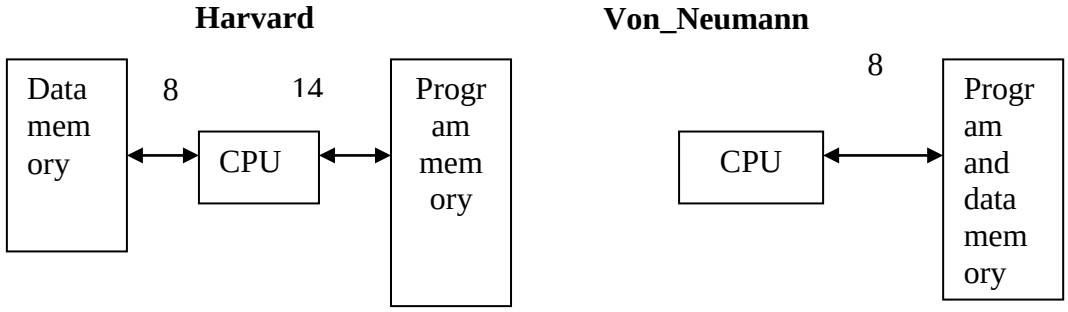
- Là sự bổ xung tốt về kiến thức cũng như về ứng dụng cho họ vi điều khiển mang tính truyền thống như 8051.

- Có sự hỗ trợ của nhà sản xuất về trình biên dịch, các công cụ lập trình,mạch nạp từ đơn giản đến mức cao.

- Các tính năng đa dạng của vi điều khiển PIC,ngày càng được mở rộng, phát triển.

2.3 Kiến trúc của PIC

Kiến trúc phần cứng của vi điều khiển được thiết kế theo 2 dạng kiến trúc:kiến trúc Von Neuman và kiến trúc Havard



Hình 1.1. Kiến trúc Harvard và kiến trúc Von-Neumann.

Tổ chức phần cứng của PIC được tổ chức theo kiểu kiến trúc Harvard. Điểm khác biệt giữa kiến trúc Harvard và Von-Neumann là cấu trúc bộ nhớ dữ liệu và bộ nhớ chương trình.

Đối với kiến trúc Von-Neumann, bộ nhớ dữ liệu và bộ nhớ chương trình nằm chung trong cùng 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 lí của CPU là 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 cấu trúc Von-Neumann không phù hợp với cấu trúc của vi điều khiển.

Đối với cấu trúc Harvard, bộ nhớ chương trình và bộ nhớ dữ liệu tách thành hai bộ nhớ riêng biệt.Do đó cùng một thời điểm CPU có thể tương tác với cả hai bộ nhớ,như vậy tốc độ xử lí được cải thiện đáng kể.

Một điểm cần chú ý nữa là tập lệnh trong kiến trúc Harvard 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 với vi điều khiển dòng 16F độ dài luôn là 14 bít (trong khi dữ liệu đươc tổ chức thành từng byte). Đặc điểm này được minh họa trong hình 1.1.

2.4. RISC và CISC

Như trên,kiến trúc Harvard là khái niệm mới hơn so với kiến trúc Von-Neumann.Khái niệm này được cải thiện nhằm cải tiến tốc độ thực thi của 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 lí dữ liệu nên 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 khiển.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 với họ 16Fxxxx chiều dài mã lệnh luôn là 14 bit),và cho phép thực thi trong một chu kì xung clock (ngoại trừ một số trường hợp đặc biệt như lệnh nhảy, lệnh gọi chương trình con-cần hai xung đồng hồ). Điều này có nghĩa là 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 hoá 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 (Reducer Instruction Set Computer) hay là vi điều khiển có tập lệnh rút gọn. Vi điều khiển được thiết kế theo kiểu 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 lệnh của nó không phải là một số cố định mà luôn là bội số của 8bit (1 byte).

2.5. PIPELINING (xử lí song song)

Đây chính là cơ chế xử lí lệnh của các vi điều khiển PIC.Một chu kì 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 có tần số là 1 MHZ (chu ki lệnh là 1 ns). Giả sử ta có 1 đoạn chương trình như sau:

1.MOVLW 55h

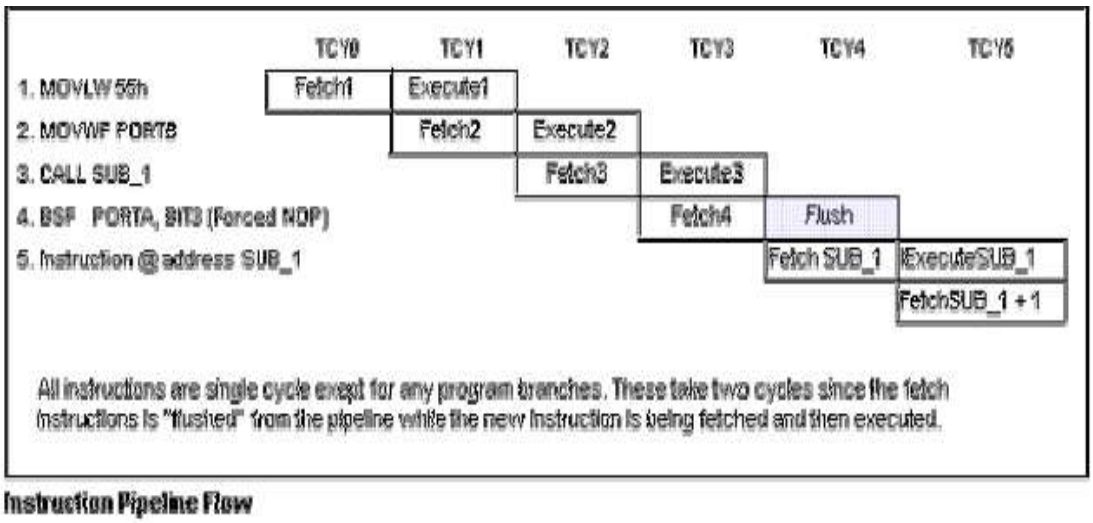
2.MOVWF PORT B

3.CALL SUB_1

4.BSFPORT A,BIT 3

5. instruction @ address SUB_1

Ở đây ta chỉ bàn đến quy trình vi điều khiển xử lí đoạn chương trình trên thông qua từng chu kì lệnh.quá trình 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 lênh 4.

TCY4: vì lệnh 4 không phải là lệnh sẽ được thực thi theo quy trình thực thi của chương trình (lệnh tiếp theo được thực thi phải là lệnh đầu tiên tại label SUB_1) nên chu trình thực thi của lệnh này chỉ được dùng để đọc lệnh đầu tiên tại label SUB_1. Như vậy có thể xem lệnh 3 cần 2 chu kì 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 kì lệnh để gọi lệnh đó, và một chu kì 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 kì lệnh. Đối với các lệnh mà quá trình thực thi làm thay đổi giá trị thanh PC (Program Counter) cần hai chu kì lệnh để thực thi vì phải thực hiện việc gọi lệnh ở địa chỉ thanh PC chỉ tới.Sau khi đã xác định đúng vị trí lệnh trong thanh ghi PC,mỗi lệnh chỉ cần một chu kì lệnh để thực thi xong.

2.6. Các dòng PIC và cách lựa chọn vi điều khiển PIC

Các kí hiệu của vi điều khiển PIC:

PIC12xxxx : độ dài lệnh 12 bít.

PIC16xxxx: độ dài lệnh 14 bit.

PIC18xxxx: độ dài lệnh 16 bit.

C : PIC có bộ nhớ EPROM.

F : PIC có bộ nhớ plash.

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

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

Bên cạnh đó có một số vi điều khiển có kí hiệu là xxFxxx là EEPROM, nếu có thêm chữ A ở cuối là plash (ví dụ 16F877 là EEEROM, 16F877A là plash).

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 cho phù hợp:

- Trước hết cần biết số chân của vi điều khiển cần thiết cho ứng dụng. Có nhiều vi điều khiển PIC có số lượng chân khác nhau, có vi điều khiển chỉ có 8 chân, có loại 28, 40, 44… chân.

- Cần chọn vi điều khiển PIC có bộ nhớ plash để có thể nạp xoá chương trình được nhiều lần hơn.

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

- Bộ nhớ chương trình mà vi điều khiển cho phép.

2.7. 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 một số ngôn ngữ lập trình dành riêng cho PIC như PICBasix, MikroBasix…

2.8. 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 sử dụng 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 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 PIC như sau:

- JDM Programmer: mạch nạp này dành cho chương trình 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 trinh này.

- WARP-13A và MCP – USB: hai mạch nạp này giống như 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.

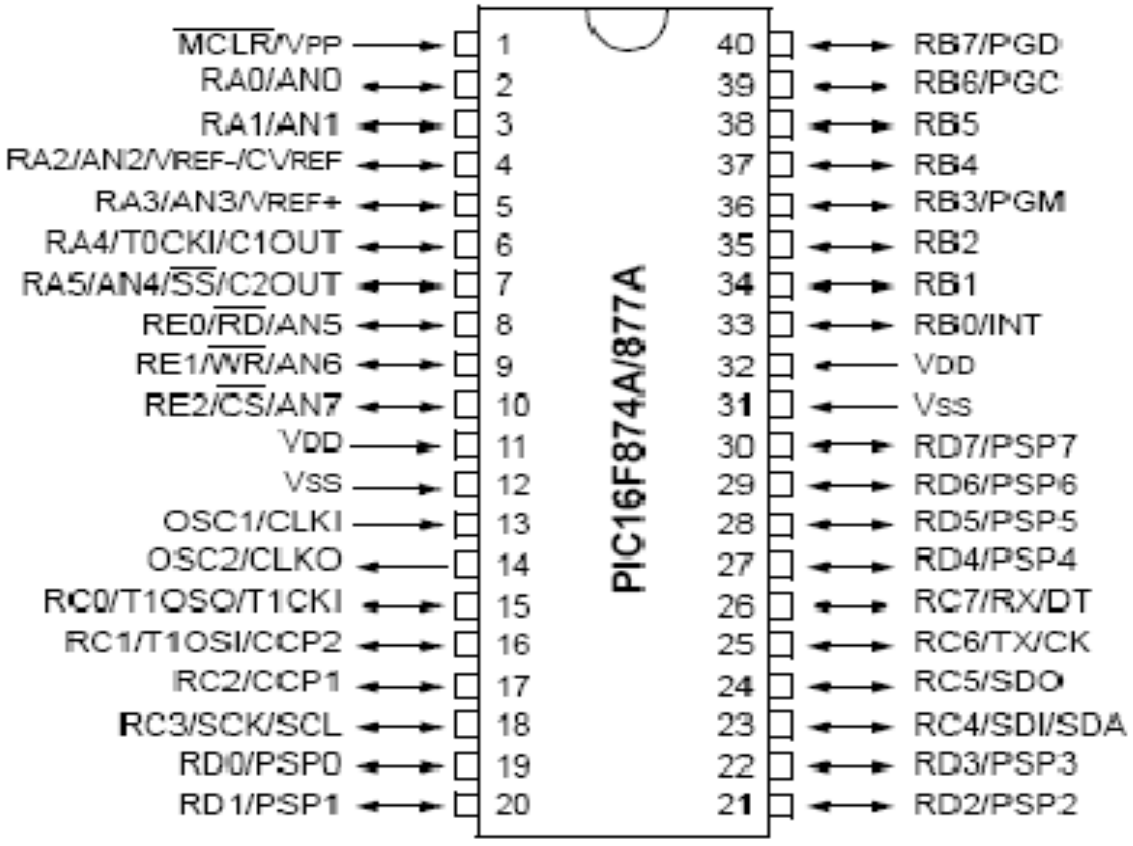
- Mạch nạp Universal của Williem, đây không phải mạch nạp chuyên dụng dành cho PIC như P16PR040.

Các mạch nạp kể trên đều có ưu điểm rất lớn là đơn giản, rẻ tiền, hoàn toàn có thể tự lắp ráp một cách dễ dàng, mọi thông tin về sơ đồ mạch nạp, thiết kế thi công, kiểm tra và chương trình nạp dễ dàng tìm được. Tuy nhiên các mạch nạp trên vẫn còn một số 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.

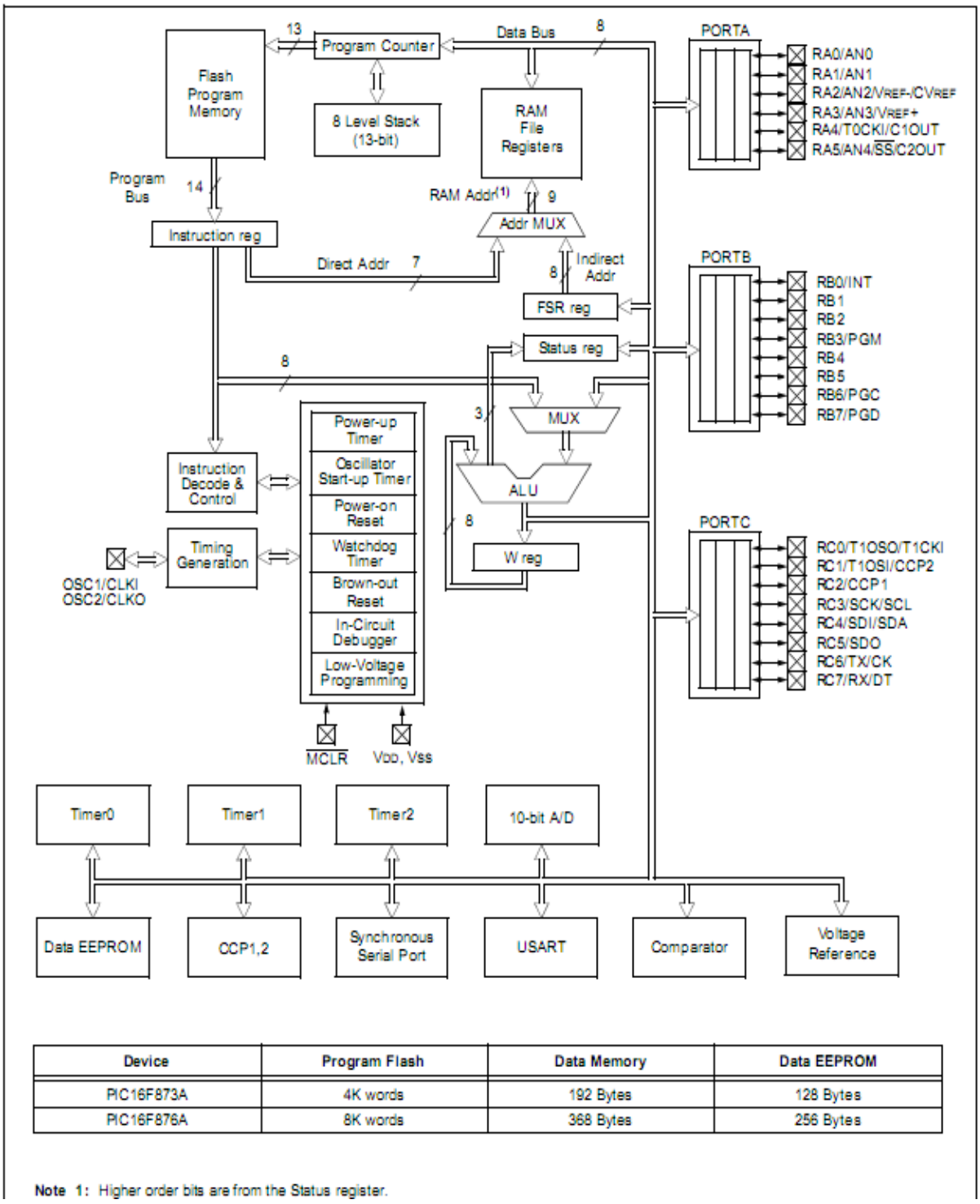
Sau khi đã trình bày tổng quan về vi điều khiển PIC. Em xin trình bày về một loại vi điều khiển PIC được sử dụng rất rộng rãi hiện nay đó là 16F877A.

3. Tổng quan về PIC 16F877A

3.1. Sơ đồ khối và bảng mô tả chức năng các chân của PIC16F877A



Hình 1.3a. PIC 16F877A



Device	Program Flash	Data Memory	Data EEPROM
PIC16F873A	4K words	192 Bytes	128 Bytes
PIC16F876A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

Hình 1.3.b. Sơ đồ khối của PIC16F877A

Bảng mô tả chức năng các chân của PIC16F877A

Pin Name	DIP Pin#	PLCC Pin#	QFT Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	1	ST/CMOS(4)	Đầu vào của xung dao động thạch anh/ngõ vào xung clock ngoại
OSC2/CLKOUT	1	2	18	O	-	Đầu ra của xung dao động thạch anh. Nối với thạch anh hay cộng hưởng trong chế độ dao động của thạch anh.Trong chế độ RC, ngõ ra của chân OSC2.
$\overline{MCLR}$ /V_{pp}	1	2	18	I/P	ST	Ngõ vào của Master Clear(Reset) hoặc ngõ vào điện thế được lập trình. Chân này cho phép tín hiệu Reset thiết bị tác động ở mức thấp.
RA0/AN0	2	3	19	I/O	TTL	PORTA là port vào ra hai chiều. RA0 có thể làm ngõ vào tương tự thứ 0.
RA1/AN1	3	4	20	I/O	TTL	RA1 có thể làm ngõ vào tương tự thứ 1
RA2/AN2/VREF–	4	5	21	I/O	TTL	RA2 có thể làm ngõ vào tương tự 2 hoặc điện áp chuẩn tương tự âm.
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 có thể làm ngõ vào tương tự 3 hoặc điện áp chuẩn tương tự dương.
RA4/T0CKI	6	7	23	I/O	ST	RA4 có thể làm ngõ vào xung clock cho bộ định thời Timer0.
RA5/$\overline{SS}$/AN4	7	8	24	I/O	TTL	RA5 có thể làm ngõ vào tương tự thứ 4
RB0/INT RB1 RB2	33 34 35	36 37 38	8 9 10	I/O I/O I/O	TTL/ST(1) TTL TTL	PORTB là port hai chiều. RB0 có thể làm chân ngắt ngoà
RB3/PGM	36	39	11	I/O	TTL	RB3 có thể làm ngõ vào của điện thế được lập trình ở mức thấp.

RB4 RB5 RB6/PGC RB7/PGD	37 38 39 40	41 42 43 44	14 15 16 17	I/O I/O I/O I/O	TTL TTL TTL/ST(2) TTL/ST(3)	Interrupt-on-change pin. Interrupt-on-change pin. Interrupt-on-change pin hoặc In-Crcuit Debugger pin . Serial programming clock. Interrupt-on-change pin hoặc In-Crcuit Debugger pin . Serial programming data .
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	PORTC là port vào ra hai chiều. RC0 có thể là ngõ vào của bộ dao động Timer1 hoặc ngõ xung clock cho Timer1
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 có thể là ngõ vào của bộ dao động Timer1 hoặc ngõ vào Capture2/ngõ ra compare2/ngõ vào PWM2.
RC2/CCP1	17	19	36	I/O	ST	RC2 có thể ngõ vào capture1/ngõ ra compare1/ngõ vào PWM1
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 có thể là ngõ vào xung
RC4/SDI/SDA	23	25	42	I/O	ST	Clock đồng bộ nội tiếp/ngõ ra trong cả hai chế độ SPI và I2C RC4 có thể là dữ liệu bên trong SPI(chế độ SPI) hoặc dữ liệu I/O(chế độ I^2C).
RC5/SDO	24	26	43	I/O	ST	RC5 có thể là dữ liệu ngoài SPI(chế độ SPI)
RC6/TX/CK	25	27	44	I/O	ST	RC6 có thể là chân truyền không đồng bộ USART hoặc đồng bộ

						với xung đồng hồ
RC7/RX/DT	26	29	1	I/O	ST	RC7 có thể là chân nhận không đồng bộ USART hoặc đồng bộ với dữ liệu.
RD0/PSP0 RD1/PSP1 RD2/PSP2 RD3/PSP3 RD4/PSP4 RD5/PSP5 RD6/PSP6 RD7/PSP7	19 20 21 22 27 28 29 30	21 22 23 24 30 31 32 33	38 39 40 41 2 3 4 5	I/O I/O I/O I/O I/O I/O I/O I/O	ST/TTL(3) ST/TTL(3) ST/TTL(3) ST/TTL(3) ST/TTL(3) ST/TTL(3) ST/TTL(3) ST/TTL(3)	PORTD là port vào ra hai chiều hoặc là parallel slave port khi giao tiếp với bus của bộ vi xử lý.
RE0/$\overline{RD}$/AN5	8	9	25	I/O	ST/TTL(3)	PORTE là port vào ra hai chiều. RE0 có thể điều khiển việc đọc parrallel slave port hoặc là ngoc vào tương tự thứ 5.
RE1/$\overline{WR}$/AN6	9	10	26	I/O	ST/TTL(3)	RE1 có thể điều khiển việc ghi parallel slave port hoặc là ngõ vào tương tự thứ 6.
RE2/$\overline{CS}$/AN7	10	11	27	I/O	ST/TTL(3)	RE2 có thể điều khiển việc chọn parallel slave port hoặc là ngõ vào tương tự thứ 7
V_{ss} V_{DD}	12, 31 11, 32	13, 34 12, 35	7, 28 6, 29	P P		Cung cấp nguồn dương cho các mức logicvà những chân I/O.
NC		1,17,28, 40	12,13 33, 4			Những chân này không được nối bên trong và nó được để trống

Ghi chú: I = input; O = output; I/O = input/output; P = power

- = Not used; TTL = TTL input; ST = Schmitt Trigger input

1. Là vùng đệm có ngõ vào Trigger Schmitt khi được cấu hình như ngắt ngoài.

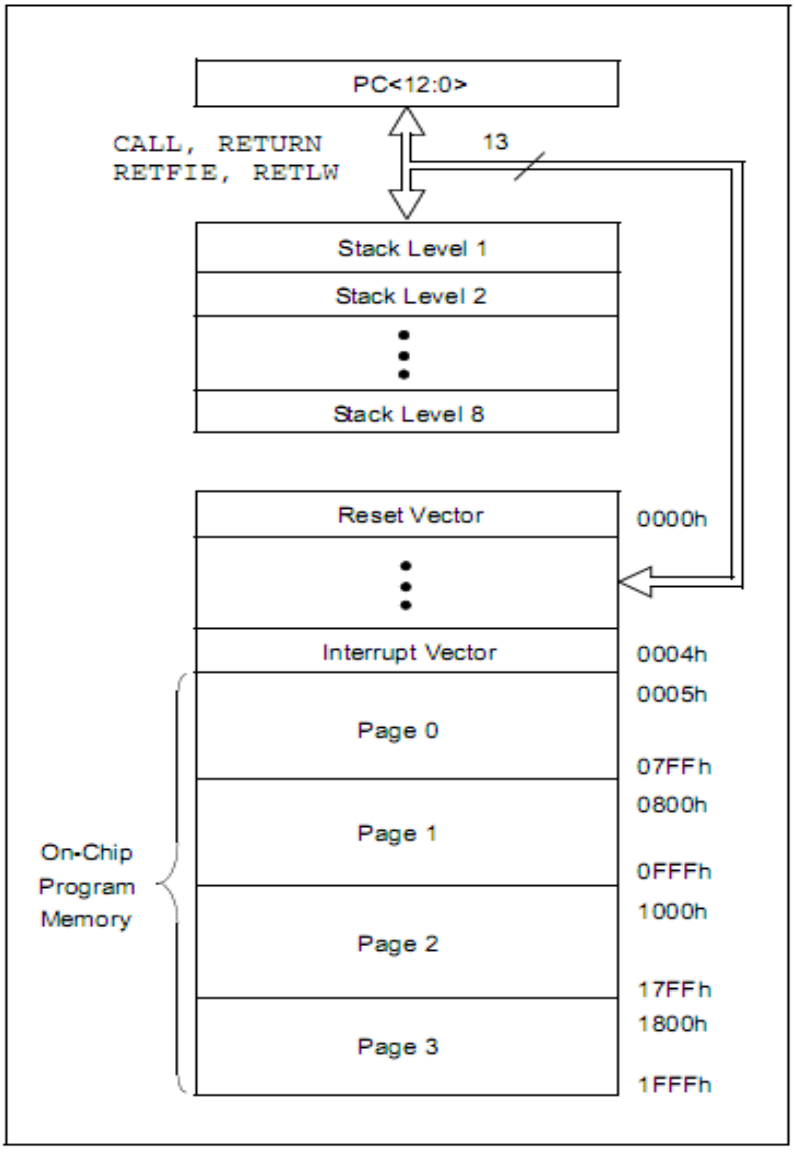
2. Là vùng đệm có ngõ vào Trigger Schmitt khi được sử dụng trong chế độ 9 Serial Programming.

3. Là vùng đệm có ngõ vào Trigger Schmitt khi được cấu hình như ngõ vào ra mục đích chung và là ngõ vào TTL khi sử dụng trong chế độ Parallel Slave Port (cho việc giao tiếp với các bus của bộ vi xử lý).

4. Là vùng đệm có ngõ vào Trigger Schmitt khi được cấu hình trong chế độ dao động RC và một ngõ vào CMOS khác.

3.2. Tổ chức bộ nhớ

Có 2 khối bộ nhớ trong các vi điều khiển họ PIC16F87X, bộ nhớ chương trình và bộ nhớ dữ liệu, với những bus riêng biệt để có thể truy cập đồng thời.



Hình 1.4. Ngăn xếp và bản đồ bộ nhớ chương trình PIC16F877A

3.2.1. Tổ chức của bộ nhớ chương trình

Các vi điều khiển họ PIC16F877A có bộ đếm chương trình 13 bit có khả năng định vị không gian bộ nhớ chương trình lên đến 8Kb.Các IC PIC16F877A có 8Kb bộ nhớ chương trình FLASH, các IC PIC16F873/874 chỉ có 4 Kb.Vectơ RESET đặt tại địa chỉ 0000h và vectơ ngắt tại địa chỉ 0004h.

3.2.2. Tổ chức bộ nhớ dữ liệu

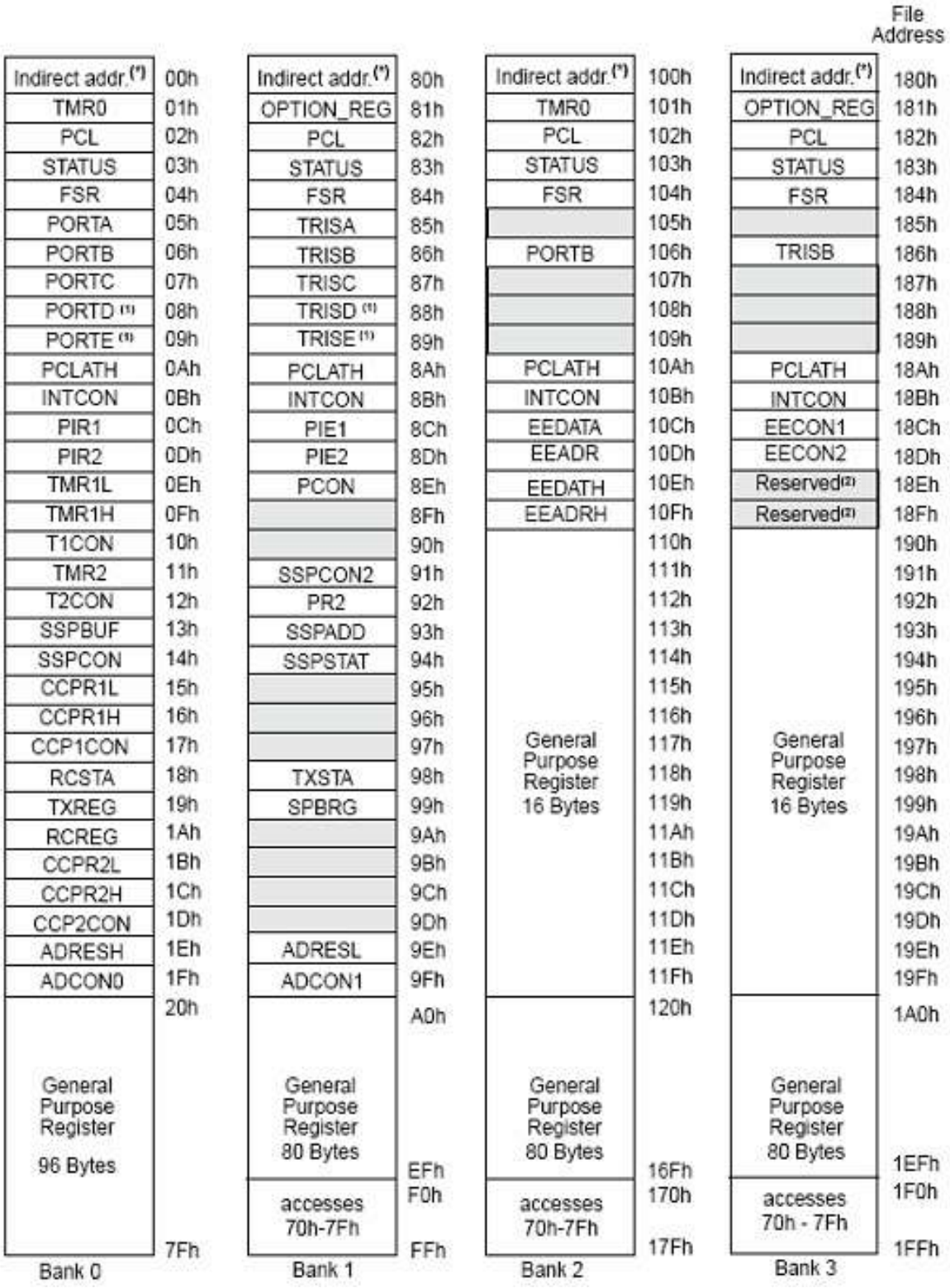
Bộ nhớ dữ liệu được chia thành nhiều dãy và chứa các thanh ghi mục đích chung và các thanh ghi chức năng đặc biệt. BIT RP1 (STATUS <6>) và RP0 (STATUS <5>) là những bit dùng để chọn các dãy thanh ghi.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

Chiều dài của mỗi dãy là 7Fh (128 byte). Phần thấp của mỗi dãy dùng để chứa các thanh ghi chức năng đặc biệt.Trên các thanh ghi chức năng đặc biệt là các thanh ghi mục đích chung, có chức năng như RAM tĩnh. Thường thì những thanh ghi đặc biệt được sử dụng từ một dãy và có thể được ánh xạ vào những dãy khác để giảm bớt đoạn mã và khả năng truy cập nhanh hơn.

3.2.3. Các thanh ghi mục đích chung

Các thanh ghi này có thể truy cập trực tiếp hoặc gián tiếp thông qua thanh ghi FSG (File Select Register).



Hình 1.5. Các thanh ghi của PIC16F877A

3.2.4. Các thanh ghi chức năng đặc biệt

Các thanh ghi chức năng đặc biệt (Special Function Resgister) được sử dụng bởi CPU và các bộ nhớ ngoại vi để điều khiển các hoạt động được yêu cầu của thiết bị. Những thanh ghi này có chức năng như RAM tĩnh. Danh sách những thanh ghi nay được trình bày ở bảng dưới. Các thanh ghi chức năng đặc biệt có thể chia thành hai loại: phần trung tâm (CPU) và phần ngoại vi.

3.2.5. Các thanh ghi trạng thái

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:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h-1FFh)
0 = Bank 0, 1 (00h-FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h-1FFh)
10 = Bank 2 (100h-17Fh)
01 = Bank 1 (80h-FFh)
00 = Bank 0 (00h-7Fh)
Each bank is 128 bytes.

bit 4 $\overline{TO}$: Time-out bit
1 = After power-up, CLRWDT instruction or SLEEP instruction
0 = A WDT time-out occurred

bit 3 $\overline{PD}$: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

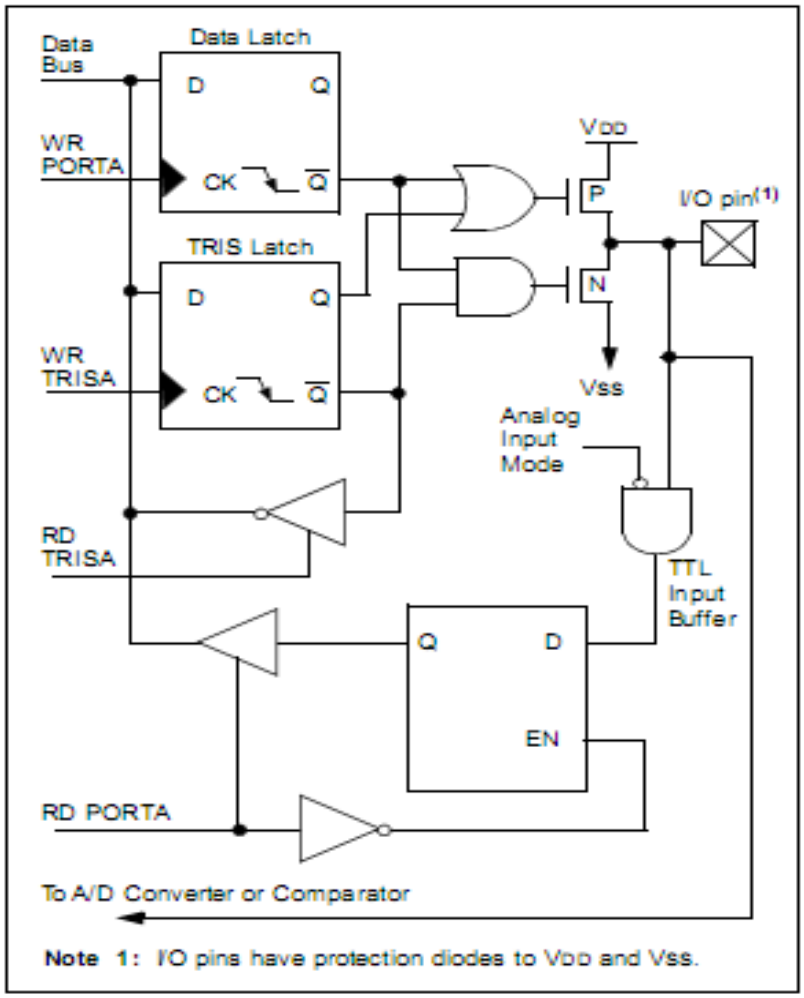
Hình 1.6. Thanh ghi trạng thái (địa chỉ 03h, 83h, 103h, 183h)

Thanh ghi trạng thái chứa các trạng thái số học của bộ ALU, trạng thái RESET và những bits chọn dãy thanh ghi cho bộ nhớ dữ liệu. Thanh ghi trạng thái có thể là đích cho bất kì lệnh nào, giống như những thanh ghi khác. Nếu thanh ghi trang thái là đích cho một lệnh mà ảnh hưởng đến các cờ Z, DC hoặc

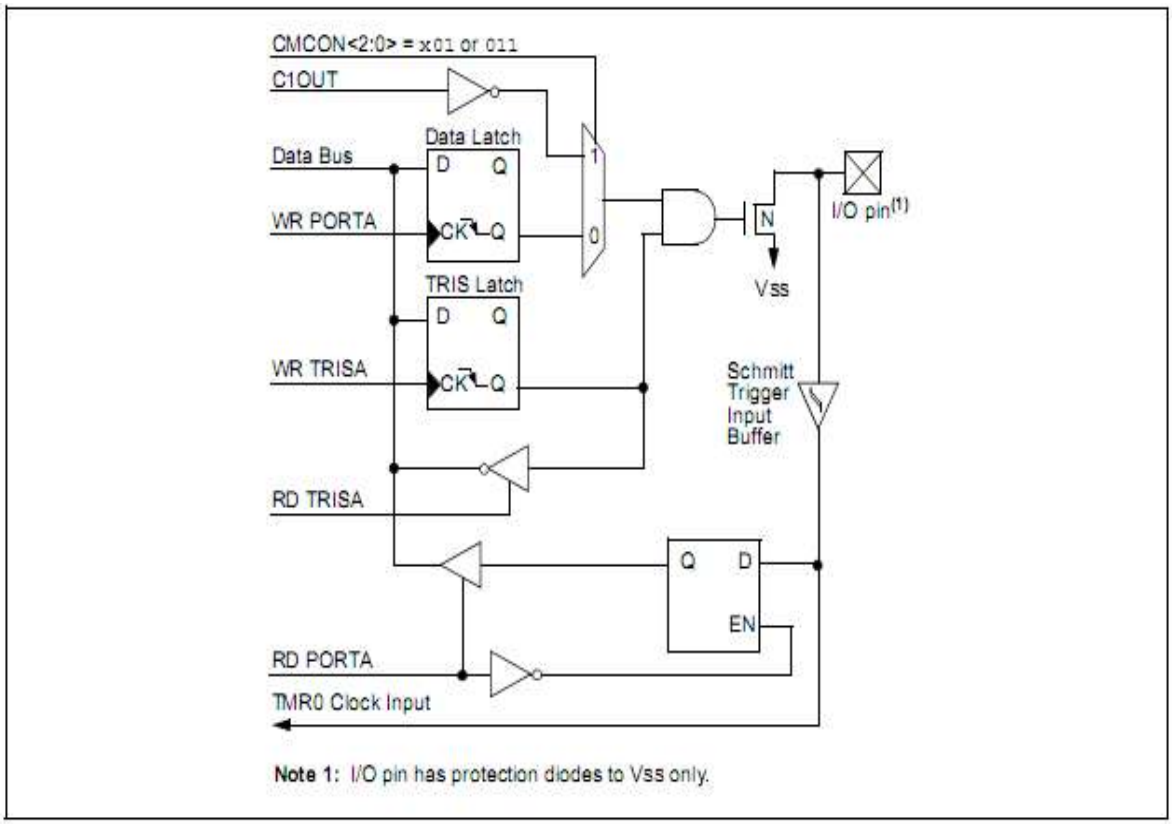
C, và sau đó những bit này sẽ được vô hiệu hoá. Những bit này có thể đặt hoặc xoá tuỳ theo trạng thái logic của thiết bị. Hơn nữa hai bit $\overline{TO}$ và PD thì không cho phép ghi, vì vậy kết quả của một tập lệnh mà thanh ghi trạng thái là đích có thể khác hơn dự định. Ví dụ, CLRF STATUS sẽ soá 3 bit cao nhất và đặt bit Z. Lúc này các bit của thanh ghi trạng thái là 000u u1uu (u = unchanged). Chỉ có các lệnh BCF, BSF, SWAPF và MOVWF được sử dụng để thay đổi thanh ghi trạng thái, bởi vì những lệnh này không làm ảnh hưởng đến các bit Z, DC hoặc C từ thanh ghi trạng thái. Đối với những lệnh khác thì không ảnh hưởng đến những bit trạng thái này.

3.3. Các cổng của PIC 16F877A

3.3.1. PORTA và thanh ghi TRISA



Hình 1.7.a. Sơ đồ khối của chân RA3: RA0 và RA5



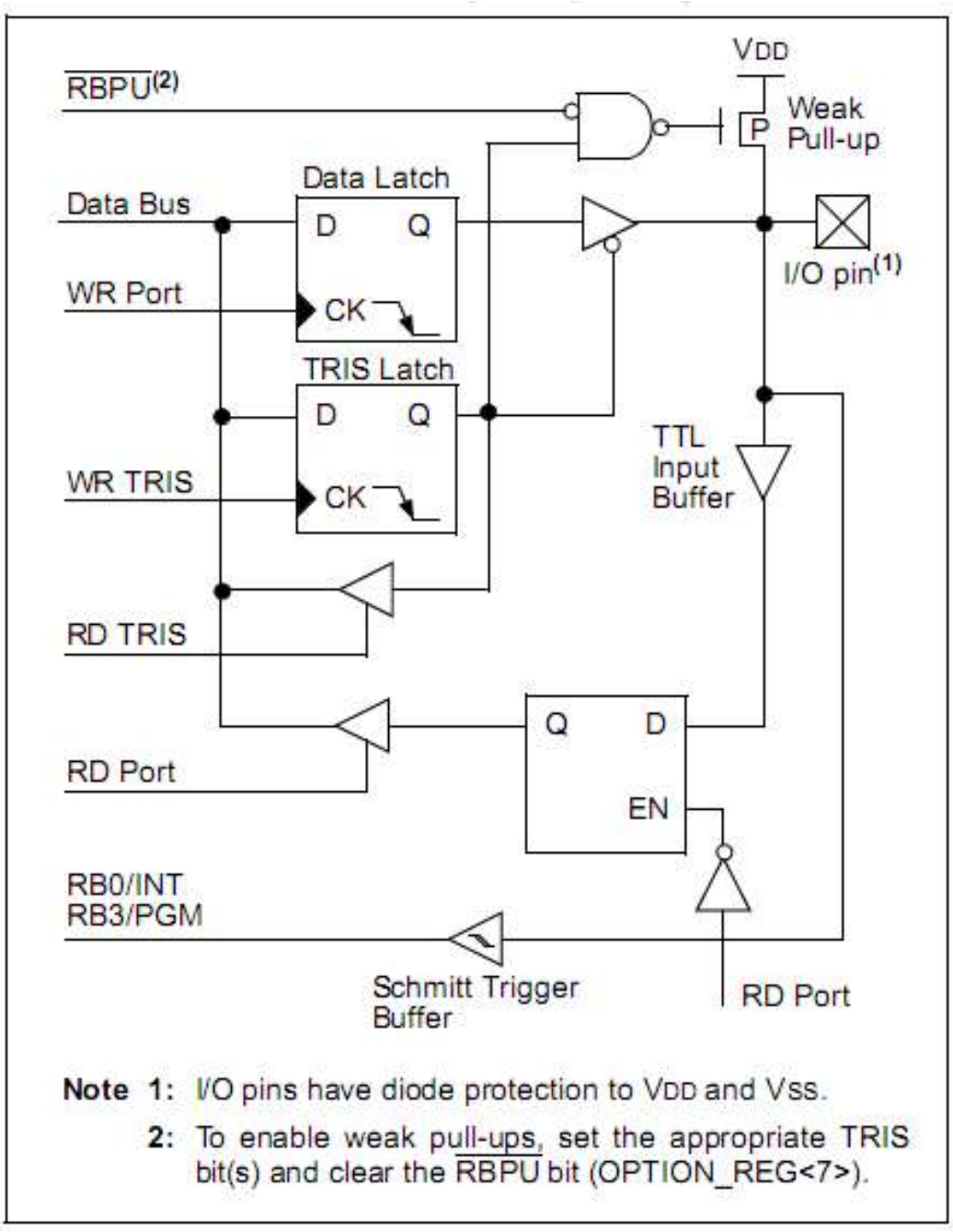
Hình 1.7.b. Sơ đồ khối của chân RA4/T0CKI

3.3.2 PORTB và thanh ghi TRISB

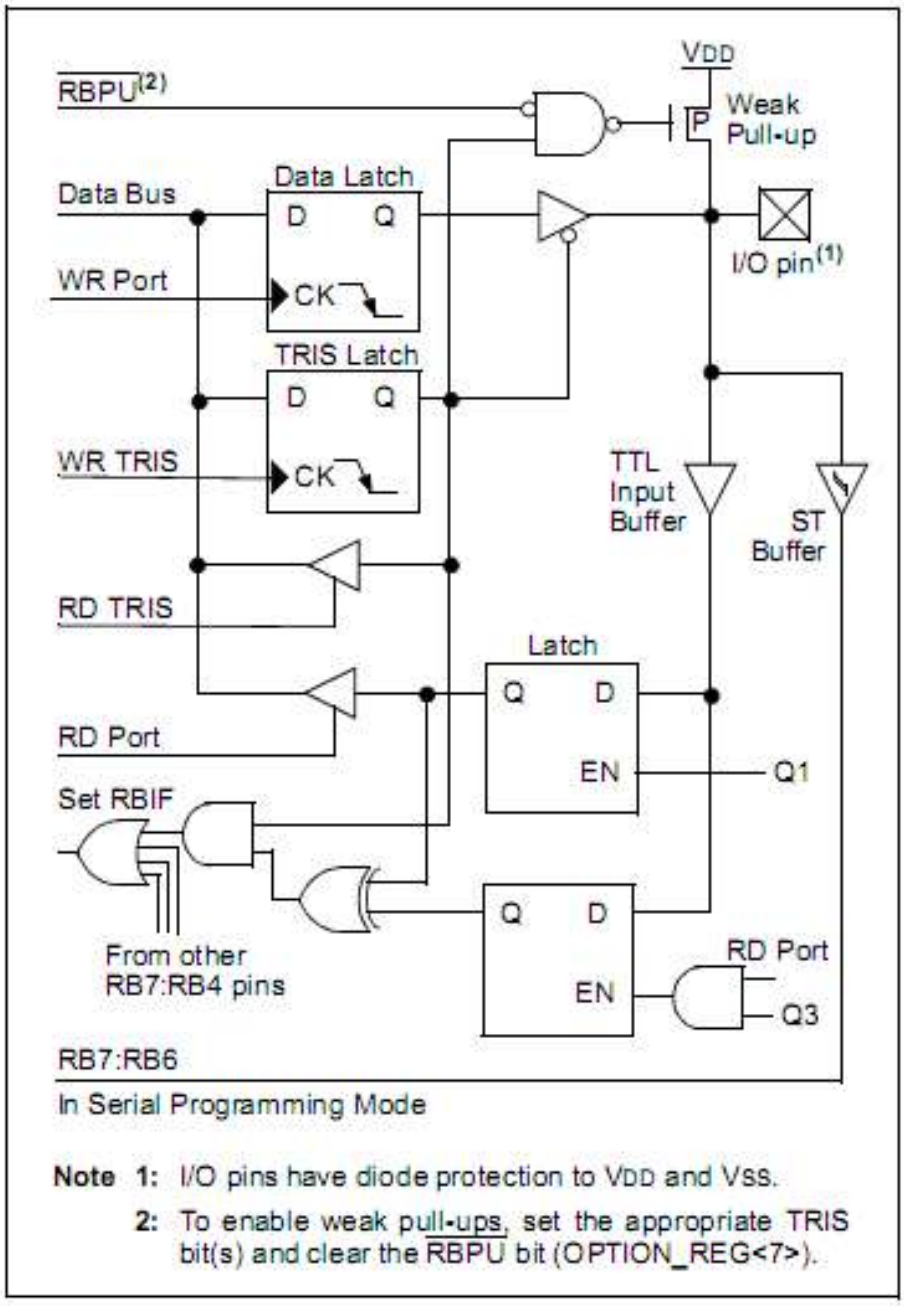
PORTB có độ rộng 8 bit, là port vào ra hai chiều. Ba chân của PORTB được đa hợp với chức năng lập trình mức điện thế thấp (Low Voltage Programming): RB3/PGM, RB6/PGC và RB7/PGD. Mỗi chân của PORTB có một điện trở kéo bên trong. Một bit điều khiển có thể mở tất cả những điện trở kéo này lên. Điều này được thực hiện bằng cách xoá bit $\overline{RBPU}$ (OPTION_REG<7>). Những điện trở này bị cấm khi có một Power-on Reset. Bốn chân của PORTB: RB7 đến RB4 có một ngắt để thay đổi đặc tính .Chỉ những chân được cấu hình như ngõ vào mới có thể gây ra ngắt này. Những chân vào (RB7:RB4) được so sánh với giá trị được chốt trước đó trong lấn đọc cuối cùng của PORTB. Các kết quả không phù hợp ở ngõ ra trên chân RB7:RB4 được OR với nhau để phát ra một ngắt Port thay đổi RB với cờ ngắt là RBIF (INTCON<0>). Ngắt này có thể đánh thức thiết bị từ trạng thái nghỉ (SLEEP). Trong thủ tục phục vụ ngắt người sử dụng có thể xoá ngắt theo cách sau:

a) Đọc hoặc ghi bất kì lên PORTB. Điều này sẽ kết thúc điều kiện không hoà hợp.

b) Xoá bit cờ RBIF.



Hình 1.8.a. Sơ đồ khối các chân RB3:RB0

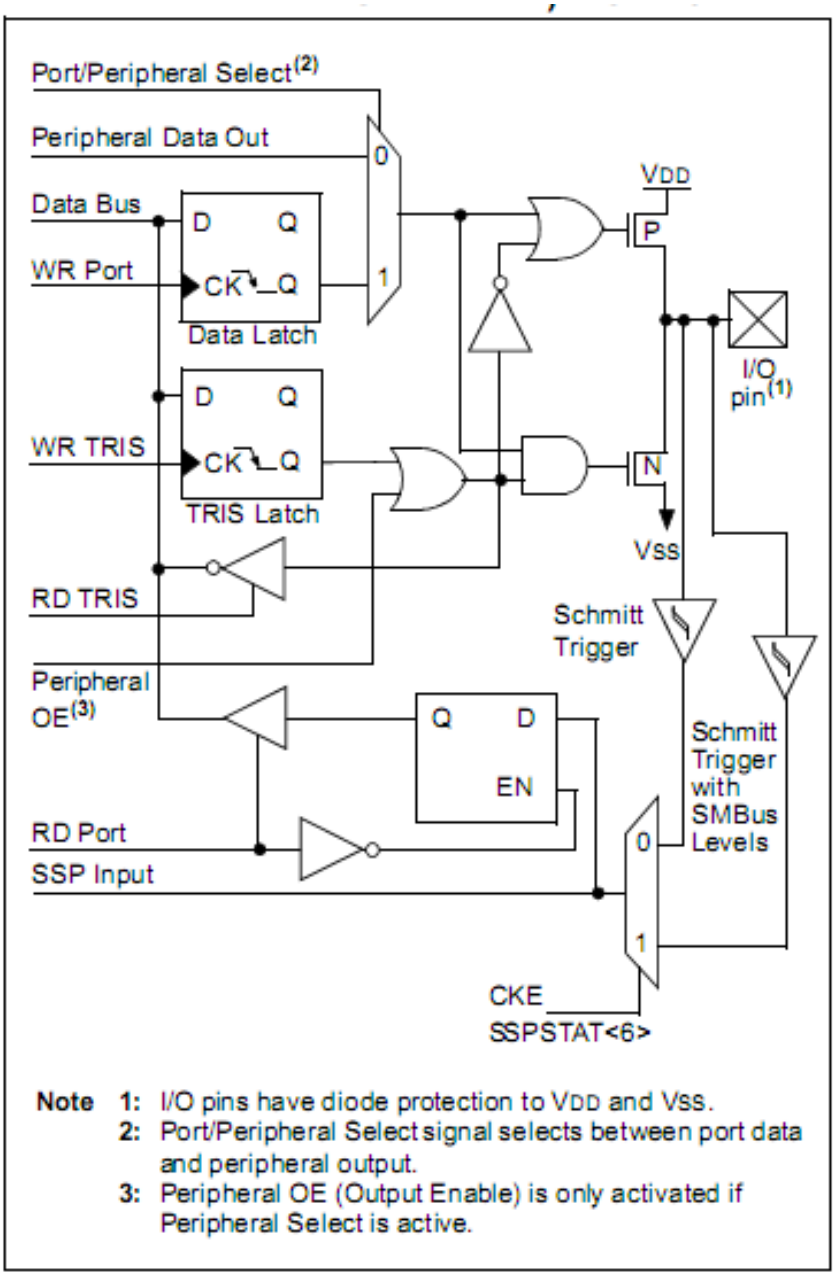


Hình 1.8.b. Sơ đồ khối các chân RB7:RB4

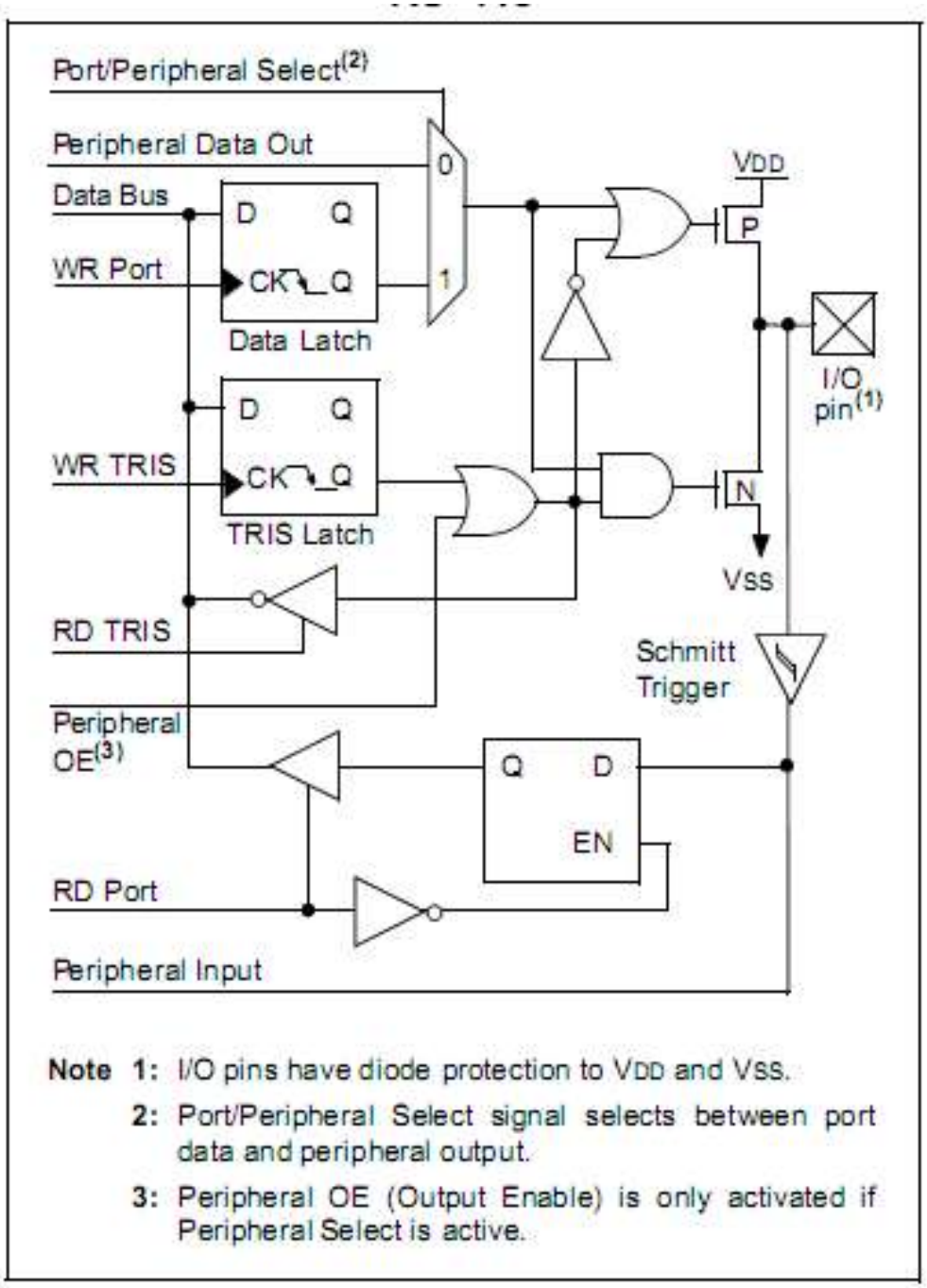
3.3.3 PORT C và thanh ghi TRIS C

PORTC có độ rộng là 8 bit, là port hai chiều. Thanh ghi dữ liệu trực tiếp tương ứng là TRISC. Cho tất cả các bit của TRISC là 1 thì các chân tương ứng ở PORTC là ngõ vào. Cho tất cả các bit của TRISC là 0 thì các chân tương ứng ở PORTC là ngõ ra. PORTC được đa hợp với vài chức năng ngoại vi, những chân của PORTC có đệm Trigger Schmitt ở ngõ vào. Khi bộ I2C được cho phép, chân

3 và 4 của PORTC có thể cấu hình với mức I2C bình thường, hoặc với mức SMBus bằng cách sử dụng bit CKE (SSPSTAT<6>). Khi những chức năng ngoại vi được cho phép, chúng ta cần phải quan tâm đến việc định nghĩa các bit của TRIS cho mỗi chân của PORTC. Một vài thiết bị ngoại vi ghi đè lên bit TRIS thì tạo nên một chân ở ngõ ra, trong khi những thiết bị ngoại vi khác ghi đè lên bit TRIS thì sẽ tạo nên một chân ở ngõ vào. Khi những bit TRIS ghi đè bị tác động trong khi thiết bị ngoại vi được cho phép, những lệnh đọc thay thế ghi (BSF, BCF, XORWF) với TRISC là nơi đến cần phải được tránh. Người sử dụng cần phải chỉ ra vùng ngoại vi tương ứng để đảm bảo cho việc đặt TRIS bit là đúng.



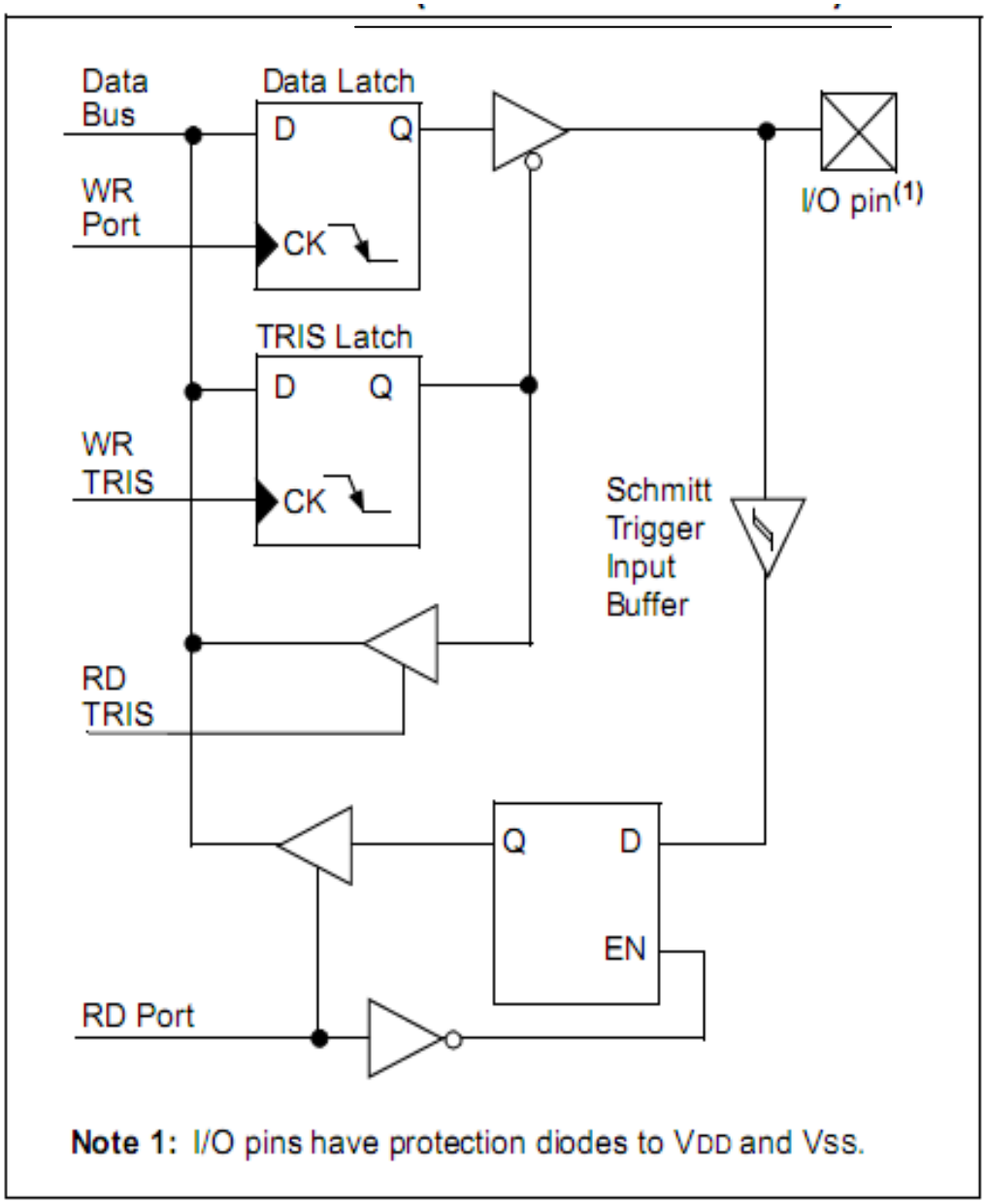
Hình 1.9.a Sơ đồ khối của các chân RC<4:3>



Hình 1.9.b Sơ đồ khối của các chân RC<2:0> và RC<7:5>

3.3.4. PORT D và thanh ghi TRIS D

PORTD là port 8 bit với đệm Trigger Schmitt ở ngõ vào. Mỗi chân có thể được cấu hình riêng lẻ như một ngõ vào hoặc ngõ ra. PORTD có thể được cấu hình như port của bộ vi xử lý rộng 8 bit (parallel slave port) bằng cách đặt bit điều khiển PSPMIDE (TRISE <4>). Trong chế độ này, đệm ở ngõ vào là TTL.



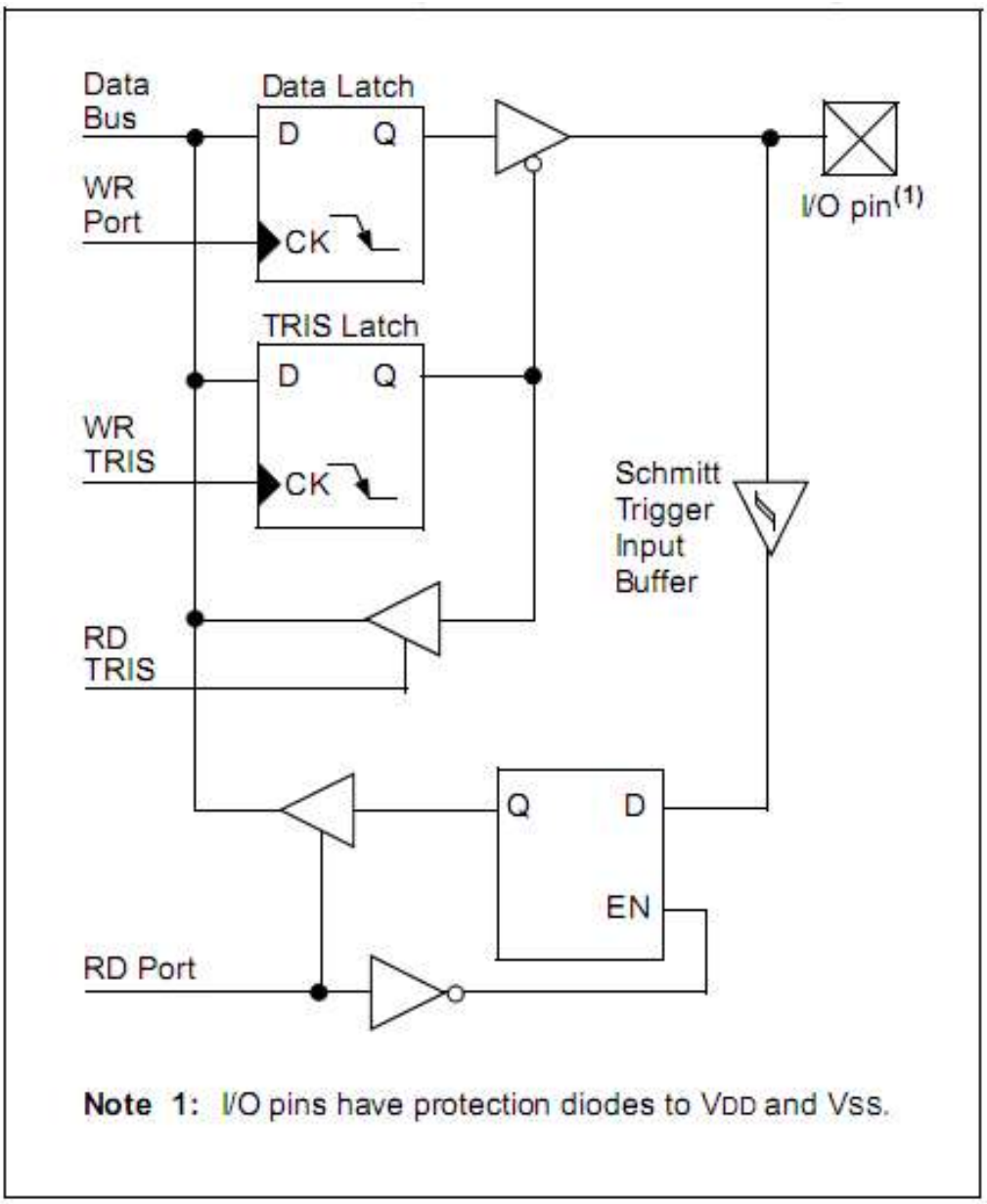
Hình 1.10. Sơ đồ khối của PORTD (trong chế độ là port I/O)

3.3.5 PORT E và thanh ghi TRIS E

PORTE có ba chân (RE0/RD/AN5, RE1/WR/AN6, và RE2/CS/AN7) mỗi chân được cấu hình riêng lẻ như những ngõ vào hoặc những ngõ ra. Những chân này có đệm Trigger Schmitt ở ngõ vào. Những chân của PORT E đóng vai trò như những ngõ vào điều khiển vào ra cho Port của vi xử lý khi bit PSPMODE (TRISE <4>) được đặt. Trong chế độ này, người sử dụng cần phải chắc chắn rằng những bit TRISE <2:0> được đặt, và chắc rằng những chân này được cấu

hình như những ngõ vào số. Cũng bảo đảm rằng ADCON1 được cấu hình cho vào ra số. Trong chế độ này, những đệm ở ngõ vào là TTL.

Những chân của PORTE được đa hợp với những ngõ vào tương tự. Khi được chọn cho ngõ vào tương tự, những chân này sẽ đọc giá trị "0". TRIS E điều khiển hướng của những chân RE chỉ khi những chân này được sử dụng như những ngõ vào tương tự. Người sử dụng cần phải giữ những chân được cấu hình như những ngõ vào khi sử dụng chúng như những ngõ vào tương tự.



Hình 1.11. Sơ đồ khối của PORTE (trong chế độ I/O port)

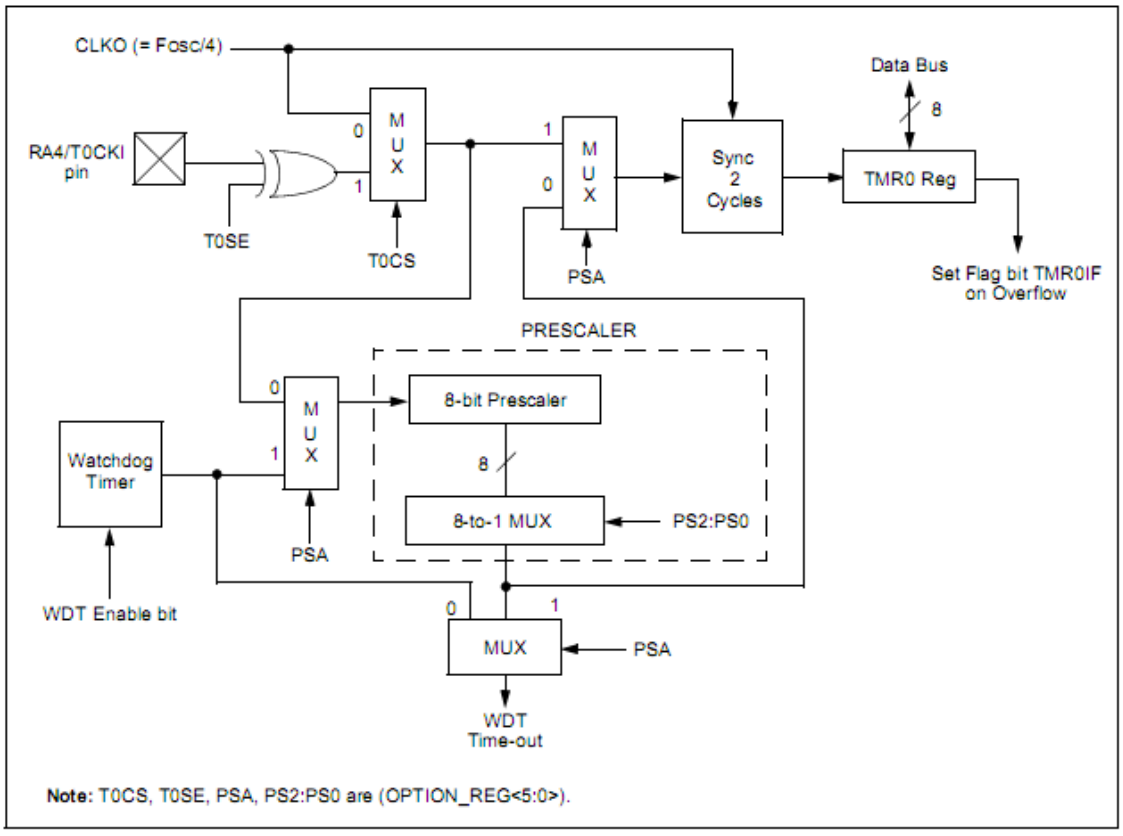
3.4 Hoạt động của định thời

3.4.1 Bộ định thời TIMER 0

Bộ định thời/bộ đếm Timer 0 có các đặc tính sau:

- Bộ định thời / bộ đếm 8 bit
- Cho phép đọc và ghi
- Bộ chia 8 bit lập trình được bằng phần mềm
- Chọn xung clock nội hoặc ngoại
- Ngắt khi có sự tràn từ FFh đến 00h
- Chọn sườn cho xung clock ngoài

Sơ đồ khối của bộ định thời Timer0 và bộ chia dùng chung với WDT được đưa ra trong hình 1.14.



Hình 1.12. Sơ đồ khối của bộ định thời Timer 0 và bộ chia dùng chung với WDT

Chế độ định thời (Timer) được chọn bằng cách xoá bit T0CS (OPTION_REG<5>). Trong chế độ định thời, bộ định thời Timer0 sẽ tăng dần sau mỗi chu kì lệnh (không có bộ chia). Nếu thanh ghi TmR0 được ghi thì sự tăng sẽ bị ngăn lại sau hai chu kì lệnh.

Chế độ đếm (Counter) được chọn bằng cách xoá bit T0CS (OPTION_REG<5>). Trong chế độ đếm, Timer0 sẽ tăng dần ở mỗi cạnh lên xuống của chân RA4/T0CKI. Sự tăng sườn được xác định bởi bit Timer0 Source Edge Select, T0SE (OPTION_RE<4>). Bộ chia chỉ được dùng chung qua lại giữa bộ định thời Timer0 và bộ định thời Watchdog. Bộ chia không cho phép đọc hoặc ghi.

Ngắt Timer0

Ngắt TMR0 được phát ra khi thanh ghi TMR0 tràn từ FFh đến 00h. Sự tràn này sẽ đặt bit T0IF (INTCON<2>). Ngắt này có thể được giấu đi bằng cách xóa bit T0IE (INTCON<5>) . Bit T0IF cần phải được xóa trong chương trình bởi thủ tục phục vụ ngắt của bộ định thời Timer0 trước khi ngắt này được cho phép lại.

Sử dụng Timer0 với xung clock ngoại

Khi bộ chia không được sử dụng, clock ngoài đặt vào thì giống như bộ chia ở ngõ ra. Sự đồng bộ của chân T0CKI với clock ngoài được thực hiện bằng cách lấy mẫu bộ chia ở ngõ ra trên chân Q2 và Q4. Vì vậy thực sự cần thiết để chân T0CKI ở mức cao trong ít nhất 2 chu kỳ máy và ở mức thấp trong ít nhất 2 chu kỳ máy.

Bộ chia

Thiết bị PIC16F87X chỉ có một bộ chia mà được dùng chung bởi bộ định thời TIMER0 và bộ định thời Watchdog. Bộ chia có các Hệ số chia dùng cho Timer0 hoặc bộ WDT. Các hệ số này không có khả năng đọc và khả năng viết. Để chọn hệ số chia xung vào Timer0 hoặc cho bộ WDT ta tiến hành xoá hoặc đặt bit PSA của thanh ghi OPTION_REG<3>.

Những bit PS2, PS1, PS0 của thanh ghi OPTION_REG<2:0> dùng để xác lập các hệ số chia.

3.4.2. Bộ định thời TIMER1

Bộ định thời TIMER1 là một bộ định thời/bộ đếm 16 bit gồm hai thanh ghi TMR1H (Byte cao) và TMR1L (byte thấp) mà có thể đọc hoặc ghi. Cặp thanh ghi này tăng số đếm từ 0000h đến FFFFh và báo tràn sẽ xuất hiện khi có sự chuyển số đếm từ FFFFh xuống 0000h. Ngắt, nếu được phép có thể phát ra khi có số đếm tràn và được đặt ở bit cờ ngắt TMR1IF. Ngắt có thể được phép hoặc cấm bằng cách đặt hoặc xoá bit cho phép ngắt TMR1IE.

Bộ định thời Timer1 có thể được cấu hình để hoạt động một trong hai chế độ sau:

- Định thời một khoảng thời gian (timer)
- Đếm sự kiện (Counter)

Việc lựa chọn một trong hai chế độ được xác định bằng cách đặt hoặc xoá bit điều khiển TMR1ON.

----	----	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

Bit7

Bit0

Bit 7-6 Không được định nghĩa

Bit 5-4 bit chọn bộ chia clock cho timer1

Bit 3 bit điều khiển cho phép bộ dao động Timer1

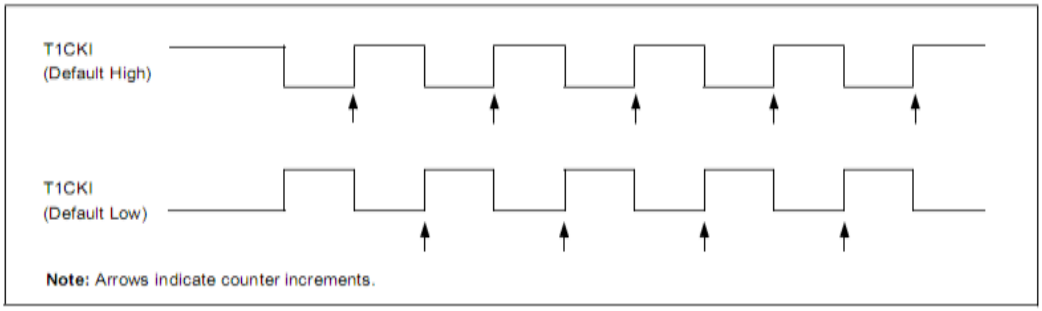
Bit 2 bit điều khiển clock ngoài Timer

Bit 1 bit chọn nguồn clock cho Timer1

Bit 0 bit điều khiển hoạt động của Timer1

Chế độ Timer

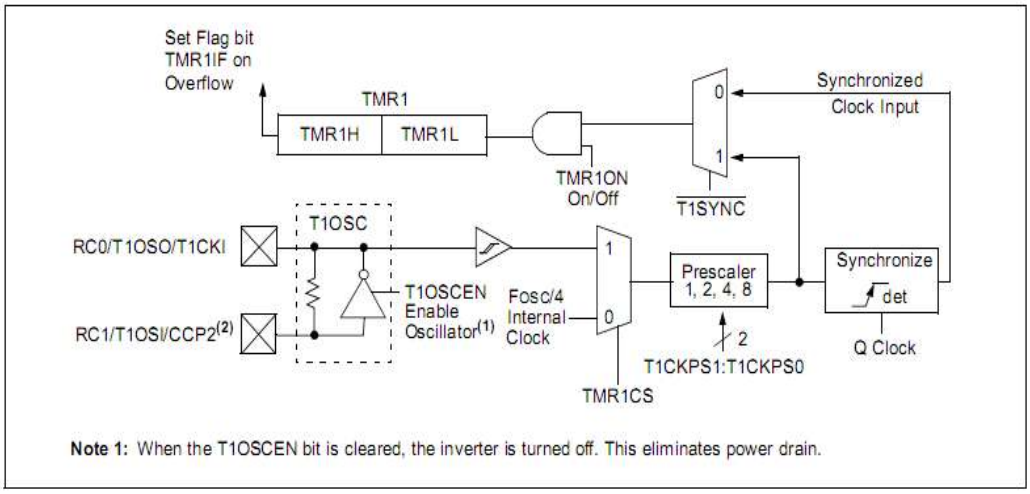
Chế độ Timer được chọn bằng cách xoá TMR1CS. Trong chế độ này, Nguồn clock đặt vào Timer là mạch dao động $F_{OSC}/4$. Bit điều khiển đồng bộ không bị tác động vì clock ngoài luôn luôn đồng bộ.



Hình 1.12.a. Cạnh tăng timer1

Chế độ counter

Trong chế độ này, bộ định thời tăng số đếm qua clock ngoài. Việc tăng xảy ra sau mỗi sườn lên của xung clock ngoài. Bộ định thời phải có một sườn lên trước khi việc đếm bắt đầu.



Hình 1.12.b Sơ đồ khối bộ định thời timer1

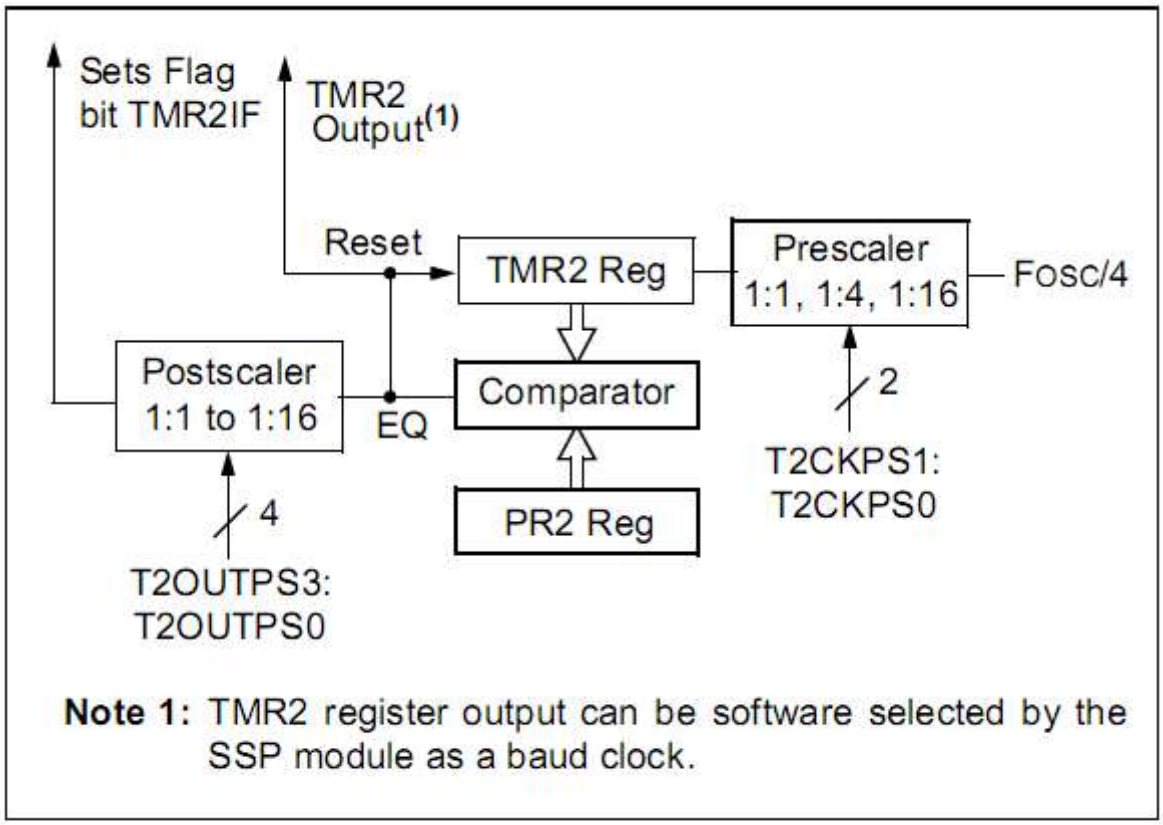
3.4.3. Bộ định thời TIMER2

Bộ định thời TIMER2 là bộ định thời 8 bit với một bộ đếm và một bộ potscaler. Nó thường dùng chung với bộ CCP trong chế độ PWM (sẽ được đề

cập ở phần sau). Thanh ghi TMR2 có thể đọc hoặc ghi và được xoá khi có bất kì tín hiệu reset nào của thiết bị.

Bộ định thời TIMER2 có một thanh ghi chu kỳ 8 bit, PR2. Bộ định thời tăng số đếm lên từ 00h đến giá trị được ghi trong thanh ghi TR2 và sau đó reset lại giá trị 00h trong chu kỳ kế tiếp. PR2 là thanh ghi có thể đọc hoặc ghi.

Giá trị trùng hợp trong thanh ghi TMR2 được đi qua bộ postscaler 4 bit để phát ra một ngắt TMR2 (được đặt ở bit cờ ngắt TMR2IF). Bộ định thời TIMER2 có thể được tắt (không hoạt động) bằng cách xoá bít điều khiển TMR2ON để giảm thiểu công suất tiêu tán nguồn.



Hình 1.13.a Sơ đồ khối của 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 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 postscale
0001 = 1:2 postscale
0010 = 1:3 postscale
•
•
•
1111 = 1:16 postscale

bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Hình 1.13.b. T2CON: Thanh ghi điều khiển Timer2 (địa chỉ 12h)

Một đặc điểm khác của vi điều khiển Pic16F877A là có bộ dao động chủ trên chip điều, nó sẽ giúp tránh được những sai số không cần thiết trong việc tạo xung dao động, vi điều khiển Pic16F877A có khả năng tự Reset bằng bộ WDT, và có thêm 256 byte EEPROM.

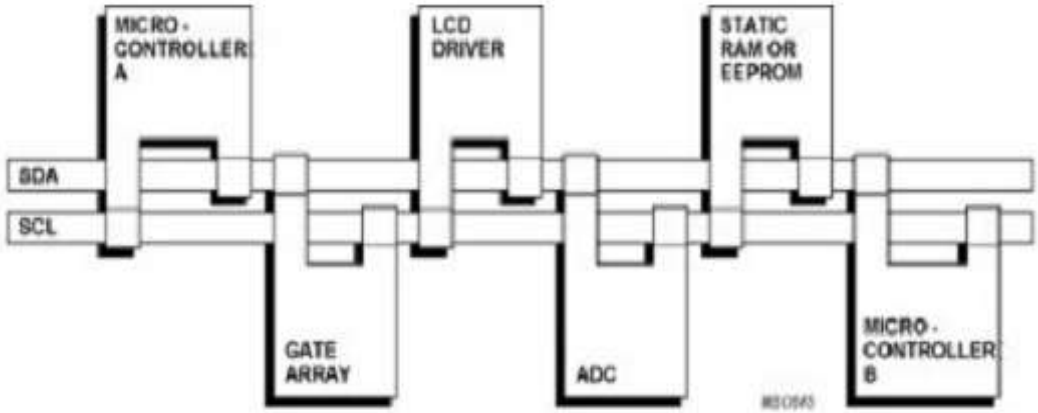
4. Giao tiếp I2C

4.1.Giới thiệu chung về I2C

Ngày nay trong các hệ thống thông tin điện tử hiện đại, rất nhiều ICs hay thiết bị ngoại vi cần phải giao tiếp với các ICs hay thiết bị ngoại vi khác - giao tiếp với thế giới bên ngoài. Với mục tiêu đạt được hiệu quả cho phần cứng tốt nhất với mạch điện đơn giản, Philips đã phát triển một chuẩn giao tiếp nối hai dây được gọi là I2C. I2C là tên viết tắt của cụm từ Inter Intergrated Circuit - bus giao tiếp giữa các IC với nhau.

I2C mặc dù được phát triển bởi Philips nhưng nó được rất nhiều nhà sản xuất trên thế giới sử dụng. I2C trở thành một chuẩn công nghiệp cho các giao tiếp điều khiển. Có thể kể ra một vài tên tuổi ngoài Philips như Texas Intrusment (TI), Maxim dallas, Analog device, National Semiconductor… Bus I2C được

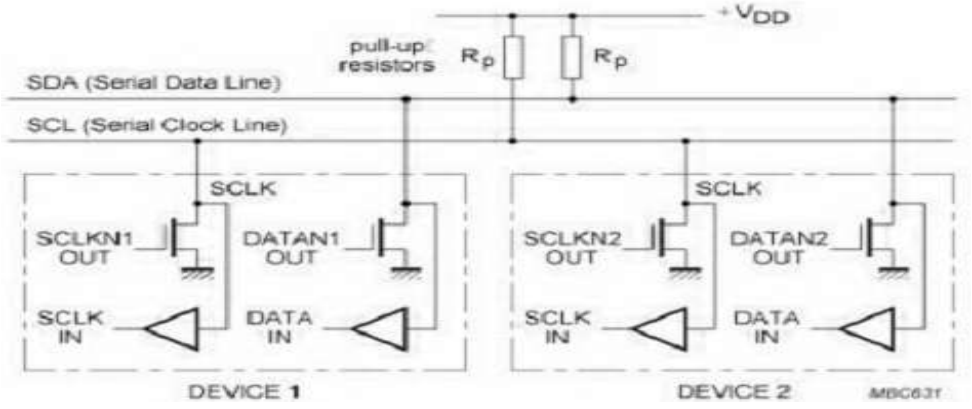
dùng làm bus giao tiếp ngoại vi cho rất nhiều loại IC khác nhau như các loại vi điều khiển 8051, PIC, AVR, ARM, chip nhớ như Ram tĩnh (Static ram), EEPROM, bộ chuyển đổi tương tự số (ADC), số tương tự (DAC), IC điều khiển LCD, LED…



Hình 1.14 Bus I2C và thiết bị ngoại vi

4.1.1 Đặc điểm giao tiếp I2C

Một giao tiếp I2C gồm có hai dây: serial data (SDA) và serial clock (SCL). SDA là đường truyền dữ liệu theo hai hướng, còn SCL là đường truyền xung đồng hồ và chỉ theo một hướng. Khi một thiết bị ngoại vi kết nối vào đường I2C thì chân SDA của nó sẽ được nối với dây SDA của bus, chân SCL được nối với dây SCl.



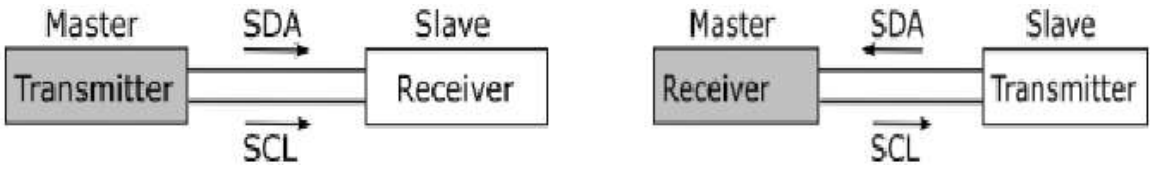
Hình 1.15.a Kết nối thiết bị vào bus I2C ở chế độ chuẩn (Standard mode) và chế độ nhanh (Fast mode)

Mỗi dây SDA hay SCL đều được nối với điện áp dương của nguồn cấp thông qua một điện trở kéo lên (full-up resistor). Sự cần thiết của các điện trở kéo

này là vì chân giao tiếp I2C của các thiết bị ngoại vi thường là dạng cực máng hở (open-drain or open-collector). Giá trị các điện trở này khác nhau tùy vào từng thiết bị và chuẩn giao tiếp, thường dao động trong khoảng từ 1kΩ đến 4,7kΩ.

Trở lại với hình 1.15.a ta thấy có rất nhiều thiết bị (ICs) cùng được kết nối vào một bus I2C, tuy nhiên sẽ không xảy ra chuyện nhầm lẫn giữa các thiết bị, bởi mỗi thiết bị sẽ được nhận ra bởi một địa chỉ duy nhất có mối quan hệ chủ/tớ tồn tại trong suốt thời gian kết nối. Mỗi thiết bị có thể hoạt động như là thiết bị nhận dữ liệu hay có thể vừa truyền vừa nhận. Hoạt động truyền hay nhận con phụ thuộc vào thiết bị đó là chủ (mater) hay tớ (slave).

Một thiết bị hay một IC khi kết nối với bus I2C, ngoài một địa chỉ (duy nhất) để phân biệt, nó còn được cấu hình là thiết bị chủ (mater) hay tớ (slave). Có sự phân biệt đó bởi trên một bus I2C thì quyền điều khiển thuộc về thiết bị chủ (mater). Thiết bị chủ đóng vai trò tạo xung đồng hồ (clock) cho toàn hệ thống. Khi giữa hai thiết bị chủ/tớ giao tiếp thì thiết bị chủ có vai trò tạo xung đồng hồ và quản lý địa chỉ của thiết bị tớ trong suốt quá trình giao tiếp. Thiết bị chủ đóng vai trò chủ động, còn thiết bị tớ đóng vai trò bị động trong việc giao tiếp.



Hình 1.15.b Truyền nhận dữ liệu giữa chủ/tớ

Nhìn hình trên ta thấy xung đồng hồ chỉ có một hướng từ chủ đến tớ còn luồng dữ liệu có thể đi theo hai hướng từ chủ đến tớ hay ngược lại từ tớ đến chủ.

Với dữ liệu truyền trên bus I2C, một bus I2C chuẩn truyền 8 - bít dữ liệu có hướng trên đường truyền với tốc độ 100 kbit/s - chế độ chuẩn (standard-mode). Tốc độ truyền có thể lên tới 400 kbit/s - chế độ nhanh (Fast-mode) và cao nhất 3,4 Mbit/s - chế độ cao tốc (high speed mode).

Một bus I2C có thể hoạt động ở nhiều chế độ khác nhau:

- Một chủ - một tớ (one master – one slave).

- Một chủ - nhiều tớ (one master – mutil slave).
- Nhiều chủ - nhiều tớ (mutil master – mutil slave).

Dù ở chế độ nào, một giao tiếp I2C đều dựa vào quan hệ chủ/tớ . Giả thiết 1 thiết bị A muốn gửi dữ liệu đến thiết bị B, quá trình được thực hiện như sau:

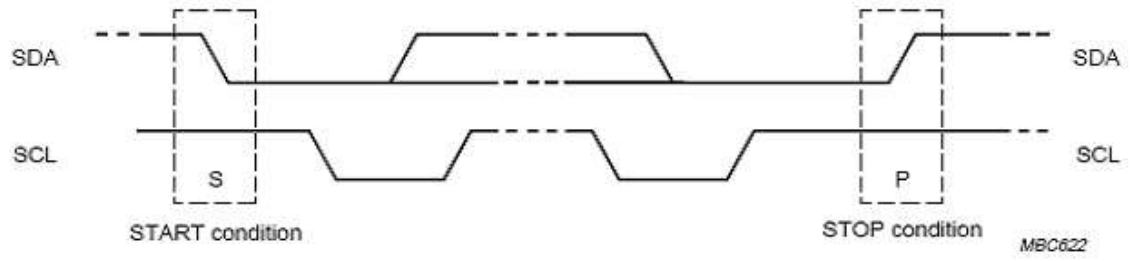
- Thiết bị A (chủ) xác định đúng địa chỉ của thiết bị B (tớ) , cùng với việc xác định địa chỉ, thiết bị A sẽ quyết định việc đọc hay ghi vào thiết bị tớ.
- Thiết bị A gửi dữ liệu tới thiết bị B.
- Thiết bị A kết thúc quá trình truyền dữ liệu.

Khi A muốn nhận dữ liệu từ B, quá trình diễn ra như trên chỉ khác là A sẽ nhận dữ liệu từ B. Trong giao tiếp này A là chủ còn B vẫn là tớ. Chi tiết việc thiết lập một giao tiếp giữa hai thiết bị sẽ được mô tả chi tiết đầy đủ trong các mục dưới đây.

4.1.1.1 START and STOP conditions (điều kiện)

START and STOP là những điều kiện bắt buộc phải có khi một thiết bị chủ muốn thiết lập giao tiếp với một thiết bị nào đó trong giao tiếp I2C. START là điều kiện khởi đầu báo hiệu bắt đầu giao tiếp, còn STOP báo hiệu kết thúc một giao tiếp. Hình dưới đây mô tả điều kiện START và STOP

Ban đầu khi chưa thực hiện quá trình giao tiếp,cả hai đường SDA và SCL đều ở mức cao (SDA=SCL=HIGH).Lúc này bus I2C được coi là dỗi (bus free),sẵn sàng cho một giao tiếp.Hai điều kiện START,STOP không thể thiếu trong việc giao tiếp giữa các thiết bị I2C với nhau.



Hình 1.16. Điều kiện Start và Stop.

Điều kiện START : một sự chuyển đổi trạng thái từ cao xuống thấp trên đường SDA trong khi đường SCl đang ở mức cao (cao = 1,thấp = 0) báo

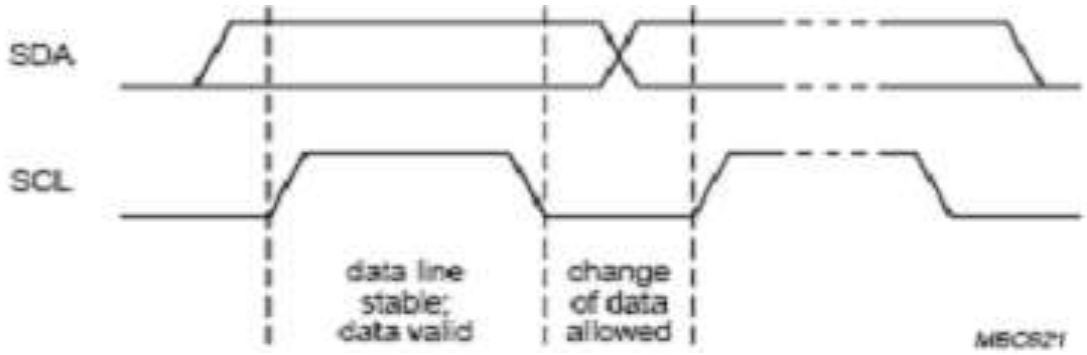
Điều kiện STOP : một sự chuyển đổi trạng thái từ mức thấp nên cao trên đường SDA trong khi đường SCL ở mức cao.

Cả hai điều kiện START và STOP đều được tạo ra bởi thiết bị chủ. Sau tín hiệu START, bus I2C coi như đang trong trạng thái làm việc (busy). Bus I2C sẽ rỗi, sẵn sàng cho một giao tiếp mới sau tín hiệu STOP từ thiết bị chủ.

Sau khi có một điều kiện START, trong quá trình giao tiếp, khi có một tín hiệu START được lặp lại thay vì có một tín hiệu STOP thì bus I2C vẫn tiếp tục ở trạnh thái bận. Tín hiệu START và lặp lại START đều có chức năng giống nhau là khởi tạo một giao tiếp.

4.1.1.2 Định dạng dữ liệu truyền

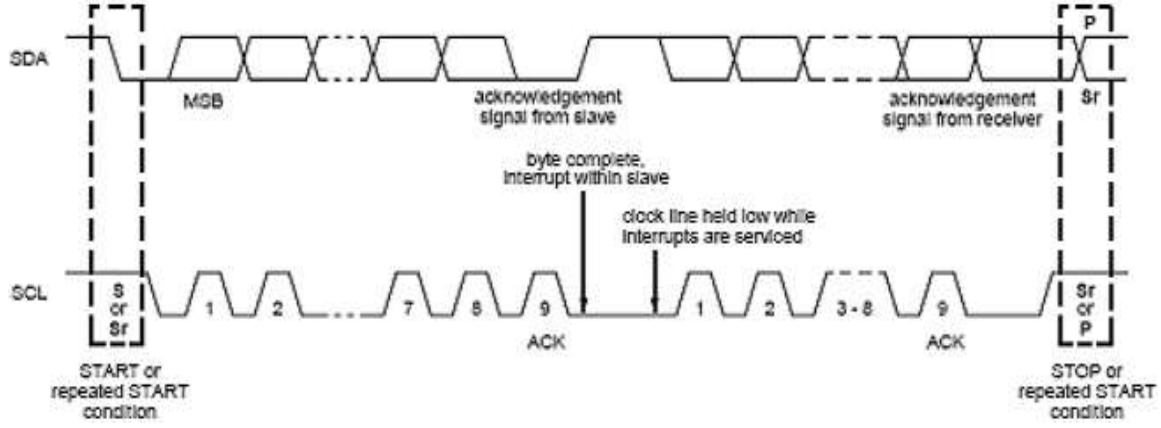
Dữ liệu được truyền trên bus I2C theo từng bit, bit dữ liệu được truyền đi tại mỗi sườn dương của xung đồng hồ trên dây SCL, quá trình thay đổi bit dữ liệu xảy ra khi SCl đang ở mức thấp.



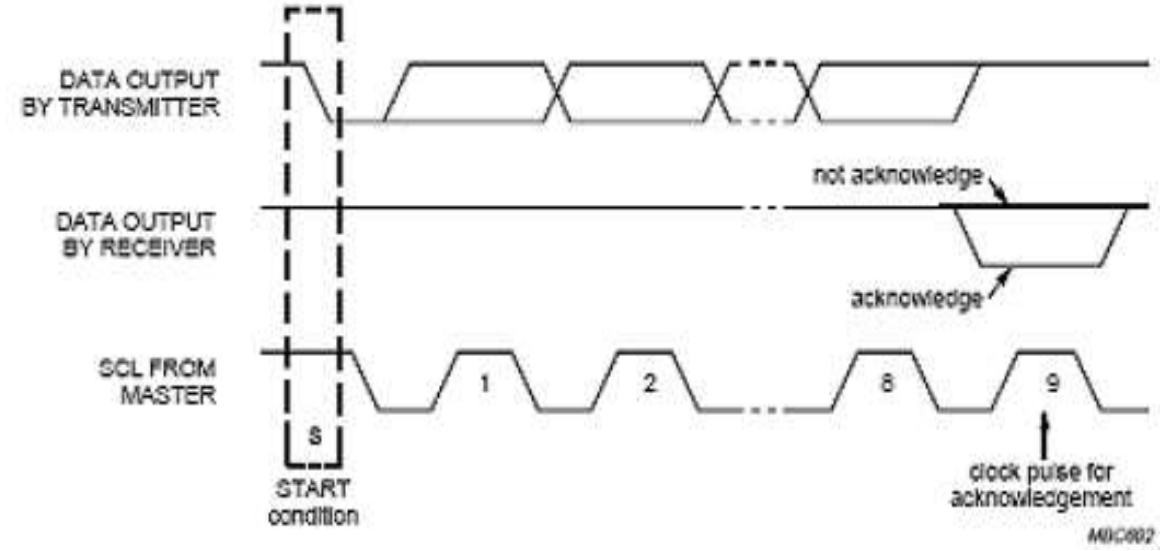
Hình 1.17.a. Quá trình truyền 1 bit dữ liệu

Mỗi byte dữ liệu được truyền có độ dài là 8 bits. Số lượng byte được truyền trong một lần là không hạn chế. Mỗi byte được truyền đi theo sau là một bit ACK để báo hiệu đã nhận dữ liệu. Bit có trọng số cao nhất (MSB) sẽ được truyền đi đầu tiên, các bit sẽ được truyền đi lần lượt. Sau 8 xung clock trên dây SCl, 8 bit dữ liệu đã được truyền đi. Lúc này thiết bị nhận sau khi đã nhận đủ 8 bit dữ liệu sẽ kéo SDA xuống mức thấp tạo một xung ACK ứng với xung clock

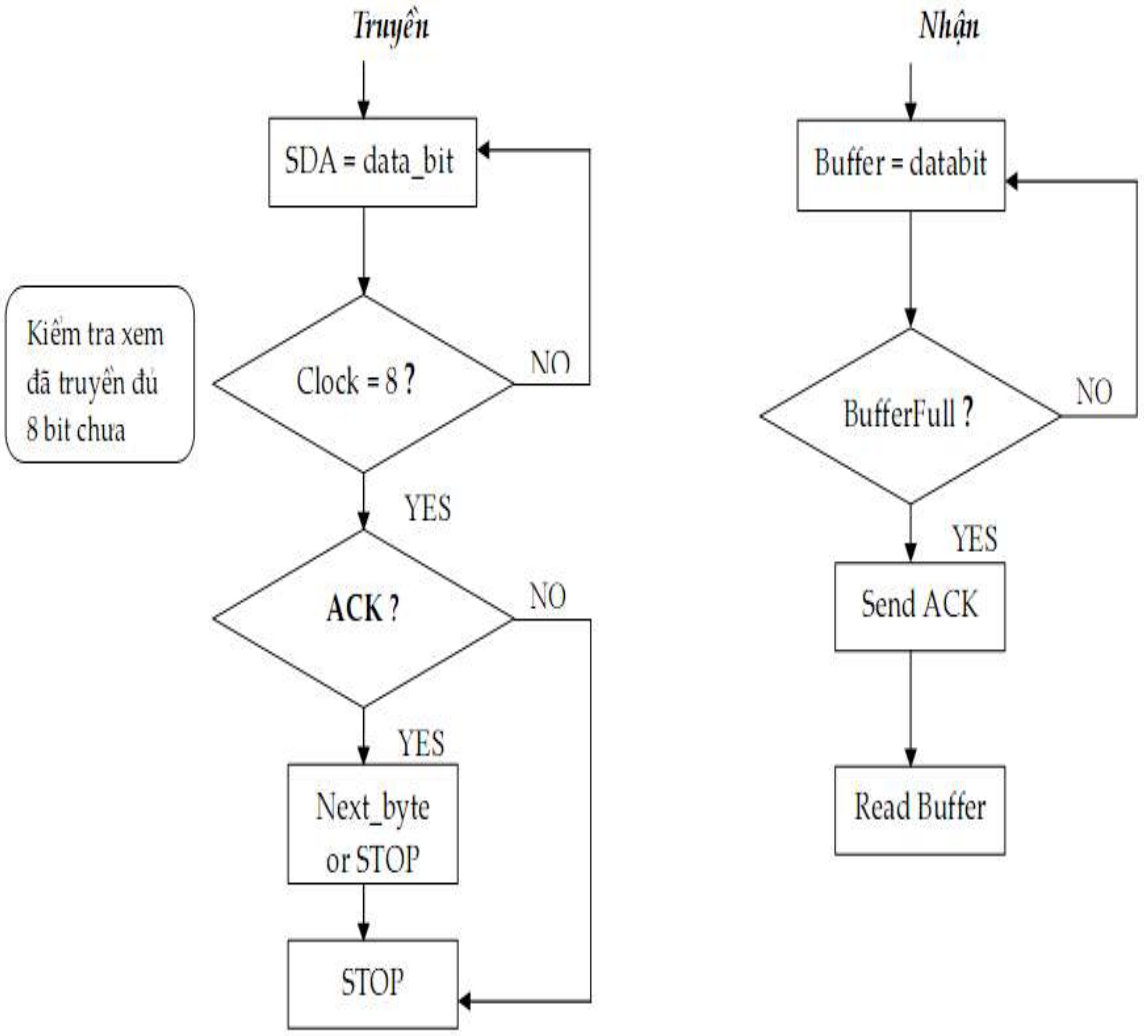
thứ 9 trên dây SDA để báo hiệu đã nhận đủ 8 bit. Thiết bị truyền khi nhận được bit ACK sẽ tiếp tục thực hiện quá trình truyền hoặc kết thúc.



Hình 1.18.b. Dữ liệu truyền trên bus I2C



Hình 1.18.c. Bit ACK trên bus I2C



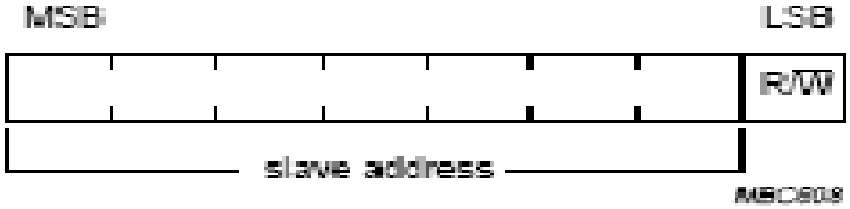
Hình 1.18.d. Lưu đồ thuật toán quá trình truyền nhận dữ liệu

Một byte truyền đi có kèm theo bit ACK là điều kiện bắt buộc, nhằm đảm bảo quá trình truyền nhận diễn ra chính xác. Khi không nhận được đúng địa chỉ hay khi muốn kết thúc quá trình giao tiếp, thiết bị nhận sẽ gửi một xung Not-ACK (SDA ở mức cao) để báo cho thiết bị chủ biết, thiết bị chủ sẽ tạo xung STOP để kết thúc hay lặp lại một xung START để bắt đầu quá trình mới.

4.1.1.3 Định dạng địa chỉ thiết bị

Mỗi thiết bị ngoại vi tham gia vào bus I2C đều có một địa chỉ duy nhất, nhằm phân biệt giữa các thiết bị với nhau. Độ dài địa chỉ là 7-bit, điều đó có nghĩa là trên một bus I2C ta có thể phân biệt được tối đa 128 thiết bị. Khi thiết bị chủ muốn giao tiếp với thiết bị ngoại vi nào trên bus I2C, nó sẽ gửi 7 bit địa chỉ

của thiết bị đó ra bus ngay sau xung START. Bye đầu tiên được gửi sẽ bao gồm 7 bit địa chỉ và một bit thứ 8 để điều khiển hướng truyền.

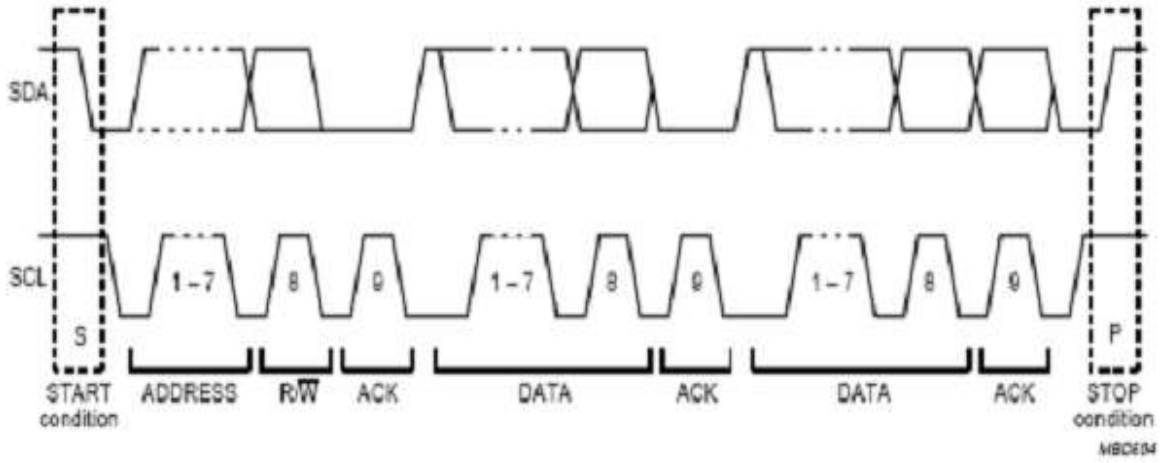


Hình 1.19. Cấu trúc byte dữ liệu đầu tiên

Mỗi thiết bị ngoại vi sẽ có một địa chỉ riêng do nhà sản xuất ra nó quy định. Địa chỉ đó có thể là cố định hay thay đổi. Riêng bit điều khiển hướng sẽ quy định chiều truyền dữ liệu. Nếu bit này bằng '0' có nghĩa là byte dữ liệu tiếp theo sau sẽ được truyền từ chủ đến tớ, còn ngược lại nếu bằng '1' thì các byte theo sau byte đầu tiên sẽ là dữ liệu từ con tớ gửi đến con chủ. Việc thiết lập giá trị cho bit này do con chủ thi hành, con tớ sẽ tuỳ theo giá trị đó mà có sự phản hồi tương ứng đến con chủ.

4.1.1.4 Truyền dữ liệu trên bus I2C, chế độ Master - Slave

Việc truyền dữ liệu diễn ra giữa con chủ và con tớ. Dữ liệu truyền có thể theo hai hướng từ chủ đến tớ hay ngược lại. Hướng truyền được quy định bởi bit thứ 8 $R/\overline{W}$ trong byte đầu tiên được truyền đi.



Hình .1.20.a Quá trình truyền dữ liệu

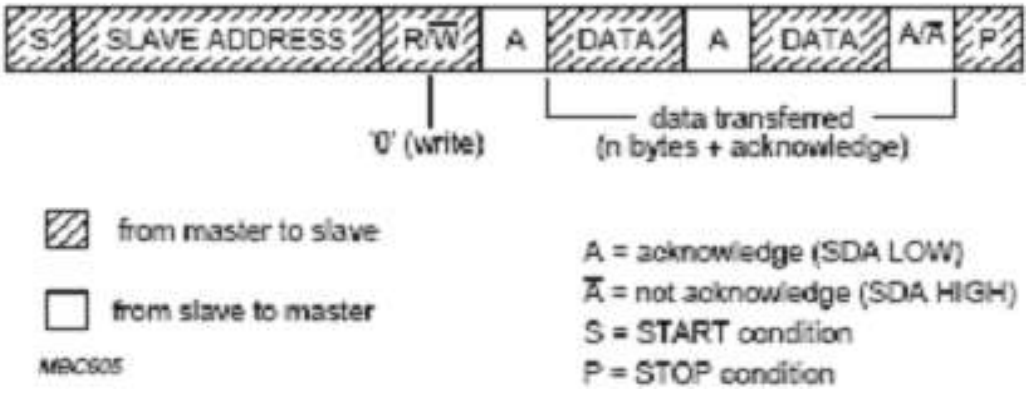
+ Truyền dữ liệu từ chủ đến tớ (ghi dữ liệu) : thiết bị chủ khi muốn ghi dữ liệu đến con tớ, quá trình thực hiện là:

- Thiết bị chủ tạo xung START.

- Thiết bị chủ gửi địa chỉ cho thiết bị tớ mà nó cần giao tiếp cùng với bit $R/\overline{W}$=0 ra bus và đợi xung ACK phản hồi từ con tớ.

- Khi nhận được xung ACK báo đã nhận diện đúng thiết bị tớ, con chủ bắt đầu gửi dữ liệu đến con tớ theo từng byte một. Theo sau mỗi byte này đều là một xung ACK. Số lượng byte truyền là không hạn chế.

- Kết thúc quá trình truyền, con chủ sau khi truyền byte cuối tạo xung STOP báo hiệu kết thúc.



Hình 1.20.b Ghi dữ liệu từ chủ đến tớ

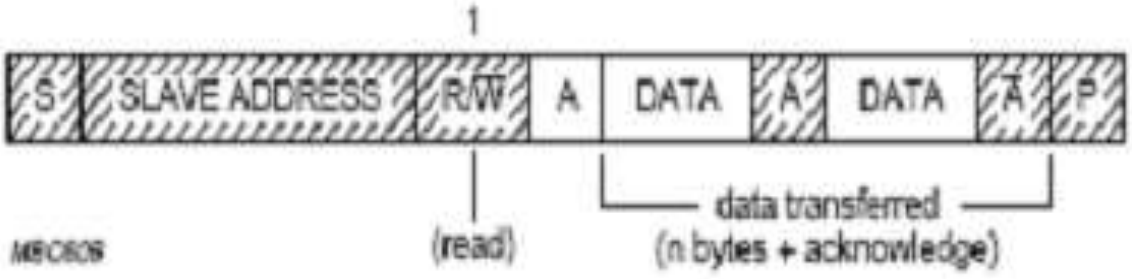
+ Truyền dữ liệu từ tớ đến chủ (đọc dữ liệu): thiết bị chủ muốn đọc dữ liệu từ thiết bị tớ, quá trình được thực hiện như sau:

- Khi bus rỗi, thiết bị chủ tạo xung START, báo hiệu bắt đầu giao tiếp.

- Thiết bị chủ gửi địa chỉ thiết bị tớ cần giao tiếp cùng với bit $R/\overline{W}$=1 và đợi xung ACK từ phía thiết bị tớ.

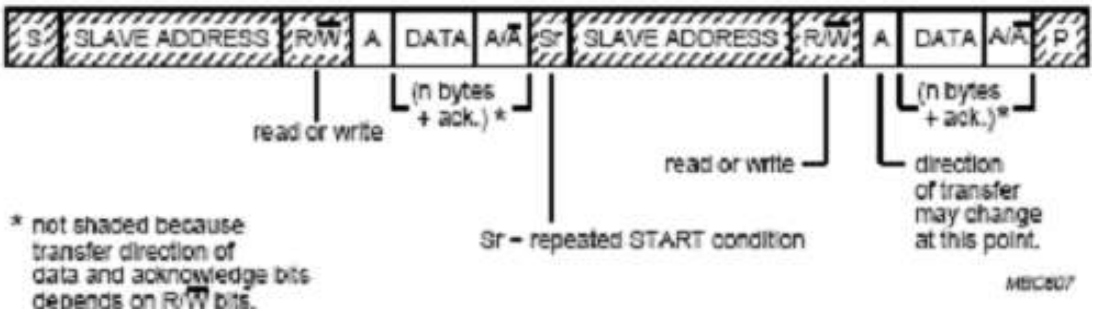
- Sau xung ACK đầu tiên, thiết bị tớ sẽ gửi từng byte ra bus, thiết bị chủ sẽ nhận dữ liệu và trả về xung ACK. Số lượng byte không hạn chế.

- Khi muốn kết thúc quá trình giao tiếp thiết bị chủ gửi xung Not-ACK và tạo cung STOP để kết thúc.



Hình 1.20.c Đọc dữ liệu từ thiết bị tớ

+ Quá trình kết hợp ghi và đọc dữ liệu: giữa hai xung START và STOP, thiết bị chủ có thể thực hiện việc đọc hay ghi nhiều lần với một hay nhiều thiết bị. Để thực hiện việc đó, sau quá trình ghi hay đọc, thiết bị chủ lặp lại một xung START và lại gửi lại địa chỉ của thiết bị tớ và bắt đầu một quá trình mới.



Hình 1.20.d Quá trình phối hợp đọc/ghi dữ liệu

Chế độ giao tiếp Master - Slave là chế độ cơ bản trong một bus I2C, toàn bộ bus được quản lý bởi một Master duy nhất. Trong chế độ này không xảy ra tình trạng xung đột bus hay mất đồng bộ xung clock vì chỉ có một master duy nhất có thể tạo xung clock.

4.1.1.5 Chế độ Multi-Master

Trên bus I2C có thể có nhiều hơn 1 Master điều khiển bus. Khi đó bus I2C sẽ hoạt động ở chế độ Multi-Master.

4.2 I2C trong vi điều khiển PIC

4.2.1 Tổng quan chung

Ở Việt Nam PIC 16Fxxxx được sử dụng và đề cập nhiều nhất là 16F84, 16F628, 16F88, 16F87x. Trong đó:

- 16F84, 16F628 không tích hợp chuẩn I2C.

- 16F88 tích hợp I2C nhưng chỉ hỗ trợ chế độ Slave không dùng được ở chế độ Master.

- 16F87x tích hợp I2C ở cả chế độ Master và Slave.

Do đó nếu muốn sử dụng ở chế độ Master với các PIC 16F84, 16F628, 16F88 chúng ta phải gây dựng bằng phần mềm.

4.2.2 Truyền và nhận dữ liệu dùng I2C

4.2.2.1 Quá trình truyền một byte dữ liệu từ Master qua Slave

Nguyên tắc truyền một byte dữ liệu gồm các bước cơ bản sau:

- Gửi bit start từ Master đến Salve . Đợi cho đến khi truyền xong.
- Gửi địa chỉ của Slave lên đường truyền để chọn Slave nào hoạt động , đợi cho đến khi truyền xong.
- Gửi địa chỉ cần lưu dữ liệu tới. Đợi cho đến khi truyền xong.
- Gửi dữ liệu cân truyền tới Slave . Đợi cho đến khi truyền xong.
- Tiếp tục gửi dữ liệu…..
- Khi muốn kết thúc thì gửi bit Stop lên đường truyền.

4.2.2.2 Quá trình nhận dữ liệu từ Slave

Quá trình nhận dữ liệu từ Slave diễn ra theo các bước sau:

- Gửi bit Start từ Master về Slave. Đợi cho đến khi truyền xong.

- Gửi địa chỉ của Slave (bit 0 = 0) lên đường truyền. Dùng để chọn Slave nào hoạt động. Đợi cho đến khi truyền xong.

- Gửi địa chỉ của dữ liệu cần nhận. Đợi cho đến khi truyền xong.

- Gửi bit Restart. Đợi cho đến khi truyền xong.

- Gửi địa chỉ Slave lên đường truyền (bit 0 =1 báo rằng hoạt động sắp tới là đọc). Đợi cho tới khi truyền xong.

- Đọc dữ liệu từ Slave. Đợi cho đến khi truyền xong.

- Phát bit ACK báo tiếp tục nhận dữ liệu . Đợi cho đến khi truyền xong.

- …….

- Đọc dữ liệu từ Slave. Đợi cho đến khi đọc xong.

- Phát bit NACK báo rằng quá trình nhận dữ liệu đã kết thúc . Đợi cho đến khi truyền xong.

Phát bit Stop để kết thúc

4.2.3 Giao tiếp I2C trong vi điều khiển 16F87x

4.2.3.1 Cách sử dụng I2C chế độ Master

Trong PIC 16F87x có 3 thanh ghi điều khiển quá trình truyền và nhận dữ liệu đó là SSPSTAT (94h bank 1), SSPCON1 (14h bank 0), SSPCON2 (91h bank1) . Trong đó thì:

- SSPSTAT:

SSPSTAT: MSSP STATUS REGISTER (I2C mode) (ADDRESS 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

SMP : Chọn Speed chuẩn (=1:100 Khz,1 Mhz, =0 :400Khz)

CKE

R/W : Báo rằng quá trình truyền vẫn đang diễn ra

BF : Báo rằng SSPBUF vẫn đang đầy (trong cả hai trường hợp transmit, receive).

- SSPCON1:

SSPCON1 : MSSP CONTROL REGISTER 1 (I2C mode) (ADDRESS 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

WCOL : báo rằng có sự xếp chồng dữ liệu.

SSPEN : enable chế độ I2C.

SSPM3 : SSPM10 : chọn chế độ với chế độ I2C Master

là: 1000.

- SSPCON2:

SSPCON2: MSSP CONTROL REGISTER 2 (I2C) (ADDRESS 91h mode).

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

- ACKSTAT: bit ACK được nhận từ slave (=0, chỉ dùng trong transmit).

- ACKDT, ACKEN: dùng để phát bit ACK hay NACK từ Master (trong chế độ Receive ACKDT = 0 là ACK,=1 là NACK)

- RCEN : tín hiệu báo hiệu quá trình nhận (chỉ dùng trong Receive, khi RCEN = 1, Master nhận tín hiệu từ slave)

- PEN , RSEN , SEN : bit khởi tạo quá trình truyền stop, restart, start.

Để điều khiển tốc độ baud của chế độ, người ta dùng thanh ghi SSPADD.

I2C làm việc ở 3 chế độ chuẩn (tất nhiên chỉ tương đối) : 100Kb, 400 Kb, 1Mb. Nếu ta dùng thạch anh 4Mhz và cần sử dụng tốc độ 100 Kb thì ta phải nạp giá trị vào thanh ghi SSPADD là 28H với tốc độ 400Kb ta cần giá trị 0Ah.

Còn để lưu và nhận giữ liệu người ta dùng thanh ghi SSPBUF.

Như vậy có cả thảy 5 thanh ghi được dùng đến SSPSTAT, SSPCON1, SSPCON2 (chọn chế độ và điều khiển đường truyền), SSPADD (khởi tạo tốc độ baud) và SSPBUF dùng để lưu trữ dữ liệu trong hai quá trình Recevie và Tranmister.

4.2.3.2. Hàm khởi tạo I2C trong pic 16F87x

Cũng tương tự như khi dùng USTAR, LCD, PWM… đầu tiên ta phải khởi tạo các giá trị ban đầu của chúng. Chúng ta nên tách riêng hàm khởi tạo này thành một chương trình con.

Việc khởi tạo theo các bước như sau:

- Chọn chế độ Master mode bằng việc SSPM3: SSPM0 = 1000.

- Enable Master mode : SSpen=1.

- Chọn baud chuẩn với 100 Kb thì SMP=1.

- Xét tốc độ baud của đường truyền: với 100 Kb thì SSPDD=28h.

4.2.2.3 Quá trình truyền nhận trong PIC 16F87x

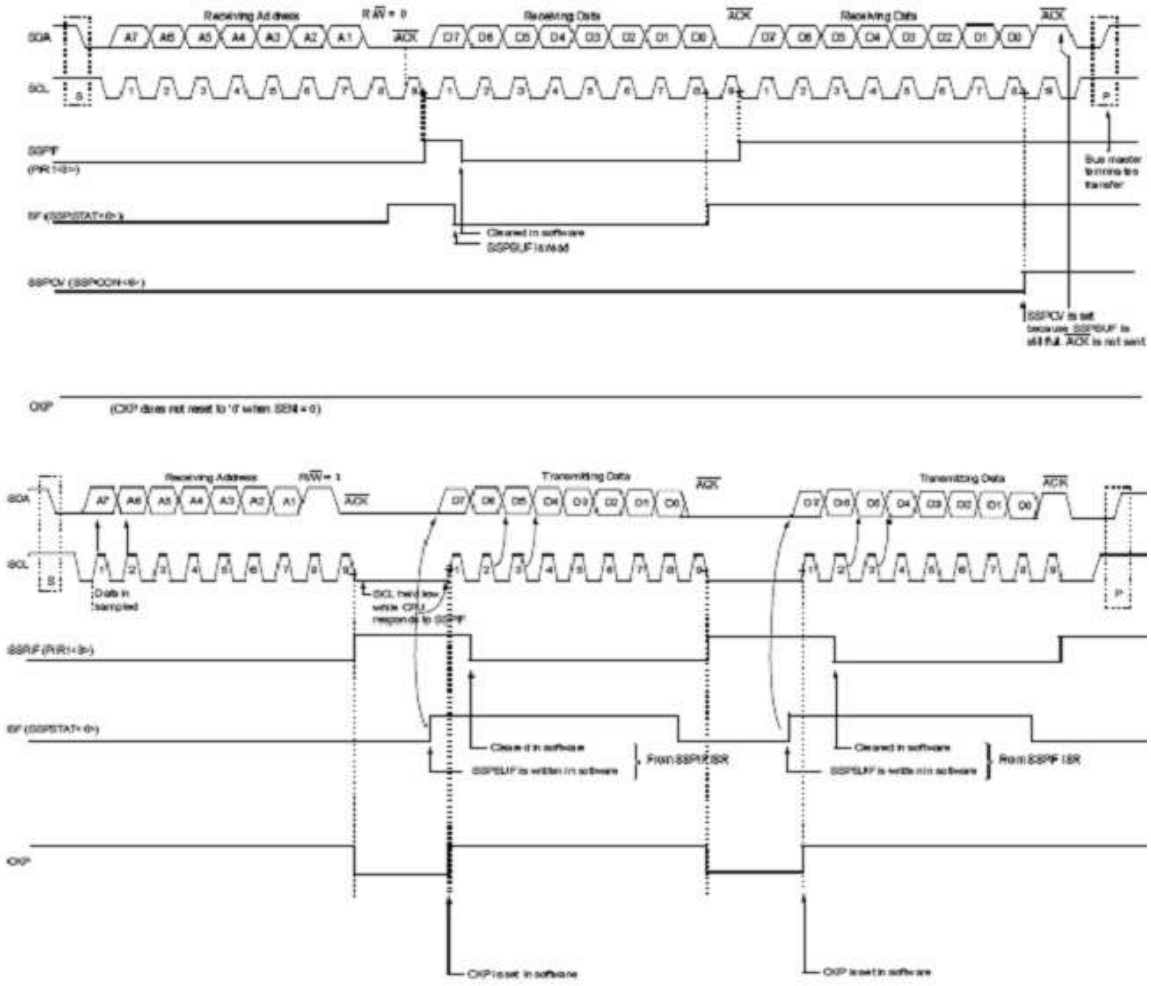
1. Phương thức truyền và nhận dữ liệu trong PIC 16F87x

a. Nhận dữ liệu

Để bắt đầu nhận dữ liệu từ Slave thì Master phải xét bit RCEN = 1, cũng tương tự như các bit Stop, Start…, đây cũng là một hoạt động , sau khi bit cuối cùng được nhận thì RCEN sẽ tự động Clear, do đó bắt đầu quá trình Read. Tiếp theo thì ta sẽ phải tiếp tục SETRCEN (đối với quá trình đọc ít nhất 2 lần trở lên) để tiếp tục một hoạt động mới, đảm bảo rằng quá trình transmit đã kết thúc và sẵn sàng cho hoạt động Receive. Khi quá trình truyền kết thúc ta gửi bit ACK tới Slave báo cho Slave biết rằng Master sẵn sàng nhận dữ liệu tiếp theo.

b. Quá trình truyền dữ liệu

Quá trình truyền dữ liệu ngay sau khi chúng ta ghi dữ liệu cần truyền lên thanh ghi SSPBUF. Quá trình Transmit bắt đầu (lúc bắt đầu ghi dữ liệu lên SSPUF) khi các hoạt động của nó đã kết thúc.



Hình 1.21. Dạng xung của quá trình truyền và nhận PIC 16F877A

2. Hàm MaitMSSP và các bit chức năng trong PIC 16F877A

Để truyền và nhận trong I2C tạo thành khối người ta dùng các bit Stop, Start, Restart, ACK, NACK (ứng với PEN, SEN, RSEN, ACKEN, ACKDT).

Quá trình hoạt động các bit này khá giống nhau để bắt đầu phát đi: ta phải xét bit tương ứng trong thanh ghi đó. Ví dụ muốn gửi bit STOP ta chỉ cần PEN = 1, tương tự với các bit kia và khi truyền xong rồi thì các bit đó sẽ tự động chuyển về 0 (băng Hardware) .

Quá trình này chỉ có tác dụng khi I2C đã hoàn thành xong nhiệm vụ trước đó. Như vậy ta cần biết lúc nào chương trình đã hoàn thành xong nhiệm vụ đây chính là vai trò của cờ SSPIF trong thanh ghi PIR1. Cũng tương tự như các cờ khác , SSPIF báo cho I2C biết là hoạt động đã kết thúc bằng cách Set từ 0 lên 1

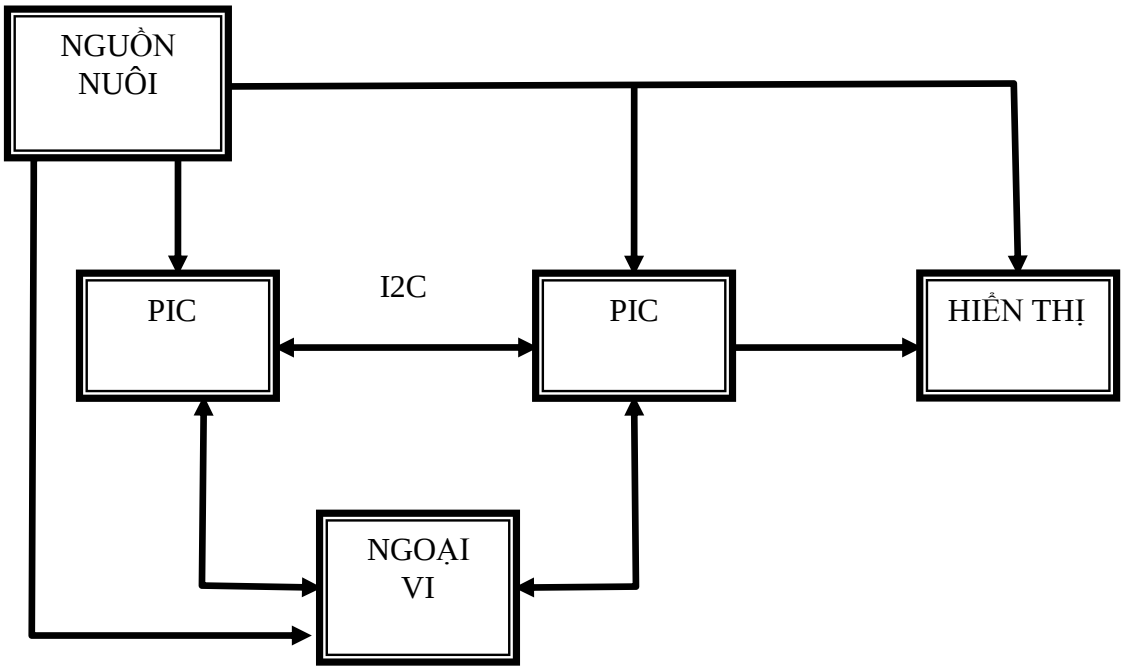
và ta phải xoá cờ này bằng phần mềm cho các hoạt động tiếp theo. Do yêu cầu của I2C là: khi hoạt động này kết thúc thì mới cho phép hoạt động kia bắt đầu.

Có sự khác biệt một chút trong bit ACK bit và NACK bit là: ACKEN gửi bit ACK nói chung đi, còn ACKDT dùng để chọn bit gửi đi là ACK (=0), hay NACK (=1).

CHƯƠNG 2:
THIẾT KẾ HỆ THỐNG GIAO TIẾP I2C GIỮA 2 PIC

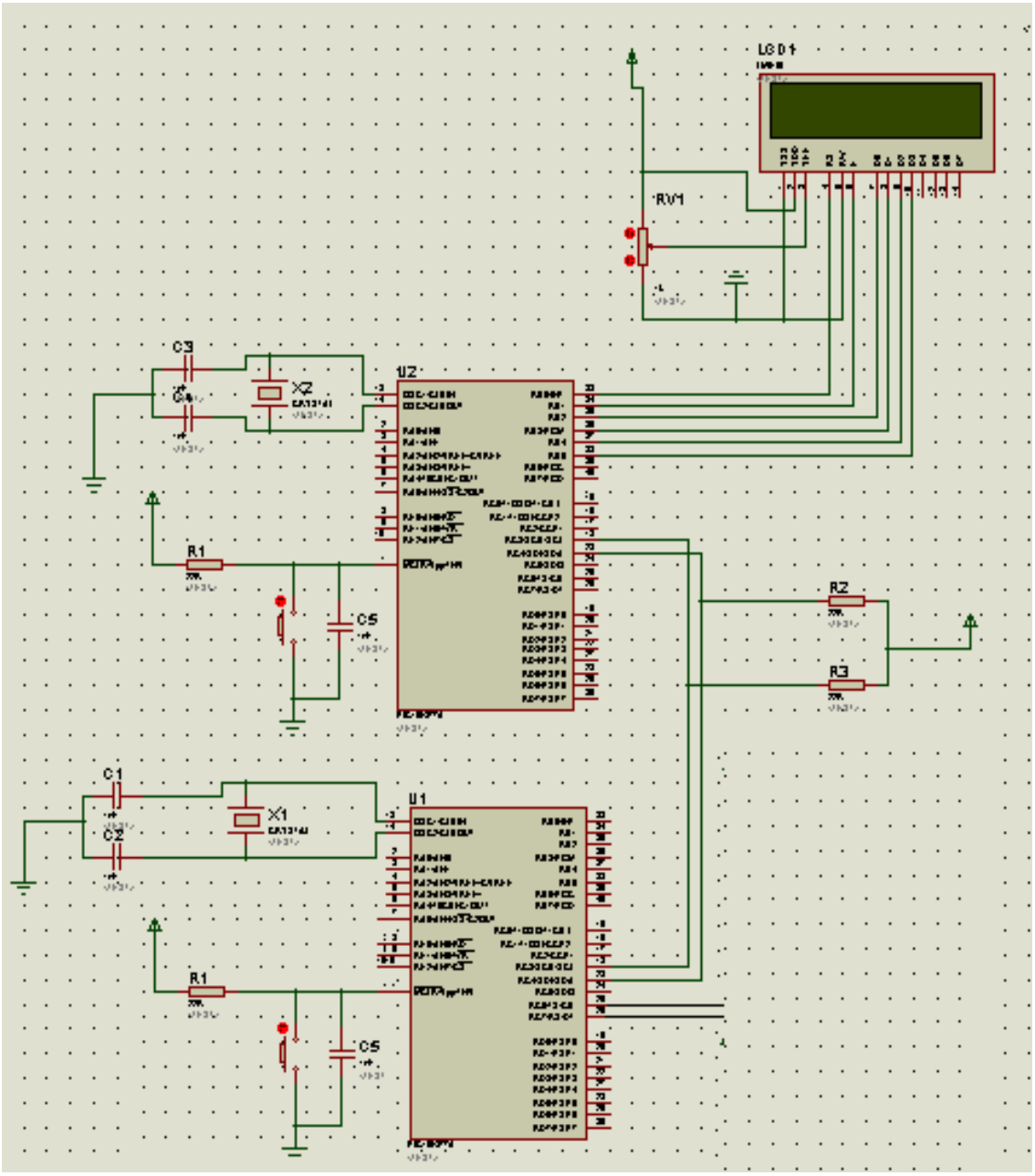
2.1 Sơ đồ khối hệ thống

Với những lí thuyết đã trình bày ở trên ta có thể thiết lập được giao tiếp I2C giữa 2 PIC. Ở phần này em trình bày việc thiết kế hệ thống giao tiếp I2C giữa 2 PIC. 2 PIC giao tiếp I2C với nhau thông qua 2 đường dây là SDA và SCL. Trên 2 dây đều được nối với điện áp dương của nguồn cấp thông qua một điện trở kéo lên. Một PIC em sử dụng có vai trò là PIC chủ (Master) và 1 PIC là tớ (Slave) . Trong hệ thống giao tiếp I2C giữa 2 PIC em sử dụng thêm thiết bị ngoại vi và kết quả của trình thực hiện giao tiếp được hiển thị trên màn hình LCD. 1 PIC sẽ nhận tín hiệu từ thiết bị ngoại vi và thực hiện quá trình xử lí tín hiệu đó. Sau đó sẽ thực hiện giao tiếp I2C. Quá trình xử lí, hiển thị thực hiện trên PIC chủ hoặc tớ. Ở phần cứng của em làm, em sử dụng PIC chủ (master) thực hiện việc nhận, xử lí dữ liệu từ ngoại vi. Sau đó kết quả được truyền qua PIC tớ và nhận ngược lại thông qua giao tiếp I2C. Và hiển thị kết quả nên LCD.



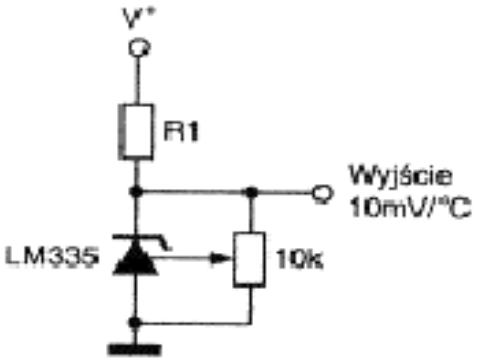
Hình 2.1. Sơ đồ khối hệ thống giao tiếp I2C giữa 2 PIC

2.2 Sơ đồ chi tiết



Hình 2.2. Sơ đồ mạch chi tiết giao tiếp I2C với 2 PIC

Để tường minh việc truyền nhận dữ liệu giữa hai vi điều khiển PIC ở đây em sử dụng mạch đo nhiệt độ dùng cảm biến nhiệt LM335. Tín hiệu điện áp lối ra của cảm biến được đưa vào bộ biến đổi tương tự số của vi điều khiển PIC. Có thể đưa vào hoặc của vi điều khiển chủ (Master) hoặc của vi điều khiển tớ (Slave). Trong thiết kế em đưa tín hiệu vào chân AN0 của vi điều khiển chủ. Sơ đồ mạch đo như hình 2.3 dưới đây.

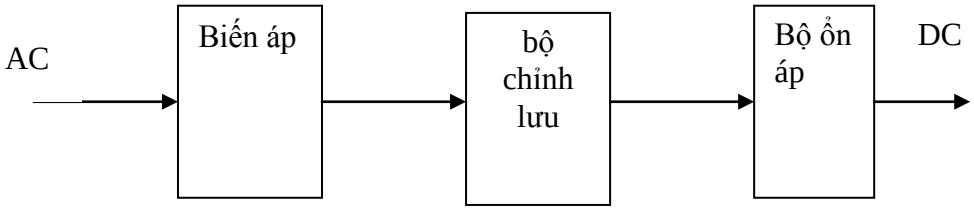


Hình 2.3. Mạch cảm biến đo nhiệt độ dùng để lấy dữ liệu truyền nhận cho giao tiếp I2C.

2.3 Thiết kế các khối

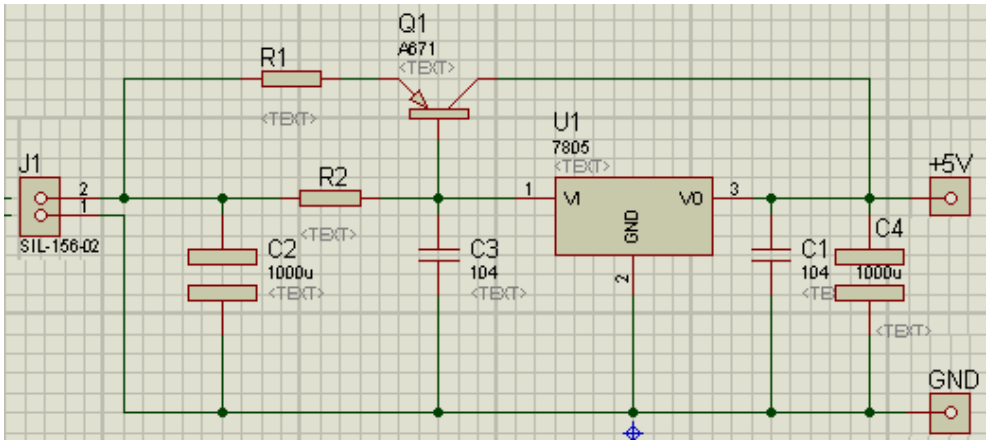
2.3.1 Khối nguồn nuôi

a.Sơ đồ khối



Hình 2.4. Sơ đồ khối nguồn nuôi

b. Sơ đồ nguyên lí



Hình 2.5. Sơ đồ chi tiết mạch nguồn.

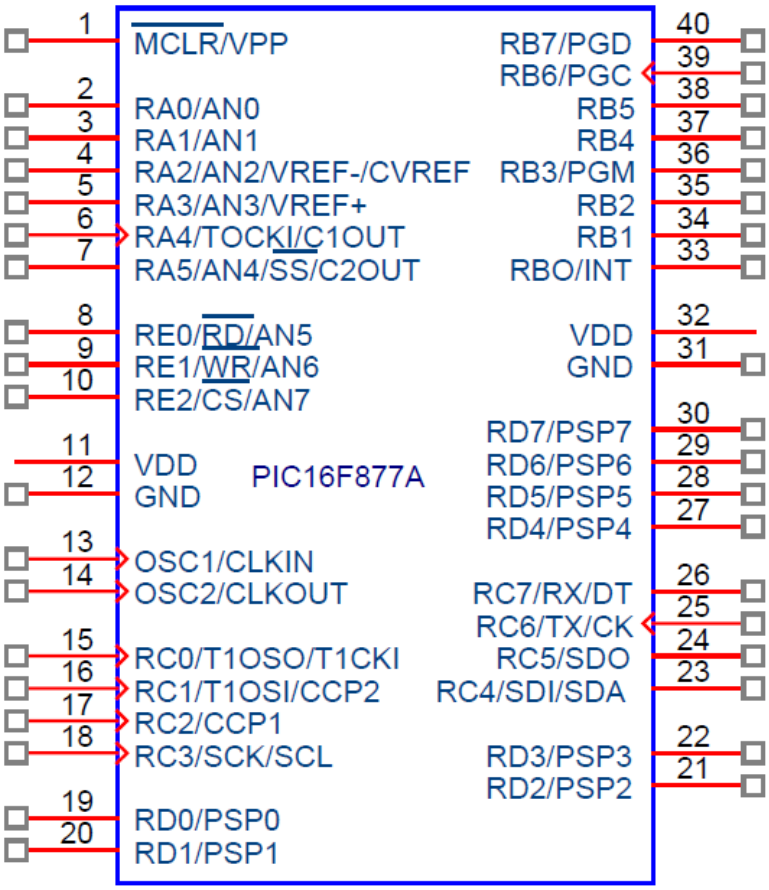
Nguồn ổn định cung cấp cho hệ thống là nguồn +5V.

Mạch trên lấy nguồn một chiều từ một máy biến áp với điện áp từ 6V đến 12V đưa vào ngõ vào. Sau đó cho qua IC ổn áp 7805 để tạo ngõ ra OUT +5V ổn định cấp cho toàn mạch. Tụ điện đóng vai trò ổn định và chống nhiễu cho nguồn.

2.3.2 PIC 26F877A

Ở đây em sử dụng m PIC đóng vai trò là Master, một PIC là Slave, giao tiếp với nhau thông qua giao tiếp I2C.

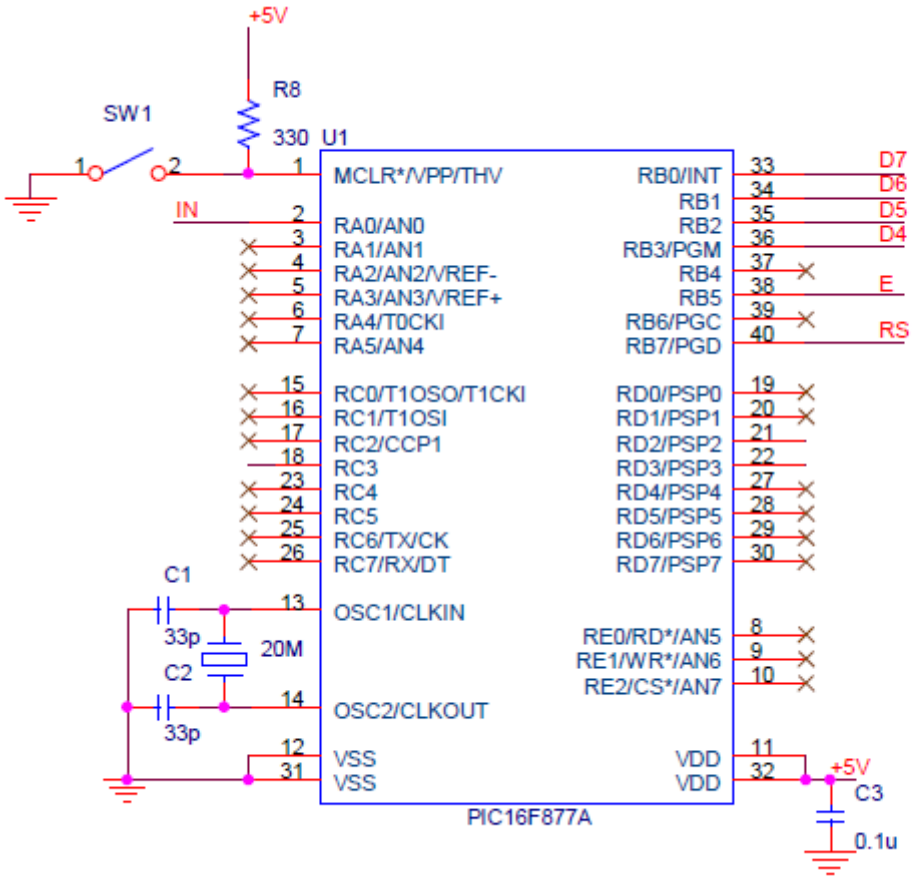
Sơ đồ chân chi tiết vi điều khiển PIC16F877A được cho dưới đây.



Hình 2.6. Sơ đồ chân chi tiết

Để vi điều khiển hoạt động ta cần cấp nguồn cho nó, PIC 16F877A có 4 chân cấp nguồn trong đó chân 11, 32 nối nguồn +5V, chân 12, 31 nối đất. Sau khi cấp nguồn ta cần cung cấp tiếp xung clock cho hoạt động của vi điều khiển. Ở đây ta sẽ dùng thạch anh làm nguồn xung để cấp cho PIC qua chân 13,14 của PIC. Tuy nhiên như ta đã biết, các xung dao động do thạch anh tạo ra cũng không thực sự

ổn định một cách tuyệt đối, và cách khắc phục là gắn thêm các tụ lọc vào thạch anh. Thạch anh sử dụng ở đây là 20MHz. Vậy ta sẽ mắc được sơ đồ mạch nguyên lý của khối này như sau:



Hình 2.7 Sơ đồ mạch 16F877A

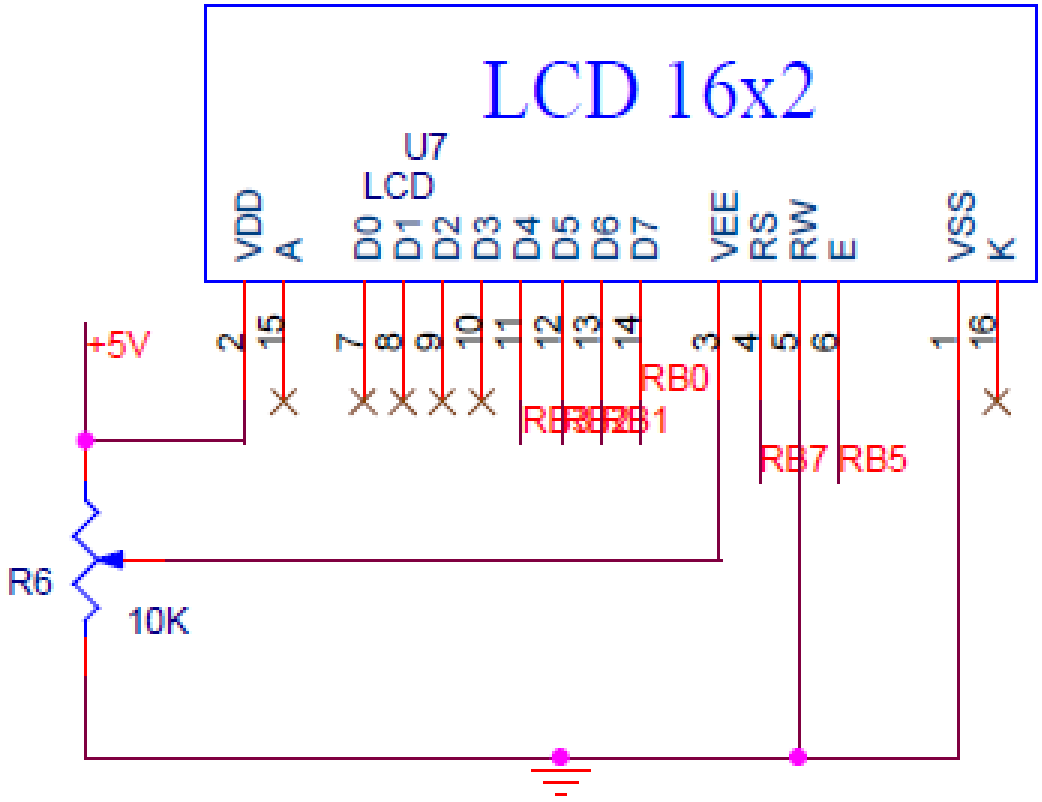
Chân số 1 MCLR được đấu nối thêm như trên đóng vai trò reset PIC, làm việc ở sườn xuống (mức 1 về 0). Khi SW1 mở điện áp vào chân số 1 là +5V (mức 1) PIC không được reset, khi SW1 đóng, mạch kín, chân số 1 nối đất, điện áp vào sẽ là 0V (mức 0) là mức kích hoạt, hoạt động của PIC được reset lại.

Do giao tiếp giữa 2 PIC là giao tiếp I2C nên 2 chân 18 (SDA) 23 (SCL) cuả 2 PIC được nối với nhau. Mỗi dây SDA hay SCL đều được nối với điện áp dương của nguồn cấp thông qua một điện trở kéo lên (full-up resistor).

2.3.3 Khối hiển thị

Ngày nay, thiết bị hiển thị LCD (Liquid Crystal Display) được sử dụng trong rất nhiều các ứng dụng của VĐK. LCD có rất nhiều ưu điểm so với các

dạng hiển thị khác như nó có khả năng hiển thị kí tự đa dạng, trực quan (chữ, số và kí tự đồ họa), dễ dàng đưa vào mạch ứng dụng theo nhiều giao thức giao tiếp khác nhau, tốn rất ít tài nguyên hệ thống và giá thành rẻ... Ở đây em sử dụng HD44780 của Hitachi, một loại thiết bị hiển thị LCD rất thông dụng ở nước ta, cụ thể là sử dụng LCD_DM 1602A).



Hình 2.8 Sơ đồ chi tiết LCD 16x2

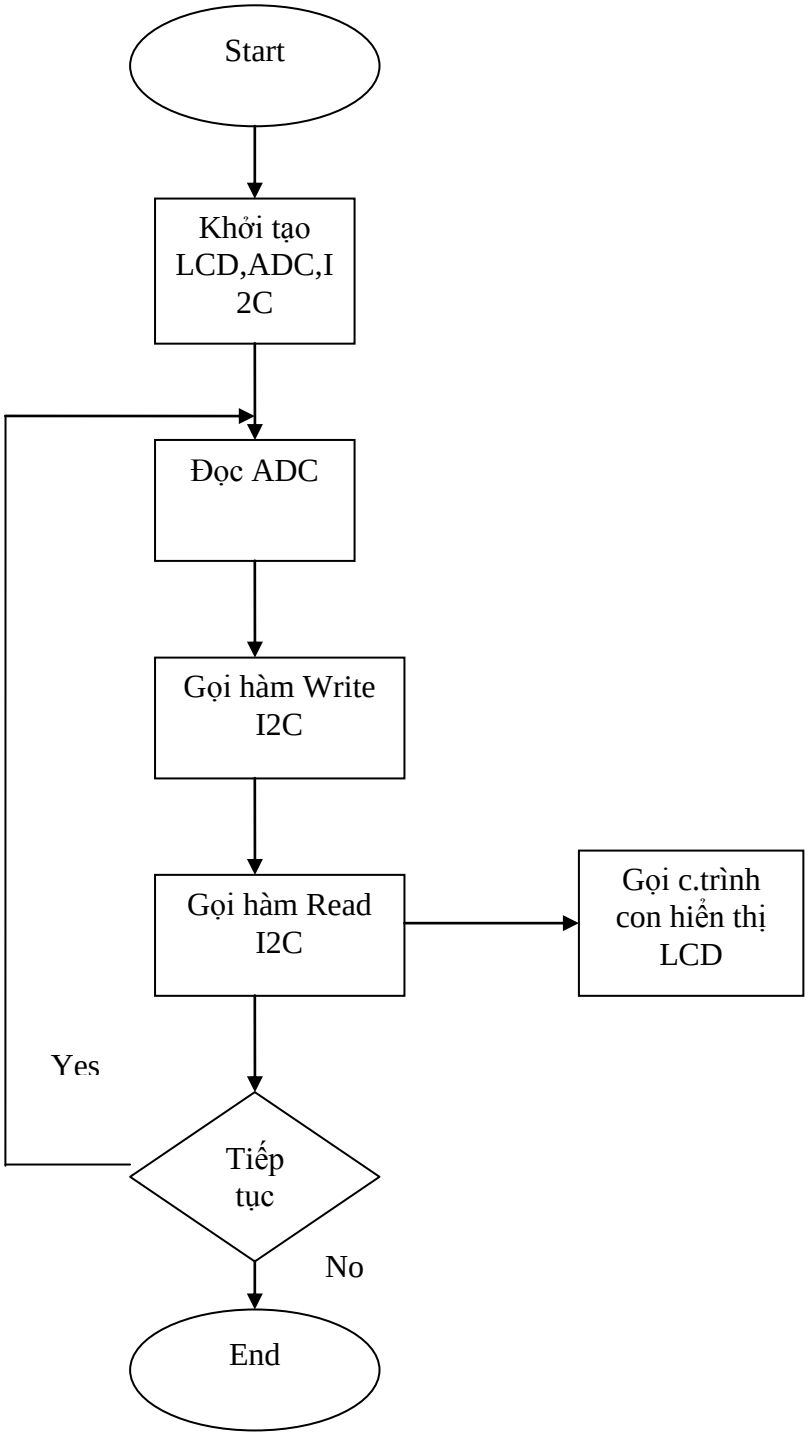
LCD1602 là loại 2 dòng, 16 kí tự, sử dụng nguồn nuôi thấp (từ 2,5 đến 5V). Có thể hoạt động ở hai chế độ 4 bit hoặc 8 bit (trong đề tài này em sử dụng chế độ 4 bit). Với đầu vào 4 bit được lấy từ 4 chân D4→D7 của LCD nối từ RB2→RB5 của vi điều khiển PIC. Chân RW đóng vai trò chọn chế độ đọc ghi cho LCD, mức logic "0" LCD hoạt động ở chế độ ghi, ngược lại ở chế độ đọc. Chân RS của LCD được nối với chân RB6 của Vi điều khiển. Chân E của LCD được nối với chân RB7 của Vi điều khiển. Các tín hiệu điều khiển cho phép hiển thị trên LCD được thực hiện thông qua lập trình.

2.3.4 Khối ngoại vi

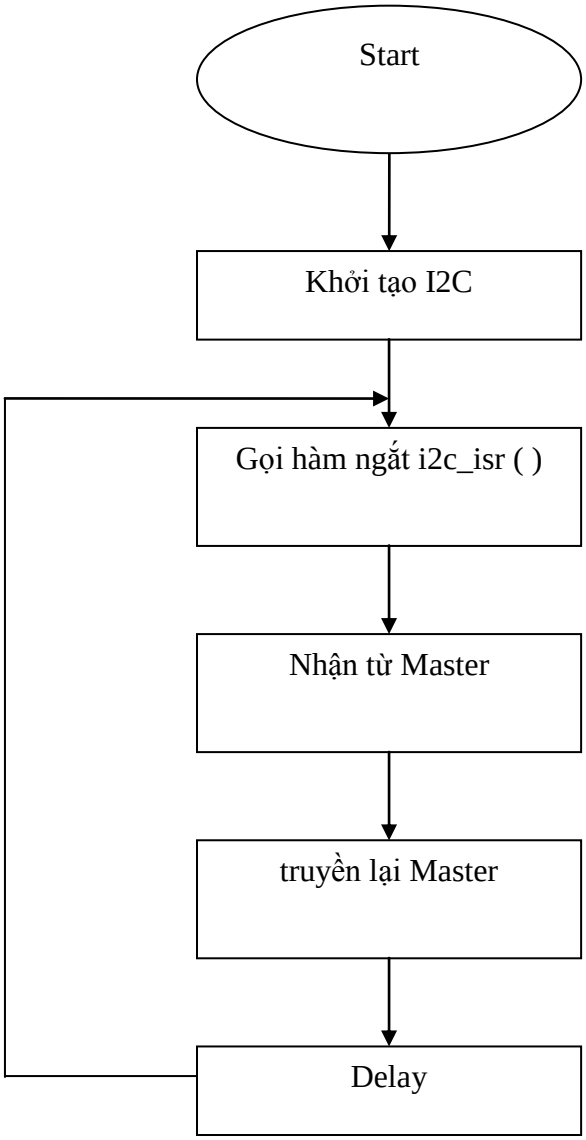
Như đã trình bày ở trên.

2.4 Lưu đồ thuật toán

Với Master



Với Slave


Start
Khởi tạo I2C
Gọi hàm ngắt i2c_isr ( )
Nhận từ Master
truyền lại Master
Delay

2.5 Thiết kế chương trình

Cho Master

```
#include <16F877A.H>
#fuses NOWDT, HS, NOPUT, NOPROTECT, NODEBUG, NOBROWNOUT, NOLVP, NOCPD, NOWRT
#use Delay (Clock=4000000)
#include <lcd_lib_4bit_nguyen.c>
#use i2c (master, sda=PIN_C4, scl=PIN_C3, force_hw)
#use fast_io (b)
#use fast_io (c)
void convert_bcd (unsigned int data)
  {
   int8 d1, d2, d3;
   d1=(int8) data/100;
   d2=(int8) (data/10)%10;
   d3=(int8) data%10;
   d1=d1+0x30;
   d2=d2+0x30;
   d3=d3+0x30;
   lcd_putcmd (line_2+11);
   lcd_putchar (d1);
   lcd_putcmd (line_2+12);
   lcd_putchar (d2);
   lcd_putcmd (line_2+13);
```

```
  lcd_putchar (d3);
  lcd_putcmd (line_2+14);
  lcd_putchar (" ");
  lcd_putcmd (line_2+15);
  lcd_putchar ("C");
}
void write_I2C (int8 value, int8 slave_addr)
{
  i2c_start ();
  i2c_write (slave_addr);
  i2c_write (value);
  i2c_stop ();
}
int8 read_I2C (int8 slave_addr)
{
  int8 value_re;
  i2c_start();
  i2c_write (slave_addr + 1);
  value_re = i2c_read (0);
  i2c_stop ();
  return value_re;
}
void main ()
{
```

```
int8 value;
const int8 slave_addr = 0x10;
set_tris_b (0x00);
set_tris_c (0x80);
set_tris_a (0xff);
lcd_init ();
lcd_putcmd (line_1);
printf (lcd_putchar,"Giao tiep I2C");
lcd_putcmd (line_2);
printf (lcd_putchar,"Value:");
setup_adc_ports (AN0);
setup_adc (ADC_CLOCK_INTERNAL);
set_ADC_channel (0);
read_adc (adc_start_only);
delay_us (10);
while (1)
{
    value=(float) read_adc ();
    write_I2C (value, slave_addr);          // Gui di
    delay_ms (50);
    value = read_I2C (slave_addr);          // Nhan lai
    value = (value - 139.23)/0.513;
    convert_bcd (value);
    delay_ms (10);
```

```
        output_d (value);
    }
}
```

Cho Slave

```
#include <16F877A.H>
#fuses XT,NOWDT,NOPROTECT,NOLVP
#use delay (Clock=4000000)
#ues  i2c (SLAVE,SDA=PIN_C4,SCL=PIN_C3.address=0x10,force_hw)
int8 value = 0x01;
#INT_SSP
Void i2c_isr ( )
{
        Int8 state;
state = i2c_isr_state ();
        if (state < 0x80)
             value = i2c_read ();
        if (state == 0x80)  {
i2c_write (value);
}
       }
        void main ( )
      {
        Enable_int errupts (GLOBAL);
Enable_int erupts  (INT_SSP);
set_tris_b (0x00);
```

```
uhile (1)  {
output_b (value) ;
  }
}
```

KẾT LUẬN

Sau thời gian nghiên cứu và làm đồ án, cùng với sự giúp đỡ tận tình của các thầy cô giáo và các bạn. Đặc biệt là thầy Đoàn Hữu Chức em đã hoàn thành nhiệm vụ đồ án của mình.

Qua đồ án em thấy được ứng dụng quan trọng của vi điều khiển .Giao tiếp I2C giao tiếp trong đo lường và điều khiển, sử dụng vi điều khiển chúng ta thu thập được các đại lượng cần đo, xử lý các đại lượng đó và đưa ra kết quả mong muốn. Hiện nay vi điều khiển rất đa năng, nhỏ gọn, do đó áp dụng vi điều khiển vào trong cuộc sống là rất cần thiết.

Mặc dù rất cố gắng nhưng trong quá trình làm đồ án tốt nghiệp, do sự hạn chế về thời gian, tài liệu và trình độ có hạn nên không tránh khỏi có thiếu sót. Em rất mong được sự góp ý, chỉ bảo của các thầy cô và các bạn để giúp em nâng cao kiến thức, chuyên môn phục vụ cho công việc sau này.

Em xin chân thành cảm ơn!

MỤC LỤC