

**BỘ CÔNG THƯƠNG  
TRƯỜNG ĐẠI HỌC KINH TẾ - KỸ THUẬT CÔNG NGHIỆP  
KHOA ĐIỆN**

**Hà Huy Giáp  
Trần Ngọc Sơn, Nguyễn Đức Điển, Mai Văn Duy**

**TÀI LIỆU HỌC TẬP  
VI ĐIỀU KHIỂN ỨNG DỤNG TRONG ĐO  
LUỜNG VÀ ĐIỀU KHIỂN**

*(Lưu hành nội bộ)*

**HÀ NỘI – 2019**

## LỜI NÓI ĐẦU

Bộ vi xử lý ngày càng phát triển đa năng và được sử dụng hầu hết trong các hệ thống điều khiển trong công nghiệp cũng như trong các thiết bị điện tử dân dụng. Chính vai trò, chức năng của vi xử lý đã đem lại nhiều ưu điểm, tính năng đặc biệt cho các hệ thống điều khiển.

Các nhà nghiên cứu không ngừng nghiên cứu các hệ thống điều khiển và sử dụng vi xử lý để thay thế nhằm nâng cao khả năng tự động thay thế cho con người, và cũng chính vì thế đã thúc đẩy lĩnh vực vi xử lý ngày càng phát triển không ngừng, đáp ứng yêu cầu điều khiển. Để giảm bớt sự phức tạp của phần cứng khi dùng vi xử lý, các nhà khoa học đã tích hợp hệ vi xử lý, bộ nhớ, các ngoại vi thành một mạch duy nhất gọi là vi điều khiển.

Nội dung giáo trình trình bày các kiến thức cơ bản của vi điều khiển. Do có nhiều họ vi điều khiển khác nhau, từ hệ 8 bit cho đến hệ 32 bit, mức độ tích hợp từ đơn giản đến phức tạp, nhiều hãng chế tạo khác nhau, nhiều chủng loại khác nhau có thể làm cho người mới bắt đầu học hay nghiên cứu gặp nhiều bỡ ngỡ không biết bắt đầu từ hệ nào cho phù hợp, chính vì vậy tài liệu trình bày về vi điều khiển 8 bit của hãng Microchip nhằm giúp các bạn sinh viên ngành “Công nghệ kỹ thuật Điều khiển và Tự động hóa” có giáo trình để học tập và nghiên cứu dễ dàng. Do đặc thù tính ứng dụng của ngành, giáo trình lựa chọn dòng vi điều khiển 18F4431, dòng vi điều khiển này tính năng tương tự như DsPIC30F4011 (16 bit). Vì vậy về tính ứng dụng có thể sử dụng vi điều khiển DsPIC30F4011 (16 bit) để triển khai các ứng dụng thực tế, thay vì sử dụng 18F4431. Các ứng dụng trong giáo trình, các tác giả trình bày về 2 nội dung: “Ứng dụng vi điều khiển trong hệ thống đo lường, điều khiển các ngoại vi cơ bản và thiết kế bộ điều khiển PID số”.

Trong quá trình biên soạn, mặc dù các tác giả đã rất cố gắng, nhưng do trình độ và thời gian có hạn, tài liệu không tránh khỏi những sai sót. Chúng tôi mong nhận được góp ý và nhận xét của bạn đọc để cuốn sách được hoàn thiện hơn trong lần tái bản sau..

### Các tác giả

## MỤC LỤC

LỜI NÓI ĐẦU -----	2
MỤC LỤC-----	3
CHƯƠNG 1. TỔNG QUAN VỀ VI ĐIỀU KHIỂN PIC-----	5
1.1. CẤU TRÚC CỦA VI ĐIỀU KHIỂN-----	5
1.2. KIẾN TRÚC CỦA VI ĐIỀU KHIỂN-----	8
1.3. MỘT SỐ HỌ VI ĐIỀU KHIỂN THÔNG DỤNG-----	10
1.4. VI ĐIỀU KHIỂN PIC18F4431-----	12
CÂU HỎI HƯỚNG DẪN ÔN TẬP, THẢO LUẬN -----	27
CHƯƠNG 2. CÁC TÀI NGUYÊN CƠ BẢN CỦA VI ĐIỀU KHIỂN PIC18F-----28	28
2.1. CÁC PHẦN MỀM LẬP TRÌNH-----28	28
2.1.1. Phần mềm MPLAB -----	30
2.1.2. Phần mềm CCS-----	31
2.1.3. Phần mềm MikroC-----	32
2.2. PHẦN MỀM MPLAB VÀ XC8 -----	33
2.3. HOẠT ĐỘNG NGẮT -----	45
2.3.1. Giới thiệu-----	45
2.3.2. Tổ chức ngắn của PIC 18F4431 -----	46
2.4. HOẠT ĐỘNG VÀO/RA (I/O) -----	60
2.4.1. PORTA -----	61
2.4.2. PORTB -----	64
2.4.3. PORTC -----	66
2.4.4. PORTD -----	69
2.4.5. PORTE -----	71
2.5. BỘ ĐỊNH THỜI TIMER-----72	72
2.5.1. Timer0-----	73
2.5.2. Timer1-----	76
2.5.3. Timer2-----	80
2.6. KHÓI CCP (Capture – Compare – PWM)-----81	81
2.6.1. Chế độ PWM -----	81
2.6.2. Các thanh ghi liên quan -----	83
2.6.3. Sử dụng các bộ PWM -----	84
2.7. BỘ ĐIỆN ĐỒI ADC-----85	85

2.8. TRUYỀN THÔNG NỐI TIẾP TRONG VI ĐIỀU KHIỂN -----	95
CÂU HỎI HƯỚNG DẪN ÔN TẬP, THẢO LUẬN -----	110
BÀI TẬP ỦNG DỤNG-----	111
CHƯƠNG 3. LẬP TRÌNH CÁC ỦNG DỤNG VỚI VI ĐIỀU KHIỂN PIC 18F -----	112
3.1. GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH C-----	112
3.2. LẬP TRÌNH VÀ GIAO TIẾP VỚI NÚT NHẤN, CẢM BIẾN LOGIC --	116
3.3. LẬP TRÌNH VÀ GIAO TIẾP VỚI LED 7 THANH VÀ LCD 16x2 -----	120
3.3.1. Lập trình và giao tiếp với LED 7 thanh -----	120
3.3.2. Lập trình và giao tiếp với LCD 16x2-----	124
3.4. LẬP TRÌNH VÀ GIAO TIẾP CÁC TÍN HIỆU TƯƠNG TỰ -----	126
3.4.1. Lập trình giao tiếp với cảm biến nhiệt độ-----	128
3.4.2. Lập trình giao tiếp với cảm biến dòng điện, điện áp -----	131
3.5. LẬP TRÌNH ỦNG DỤNG SỬ DỤNG GIAO TIẾP I2C -----	134
3.5.1. Giao tiếp DS1307 -----	134
3.5.2. Giao tiếp EEPROM 24C256 -----	140
3.6. LẬP TRÌNH ỦNG DỤNG SỬ DỤNG GIAO THỨC SPI -----	142
3.7. LẬP TRÌNH BỘ ĐIỀU KHIỂN PID -----	147
3.7.1. Một số vấn đề kỹ thuật khi thực hiện hệ điều khiển số với vi điều khiển.	147
3.7.2. Quy trình thực hiện hệ thống điều khiển số -----	148
3.7.3. Luật điều khiển PID -----	149
CÂU HỎI ÔN TẬP VÀ THẢO LUẬN -----	157
BÀI TẬP ỦNG DỤNG -----	157
CHƯƠNG 4. THIẾT KẾ MẠCH VÀ MÔ PHỎNG VI ĐIỀU KHIỂN PIC 18F -----	159
4.1. CÁC PHẦN MỀM THIẾT KẾ MẠCH -----	159
4.1.1. Phần mềm Proteus -----	159
4.1.2. Phần mềm Altium -----	163
4.2. THIẾT KẾ MẠCH CHO VI ĐIỀU KHIỂN PIC 18F -----	165
4.3. MÔ PHỎNG MẠCH PIC 18F -----	168
CÂU HỎI ÔN TẬP VÀ THẢO LUẬN -----	172
BÀI TẬP ỦNG DỤNG -----	172
TÀI LIỆU THAM KHẢO-----	173

# CHƯƠNG 1

## TỔNG QUAN VỀ VI ĐIỀU KHIỂN PIC

### MỤC TIÊU CỦA CHƯƠNG

- Hiểu cấu trúc chung của vi điều khiển. Biết được một số họ vi điều khiển thông dụng
- Nắm được cấu trúc và thiết kế phần cứng của vi điều khiển 18F4431

#### 1.1. CẤU TRÚC CỦA VI ĐIỀU KHIỂN

Vi điều khiển (MCU – viết tắt của cụm từ ‘Micro Control Unit’) có thể được coi như một máy tính thu nhỏ trên một chip, nó có thể hoạt động với một vầng linh kiện phụ trợ bên ngoài. Vi điều khiển khác với vi xử lý ở những điểm sau:

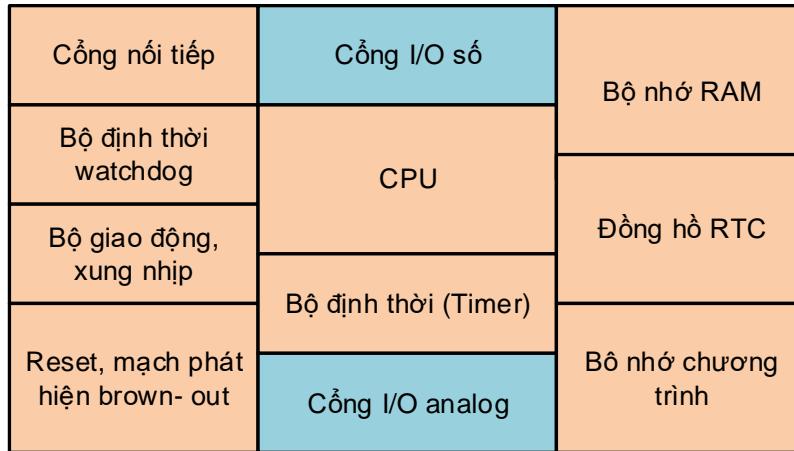
- Về cấu trúc: Vi xử lý là một CPU trên một chip còn vi điều khiển là một chip có chứa CPU, bộ nhớ, mạch vào/ra và các mạch đặc biệt khác như bộ đếm/định thời, mạch biến đổi A/D, D/A, ... Như vậy, về cấu trúc thì vi điều khiển chính là một hệ vi xử lý thu nhỏ.

- Về ứng dụng: Các bộ vi xử lý chủ yếu được dùng làm CPU trong các máy tính còn các bộ vi điều khiển được dùng trong các ứng dụng hướng điều khiển.

- Về tập lệnh: Tập lệnh cho vi xử lý là những lệnh mang tính chất tổng quát nên chúng được dung nhiều với kiểu định địa chỉ, cho phép thao tác với lượng dữ liệu lớn. Ngược lại, tập lệnh của vi điều khiển chủ yếu là những lệnh vào/ra đơn giản và các lệnh xử lý bit.

- Về bộ nhớ: Máy tính là thiết bị đa dụng nên các chương trình ứng dụng thường được lưu ở các thiết bị lưu trữ ngoài như đĩa cứng, đĩa quang, ổ Flash. Khi cần thực thi, chương trình được nạp vào bộ nhớ RAM để giải mã lệnh và thực thi. Như vậy, với máy tính thì RAM chính là bộ nhớ chương trình, còn ROM trong máy tính thường dùng để lưu các thông tin về cấu hình của máy và các chương trình vào ra cơ bản (BIOS). Điều này giải thích vì sao trong máy tính RAM có dung lượng lớn hơn ROM rất nhiều lần. Ngược lại, ở vi điều khiển thì chương trình được chứa trong ROM vì chúng là chương trình điều khiển ứng dụng, hầu như không thay đổi nội dung, còn RAM được dùng để chứa dữ liệu tạm thời cho chương trình như trạng thái các chân vào/ra, nội dung các biến được khai báo trong chương trình. Do đó ở vi điều khiển thì ROM có dung lượng lớn hơn RAM nhiều lần.

Chúng ta hãy xem xét các bộ phận khác nhau của vi điều khiển (hình 1.1).



Hình 1.1. Cấu trúc Vi điều khiển

**1. CPU:** Khối xử lý trung tâm CPU (Central Processing Unit) là bộ phận quan trọng nhất của vi điều khiển. Nó thực hiện chức năng tìm nạp các lệnh được lưu trữ trong bộ nhớ chương trình, giải mã các lệnh này, và thực hiện chúng. Chính bản thân CPU cũng là sự kết hợp của các thanh ghi, đơn vị số học và logic (ALU), bộ giải mã lệnh, và hệ thống mạch điều khiển.

**2. Bộ nhớ chương trình:** Chứa tập lệnh tạo nên chương trình. Để thích ứng với những chương trình lớn hơn, trong một số vi điều khiển, bộ nhớ chương trình được chia thành bộ nhớ bên trong và bộ nhớ bên ngoài. Bộ nhớ chương trình thường là các loại ổn định và loại EEPROM hoặc EPROM hoặc flash, Mask ROM, hoặc loại lập trình được một lần OTP (One Time Programmable).

**3. RAM:** là bộ nhớ dữ liệu của bộ vi điều khiển có nghĩa là vi điều khiển sử dụng nó để lưu trữ dữ liệu. CPU dùng RAM để lưu trữ các biến cũng như ngăn xếp. CPU sử dụng ngăn xếp để lưu trữ địa chỉ trả về sau khi hoàn thành một chương trình con hoặc một lời gọi ngắn. Nhờ đó, CPU có thể tiếp tục thực hiện chương trình chính.

**4. Bộ tạo dao động:** Bộ vi điều khiển thực thi chương trình ở một tốc độ nhất định. Tốc độ này được xác định thông qua tần số của bộ tạo dao động. Bộ tạo dao động có thể là một mạch dao động RC hoặc bộ dao động với một bộ phận đồng bộ ở bên ngoài chẳng hạn như thạch anh, hoặc mạch cộng hưởng LC hoặc thậm chí là một mạch RC. Bộ dao động bắt đầu hoạt động ngay sau khi bộ vi điều khiển được cấp nguồn nuôi.

**5. Mạch khởi động lại và mạch phát hiện sụt áp nguồn nuôi thấp:** mạch khởi động lại đảm bảo tắt cả các linh kiện và mạch điều khiển bên trong bộ vi điều khiển được khởi tạo ở trạng thái ban đầu xác định, đồng thời các thanh ghi cần thiết cũng được khởi tạo hợp lý khi vi điều khiển bắt đầu đưa vào hoạt động. Bộ phát hiện sụt áp nguồn nuôi là một mạch giám sát điện áp nguồn nuôi (Reset and Brown – out detector circuit). Nếu có sự sụt áp bất thường nó sẽ khởi tạo lại bộ vi xử lý và vì thế

không làm sai lệch nội dung của bộ nhớ và thanh ghi, nếu không bộ vi điều khiển có thể rơi vào tình trạng hoạt động thiếu chính xác.

**6. Cổng nối tiếp:** Cổng nối tiếp là một bộ phận có tác dụng rất lớn đối với hoạt động của vi điều khiển vì được sử dụng để truyền thông với các thiết bị ngoại vi thông qua việc truyền dữ liệu nối tiếp. Cổng nối tiếp có thể hoạt động ở bất kỳ tốc độ truyền dữ liệu nào. Nó nhận byte dữ liệu từ bộ vi điều khiển và chuyển từng bit dữ liệu ra ngoài. Tương tự nó nhận từng bit dữ liệu từ bên ngoài, gộp 8 bit thành một byte và gửi đến bộ vi điều khiển. Có hai kiểu truyền dữ liệu qua cổng nối tiếp là truyền đồng bộ và không đồng bộ. Trong truyền dữ liệu đồng bộ, mỗi bit dữ liệu cần có một tín hiệu xung nhịp đi kèm để thực hiện việc đồng bộ, trong khi đó việc truyền dữ liệu không đồng bộ không cần tín hiệu đó, thông tin đồng bộ và việc đồng bộ được gói trọn trong chính bit dữ liệu thông qua khoảng thời gian của các bit dữ liệu và các bit start, bit stop được bù xung thêm vào đường truyền dữ liệu.

**7. Cổng vào ra số:** Bộ vi điều khiển sử dụng cổng vào/ra số để trao đổi dữ liệu số với thế giới bên ngoài. Khác với cổng nối tiếp truyền dữ liệu nối tiếp từng bit một, cổng vào/ra số trao đổi dữ liệu theo từng byte một.

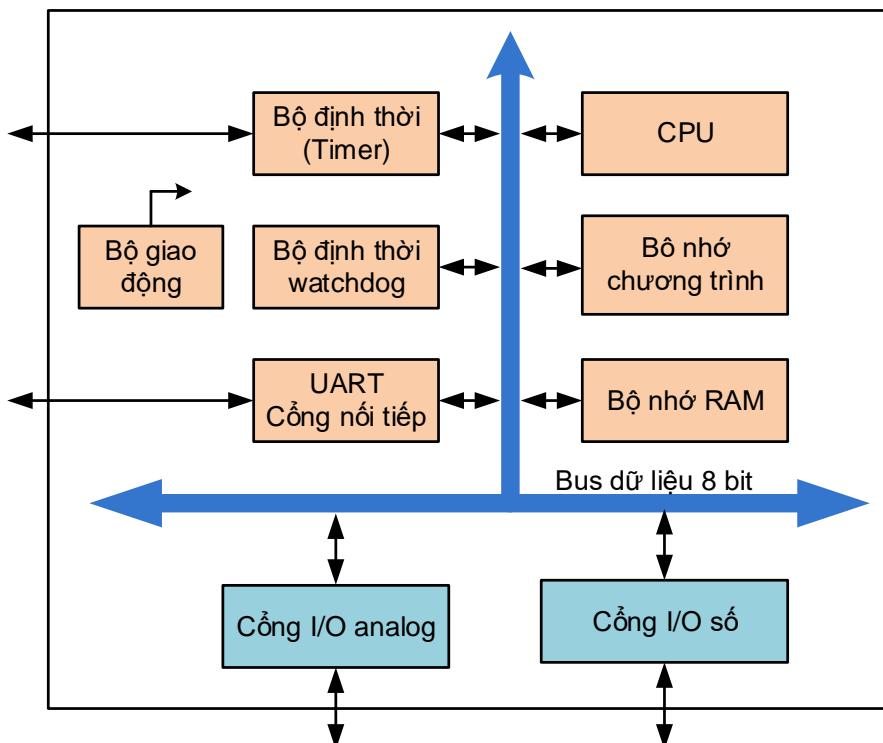
**8. Cổng vào tương tự:** Tín hiệu lối vào tương tự được xử lý qua một bộ biến đổi tương tự - số (ADC). Bộ vi điều khiển có thể có một ADC hoặc một bộ so sánh tương tự được điều khiển bởi phần mềm để thực hiện việc chuyển đổi tương tự - số. Bộ biến đổi ADC nhận dữ liệu từ những thiết bị như các cảm biến (sensor) nhiệt độ, cảm biến áp suất. Các bộ cảm biến này thường cung cấp các tín hiệu điện áp dưới dạng tương tự.

**9. Bộ định thời:** Bộ vi điều khiển sử dụng bộ định thời để quy định thời gian các sự kiện, chẳng hạn xuất dữ liệu ra màn hình với một tần số nào đó. Bộ vi điều khiển sẽ dùng bộ định thời để tạo ra tần số đó. Bộ định thời cũng được dùng để đếm các sự kiện xảy ra ở bên ngoài cũng như bên trong. Trong trường hợp đó bộ định thời được gọi là bộ đếm.

**10. Bộ định thời watchdog WDT (Watchdog Timer):** Là một bộ định thời đặc biệt thường dùng để ngăn ngừa những sự cố phần mềm. WDT hoạt động như sau: nó làm tăng giá trị một bộ đếm bên trong với một tốc độ đếm nào đó. Nếu chương trình người dùng không đặt lại bộ đếm thì bộ đếm sẽ bị tràn làm cho vi điều khiển được khởi động lại. Phần mềm của người dùng được lập trình một cách hợp lý sao cho WDT được đặt lại một cách đều đặn. Nếu chương trình người dùng bộ lỗi, nó không đặt lại WDT thì tốt hơn hết là khởi động lại hệ thống. Hệ thống sẽ gấp sự cố hoặc rơi vào tình trạng không hoạt động (treo).

**11. RTC (Real Time Clock):** Đồng hồ định thời gian thực (RTC) là một bộ định thời đặc biệt có nhiệm vụ lưu trữ các thông tin về ngày tháng.

Hình 1.1 chỉ minh họa một hệ vi điều khiển điển hình trong khi những thiết bị này rất đa dạng về kích thước và độ phức tạp. Cũng như các bộ vi xử lý (tức là CPU trên một chip), các bộ vi điều khiển cũng được phân loại theo độ rộng của các thanh ghi bên trong và thanh ghi tổng là 8 bit, 16 bit, 32 bit (hoặc 64 bit). Thông thường một hệ thống 8 bit cũng có nghĩa là CPU được nối với các bộ phận khác thông qua bus dữ liệu 8 bit. Hình 1.2 minh họa cho khái niệm này.



*Hình 1.2. Cấu trúc Vi điều khiển 8 bit*

Các bộ vi điều khiển với đường bus dữ liệu lớn có thể hoạt động tốt hơn nhưng giá của các công cụ giúp phát triển nó lại đắt hơn so với các vi điều khiển với đường bus dữ liệu nhỏ hơn. Các bộ vi điều khiển 8 bit là phổ biến nhất không chỉ vì giá thấp (so với vi điều khiển 16 bit, 32 bit) mà còn vì các công cụ phát triển cho chúng cũng rẻ hơn nhiều. Hiện nay, chúng được chế tạo với các chỉ tiêu chất lượng và mức độ tích hợp các thiết bị ngoại vi ngày càng cao.

Bên cạnh cách phân loại dựa trên độ rộng của đường bus dữ liệu bên trong, các bộ vi điều khiển còn được phân loại dựa trên kiến trúc nền tảng của chúng. Phần tiếp theo sẽ xem xét tới kiến trúc của bộ vi điều khiển

## 1.2. KIẾN TRÚC CỦA VI ĐIỀU KHIỂN

Kiến trúc của vi điều khiển được phân loại dựa trên những chỉ tiêu khác nhau. Một cách phân loại thường gặp nhất là dựa vào tập lệnh và một chỉ tiêu quan trọng là số lượng các lệnh. Theo cách đó vi điều khiển được phân ra:

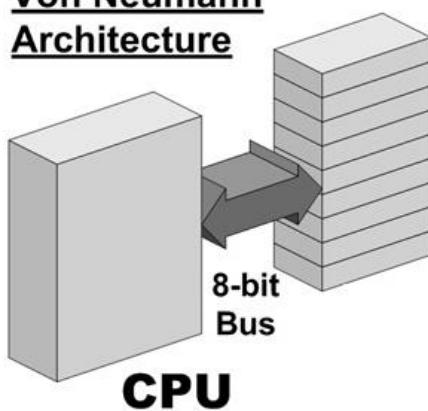
- Máy tính có tập lệnh phức tạp CISC (complex instruction set computer),
- Máy tính có tập lệnh rút gọn RISC (reduced instruction set computer), và
- Máy tính có tập lệnh tối thiểu MISC (minimal instruction set computer).

Một cách phân loại khác là dựa trên cách thức truy cập bộ nhớ dữ liệu và bộ nhớ chương trình. Nếu chúng hợp nhất thành bộ nhớ duy nhất thì gọi là kiến trúc Princeton hay kiến trúc Von Neumann. Ngược lại nếu chúng được tách rời nhau thì gọi là kiến trúc Harvard.

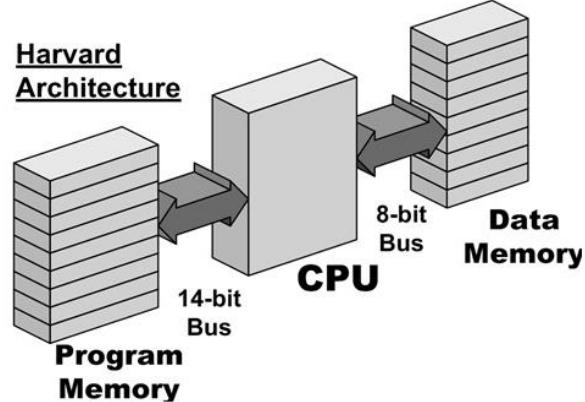
### 1.2.1. Kiến trúc Von Neumann

Cách tổ chức hệ thống bộ nhớ của kiểu kiến trúc này là cả bộ nhớ chương trình và bộ nhớ dữ liệu được xem như một vùng nhớ, dùng chung một Bus dữ liệu. Thời kỳ đầu của kỹ nguyên máy tính, bộ nhớ không có độ tin cậy như hiện nay và hay tạo ra những lỗi hệ thống. Chính vì vậy mà kiểu kiến trúc này được ưa chuộng, bởi vì nó dễ dàng thiết kế, nâng cao độ tin cậy của hệ thống và dễ dàng thay thế những vùng nhớ bị lỗi kỹ thuật. Nhờ những lợi thế đó mà trong một thời gian kiểu kiến trúc này đã được thương mại và sản xuất. Tuy nhiên nó cũng một số nhược điểm: Hạn chế băng thông, thực hiện nhiều lần lấy dữ liệu từ bộ nhớ chỉ cho một lệnh; hạn chế về tốc độ, một lần lấy lệnh từ bộ nhớ chương trình và một lần trả về kết quả trong bộ nhớ dữ liệu, không thể thực hiện song song thao tác này. Chính vì sự phô biến đầu tiên này của kiến trúc Von Neumann mà hầu hết các loại vi điều khiển đều được xây dựng quanh cấu trúc này, mặc dù giá thành bộ nhớ hiện nay đã rẻ hơn rất nhiều và độ tin cậy cũng tăng lên rất nhiều.

Von Neumann  
Architecture



a. Kiến trúc Von Neumann



b. Kiến trúc Harvard

Hình 1.3. Kiến trúc Vi điều khiển

### 1.2.2. Kiến trúc Harvard

Harvard có không gian nhớ dành cho bộ nhớ dữ liệu và bộ nhớ chương trình riêng biệt. Lợi thế về hiệu năng chính của kiểu kiến trúc này là nó có 2 bus dữ liệu riêng biệt hoạt động đồng thời phục vụ cho bộ nhớ dữ liệu và bộ nhớ chương trình:

Trong khi CPU lấy lệnh (dữ liệu) từ bộ nhớ chương trình, thì nó vẫn có thể đọc ghi dữ liệu ở vùng nhớ dữ liệu. Một lợi thế khác của kiểu kiến trúc Harvard này là độ rộng của Bus chương trình và Bus dữ liệu có thể khác nhau. Không phải tất cả các loại vi điều khiển có kiến trúc Harvard đều có lợi thế này, nhưng PIC thì có. Do Bus có độ rộng khác nhau nên độ rộng Bus bộ nhớ chương trình có thể rộng hơn bộ nhớ dữ liệu. Với PIC 8-bit thì Bus dữ liệu luôn là 8-bit, tuy nhiên Bus bộ nhớ chương trình có thể rộng hơn, bao nhiêu tùy thuộc và mục đích của loại PIC đó. Với PIC 8-bit thì có 3 loại được phân chia thành loại có độ rộng Bus bộ nhớ chương trình là 12-bit, 14-bit và 16-bit. Bus bộ nhớ chương trình rộng hơn sẽ đưa dữ liệu từ bộ nhớ chương trình nhiều hơn cũng trong 1 chu kỳ máy.

Một cách phân loại khác nữa là dựa vào cách lưu trữ và thao tác dữ liệu trong CPU. Thao tác dữ liệu là công việc của một vi điều khiển. Một bộ vi điều khiển (hoặc một bộ vi xử lý) thao tác dữ liệu theo chương trình của người dùng. Phương thức lưu trữ và truy cập dữ liệu trong CPU cũng như phương thức xử lý chúng tạo nên sự khác nhau trong kiến trúc các bộ vi điều khiển. Có 4 mô hình cơ bản: ngăn xếp, thanh ghi tổng, thanh ghi – bộ nhớ và thanh ghi – thanh ghi.

Các kiến trúc vi điều khiển trước đây dùng mô hình ngăn xếp hoặc mô hình thanh ghi tổng. Tuy nhiên, hầu hết các vi điều khiển hiện nay dùng kiến trúc thanh ghi – thanh ghi. Nguyên nhân là việc truy nhập các thanh ghi bên trong nhanh hơn nhiều so với việc truy nhập bộ nhớ ngoài. Để giảm số lần truy nhập bộ nhớ ngoài, mô hình thanh ghi – thanh ghi được xây dựng với một số lượng lớn các thanh ghi đa năng. Hơn nữa, một trình biên dịch truy nhập các thanh ghi dễ dàng hơn là truy nhập một ngăn xếp mặc dù ngăn xếp nằm trong bộ vi điều khiển.

### **1.3. MỘT SỐ HỌ VI ĐIỀU KHIỂN THÔNG DỤNG**

#### **1.3.1. Vi điều khiển của Atmel**

Atmel là một hãng cung cấp vi điều khiển lớn, sản phẩm vi điều khiển của Atmel gồm:

- Dòng vi điều khiển dựa trên kiến trúc 8051 của Intel như 83xx, 87xx, 89xx, ...
- Dòng vi điều khiển AT91CAP như AT91CAP7S250A, AT91CAP7S450A,... với tần số hoạt động từ 80 đến 200 Mhz, 2 đến 4 kênh PWM, 10 kênh ADC 10 bit, ghép nối được với các module SDRAM ngoài.
- Dòng vi điều khiển AT91SAM 32-bit ARM – based với bộ nhớ chương trình có thể lên tới 2MB, tần số hoạt động đến 240 MHz
- Dòng AVR 8 bit kiến trúc RISC như AT90PWM1, ATmega 8, ATmega 16, ATmega 32, ATmega 128, ...

- Dòng AVR32 32-bit MCU/DSP như AVR 32 UC3A, AVR 32 UC3B, ... là những bộ vi điều khiển 32 bit có thêm các lệnh xử lý tín hiệu số để xử lý âm thanh, hình ảnh.

### **1.3.2. Vi điều khiển của Microchip**

- Dòng 8 bit như PIC10, PIC12, PIC14, PIC16, PIC18 với bộ nhớ kiểu Flash, OTP, ROM dung lượng từ 0,5 đến 256K Byte.

- Dòng 16 bit như PIC24F, PIC24H.

- Dòng xử lý tín hiệu số 16 bit như dsPIC30Fxxxx, dsPIC33FJxxxx.

### **1.3.3. Vi điều khiển của Cypress**

Cypress nổi tiếng với dòng sản phẩm PsoC, đây là những vi mạch có tích hợp vi điều khiển, các linh kiện tương tự (các bộ khuếch đại, các bộ biến đổi A/D, D/A, các bộ lọc, các bộ so sánh,...) và các linh kiện số (bộ định thời, bộ đếm, bộ tạo xung PWM, SPI, UART, I2C,...) trên một chip duy nhất. Việc tích hợp hàng trăm khối chức năng cùng với một bộ vi điều khiển trên một chip cho phép giảm thời gian thiết kế, thu gọn kích thước sản phẩm, giảm công suất tiêu thụ và giảm giá thành sản phẩm.

### **1.3.4. Vi điều khiển của Hitachi**

H8 là dòng vi điều khiển được phát triển bởi Hitachi, được sản xuất bởi Renesas Technology. H8 gồm các dòng sản phẩm H8/300, H8/300H, H8/500, H8S (vi điều khiển 16 bit), và H8SX (vi điều khiển 32 bit kiểu CISC). Các vi điều khiển họ H8 được sử dụng trong các sản phẩm dân dụng như tivi, đầu DVD, camera,....

### **1.3.5. Vi điều khiển của Motorola**

Motorola sản xuất dòng vi điều khiển 68xx như 6801, 6805, 6809, 6811 ... Một sản phẩm tiêu biểu của Motorola là 68HC11, đây là một bộ vi điều khiển 8 bit; 16 bit địa chỉ; tập lệnh tương thích với các phiên bản trước như 6801, 6805, 6809; có tích hợp bộ biến đổi A/D, bộ tạo xung PWM, cổng truyền thông đồng bộ/không đồng bộ RS232, SPI.

### **1.3.6. Vi điều khiển của Maxim**

Các sản phẩm do Maxim cung cấp gồm:

- Vi điều khiển MAXQ 16 bit kiến trúc RISC như MAXQ3212, MAXQ2000.

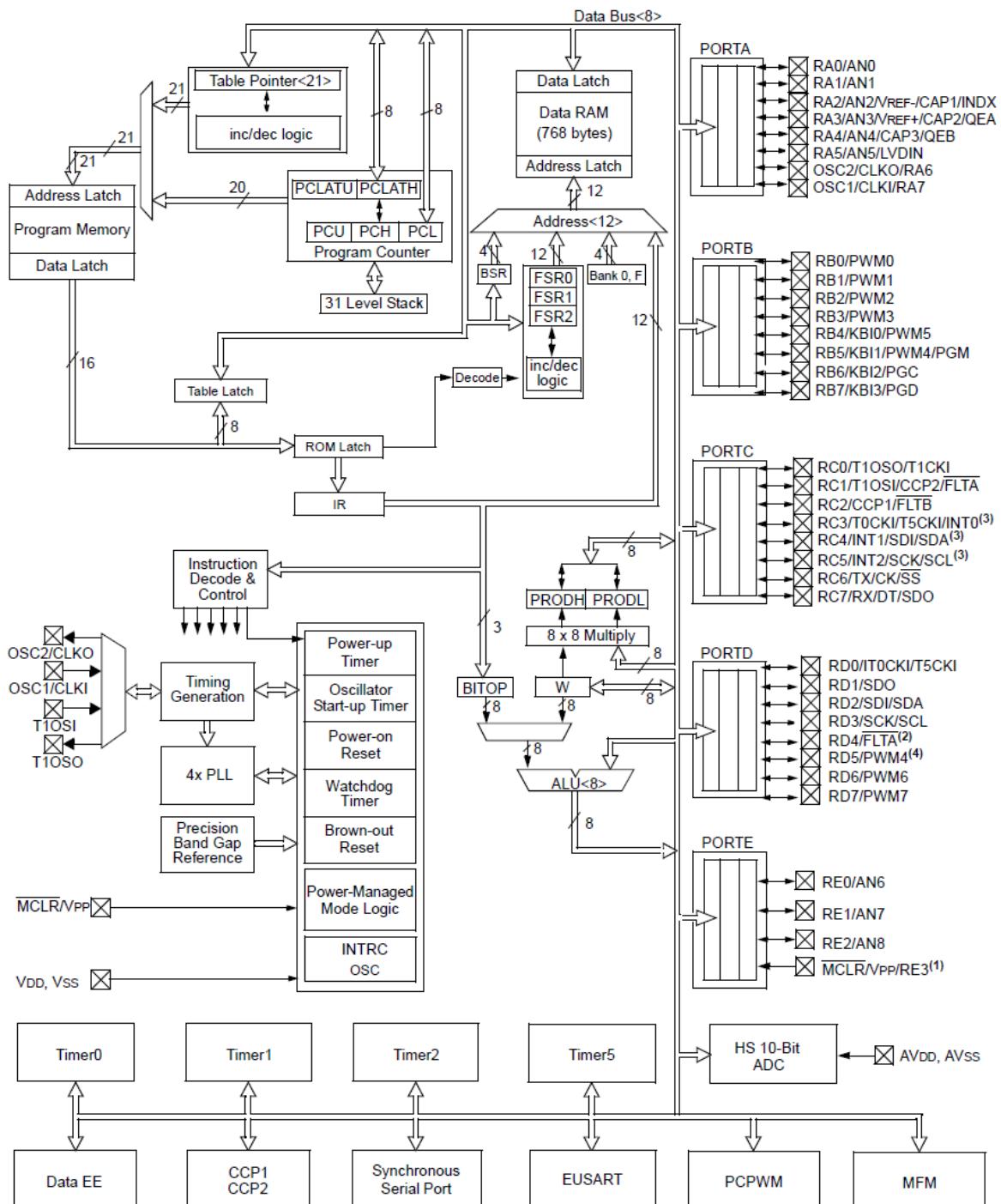
- Các sản phẩm dựa trên kiến trúc của 8051 của Intel như vi điều khiển tích hợp đồng hồ thời gian thực DS87C530, vi điều khiển tích hợp bộ biến đổi A/D 10 bit DS80CH11, vi điều khiển tích hợp giao tiếp mạng Ethernet DS80C400, DS80C430 (rất phù hợp thiết kế IP camera, các trạm đo/điều khiển phân tán như DS5250, DS2250, DS2252, ...).

## 1.4. VI ĐIỀU KHIỂN PIC18F4431

### 1.4.1. Cấu trúc phần cứng

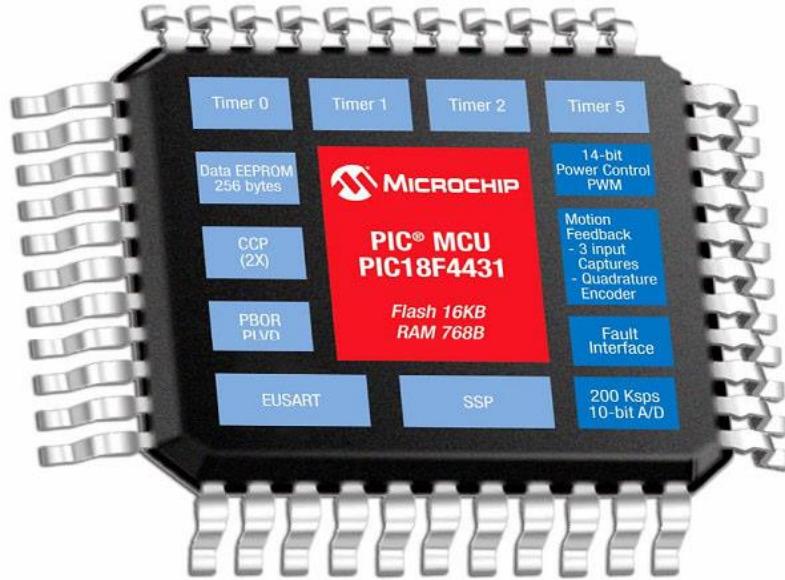
#### a. Sơ đồ khái niệm phần cứng

Vi điều khiển PIC18F4431 thuộc dòng vi điều khiển PIC 8 bit. Có cấu trúc như sau:



Hình 1.4.. Sơ đồ khái niệm của PIC18F4431

Các khối chính trên PIC 18F4431 bao gồm:



*Hình 1.4b.. Sơ đồ khối của PIC18F4431*

- **Bộ xử lý trung tâm CPU (Central Processing Unit)**

- **Bộ nhớ (Memory)**

- Bộ nhớ chương trình (Program Memory) bao gồm 16384 bytes kiểu Flash.
- Bộ nhớ dữ liệu (Data Memory) bao gồm 768 byte SRAM (Static Random Access Memory), 256 byte EEPROM.

Bộ nhớ dữ liệu SRAM được chia thành 2 vùng chức năng riêng biệt, vùng RAM đa dụng GPR (General Purpose Registers) sử dụng để chứa dữ liệu, vùng các thanh ghi chức năng đặc biệt SFR (Special Function Registers) chứa các thanh ghi chức năng điều khiển ngoại vi và CPU.

Bộ nhớ dữ liệu EEPROM là bộ nhớ mảng không bị mất dữ liệu khi mất điện, độc lập với bộ nhớ chương trình và bộ nhớ dữ liệu RAM, được sử dụng để lưu trữ dữ liệu lâu dài. Nó có thể ghi/đọc được 1.000.000 lần, dữ liệu có thể lưu trữ trong bộ nhớ 100 năm. Điều khiển và đọc/ghi bộ nhớ dữ liệu EEPROM không truy cập trực tiếp vào tệp thanh ghi hay khoảng trống bộ nhớ chương trình mà được truy cập, điều khiển gián tiếp qua các thanh ghi chức năng đặc biệt SFR.

- **Bộ phát xung hệ thống (Oscillator):** Nguồn xung từ bên ngoài hoặc từ bộ phát xung của hệ thống sẽ được đi qua bộ nhân hoặc chia tần số để lựa chọn lấy tần số thích hợp để làm xung hệ thống.

- Nguồn xung chính được đưa vào chip qua chân OSC1 và OSC2, tần cao nhất đạt 40MHz.
- Nguồn xung phụ được đưa vào chip qua các chân T1OSI, T1OSO.
- Bộ phát xung nội tần số 31kHz tới 8 MHz.

- **Watchdog Timer(WDT):** WDT là một bộ timer có chức năng đặc biệt. Nếu được “cho phép” WDT sẽ hoạt động và khi tràn sẽ khởi động lại hệ thống. Thời gian khởi động lại hệ thống có thể lựa chọn được từ 4ms đến 131,072s. WDT sẽ được khởi tạo ở đầu chương trình, trong thân chương trình sẽ được “chèn” các lệnh reset WDT sao cho khi MC thực hiện các đúng tuân tự các lệnh, WDT chưa bị tràn. Mục đích chính của việc sử dụng WDT là tránh cho vi điều khiển vô tình thực hiện phải một vòng lặp chết (dead loop) mà không thoát ra được. Khi đó do không thực hiện được các lệnh reset WDT nên MC sẽ tràn, tự động reset lại hệ thống, thoát khỏi tình trạng “bị treo” trong vòng lặp chết. Ngoài ra do có thể hoạt động trong khi MC “ngủ” (Sleep Mode) nên WDT còn được sử dụng trong các ứng dụng tiết kiệm năng lượng.

- **Bộ nạp chương trình:** Bộ nạp chương trình nối tiếp trên chip (Single-Supply In-Circuit Serial Programming) sẽ giúp nạp chương trình từ mạch nạp vào bộ nhớ ROM qua các chân PGM, PGC và PGD.

- **Bộ Debugger (In-Circuit Debugger):** Mạch Debugger trên chíp sẽ giúp người lập trình kiểm soát lỗi chương trình bằng cách cho vi điều khiển hoạt động ở chế độ chạy từng lệnh, nhóm lệnh hay toàn bộ chương trình.

- **Khối phát hiện tín hiệu reset:** Mạch phát hiện tín hiệu reset có khả năng phát hiện 03 nguồn reset:

- Reset từ chân MCLR.
- Reset khi bật nguồn (POR: Power-on Reset).
- Reset khi nguồn yếu (BOR : Brown-out Reset).

- **Khối quản lý lỗi bộ phát xung (Fail-Safe Clock Monitor):** Khối này được sử dụng để quản lý an toàn bộ phát xung hệ thống.

- **Khối định thời khởi động bộ phát xung (Oscillator Start-up Timer):** Khối này sử dụng để tạo thời gian trễ chờ cho bộ phát xung ổn định.

- **Thiết bị ngoại vi (Peripheral):** PIC18F4431 được tích hợp các thiết bị ngoại vi sau:

- Bộ phát hiện điện áp cao/thấp HLVD(High/low-Voltage Detect).
- Bộ nhớ lưu dữ liệu khi tắt nguồn EEPROM.
- 04 bộ đếm, định thời 16 bit: Timer0, Timer1, Timer2 và Timer3.
- 01 bộ so sánh tín hiệu tương tự (Comparator).
- 09 kênh biến đổi tương tự - số (ADC) độ phân giải 10 bit.
- 02 bộ CCP1, CCP2 (Capture, Compare, PWM : Chụp, So sánh, xung PWM);
- 01 bộ ECCP (Enhanced CCP).
- 08 kênh “Power Control PWM Module” 14 bits

- 01 khối “Motion Feedback Module”
- 01 cổng truyền thông nối tiếp đồng bộ (Master Synchronous Serial Port) có thể hoạt động được ở chế độ SPI hoặc I2C.
- 01 cổng truyền thông nối tiếp đồng bộ/không đồng bộ EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmitter), giúp vi điều khiển PIC có thể giao tiếp với nhau hoặc giao tiếp với cổng COM của máy tính.

- 34 nguồn ngắt

*Bảng 1.1. Cấu trúc vi điều khiển PIC 18F4431*

Features	PIC18F2331	PIC18F2431	PIC18F4331	PIC18F4431
Operating Frequency	DC – 40 MHz			
Program Memory (Bytes)	8192	16384	8192	16384
Program Memory (Instructions)	4096	8192	4096	8192
Data Memory (Bytes)	768	768	768	768
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	22	22	34	34
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM modules	2	2	2	2
14-Bit Power Control PWM	(6 Channels)	(6 Channels)	(8 Channels)	(8 Channels)
Motion Feedback Module (Input Capture/Quadrature Encoder Interface)	1 QEI or 3x IC	1 QEI or 3x IC	1 QEI or 3x IC	1 QEI or 3x IC
Serial Communications	SSP, Enhanced USART	SSP, Enhanced USART	SSP, Enhanced USART	SSP, Enhanced USART
10-Bit High-Speed Analog-to-Digital Converter module	5 Input Channels	5 Input Channels	9 Input Channels	9 Input Channels
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT			
Programmable Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin SPDIP 28-pin SOIC 28-pin QFN	28-pin SPDIP 28-pin SOIC 28-pin QFN	40-pin PDIP 44-pin TQFP 44-pin QFN	40-pin PDIP 44-pin TQFP 44-pin QFN

#### • Khối giao tiếp vào/ra số:

Vi điều khiển PIC18F4520 có 5 cổng vào/ra A, B, C, D và E. Mỗi cổng có một thanh ghi đệm dữ liệu tương ứng là PORTA, PORTB, PORTC, PORTD và PORTE, các thanh ghi này được định địa chỉ theo byte và theo bit.

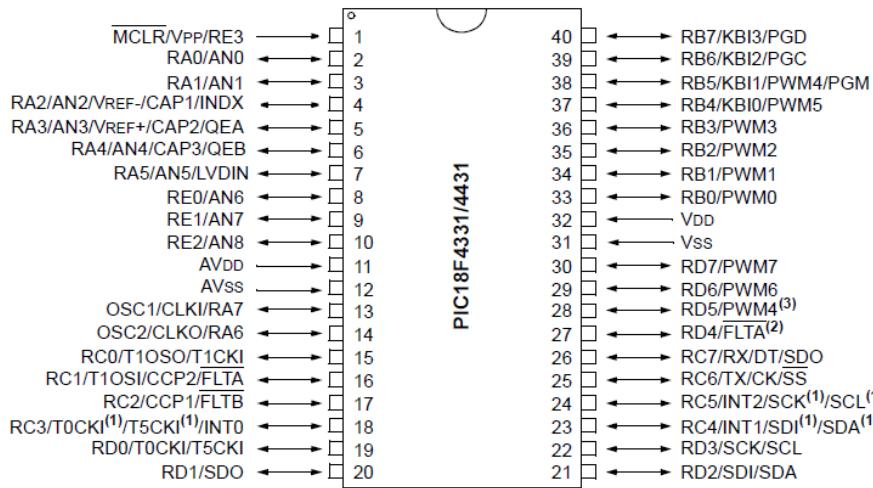
- PORTA : RA7 - RA0.
- PORTB : RB7 - RB0.
- PORTC : RC7 - RC0.

- PORTD : RD3 - RD0.
- PORTE : RE3-RE0.

Bảng 1.1 trình bày tóm tắt cấu trúc các dòng vi điều khiển cùng họ với 18F4431.

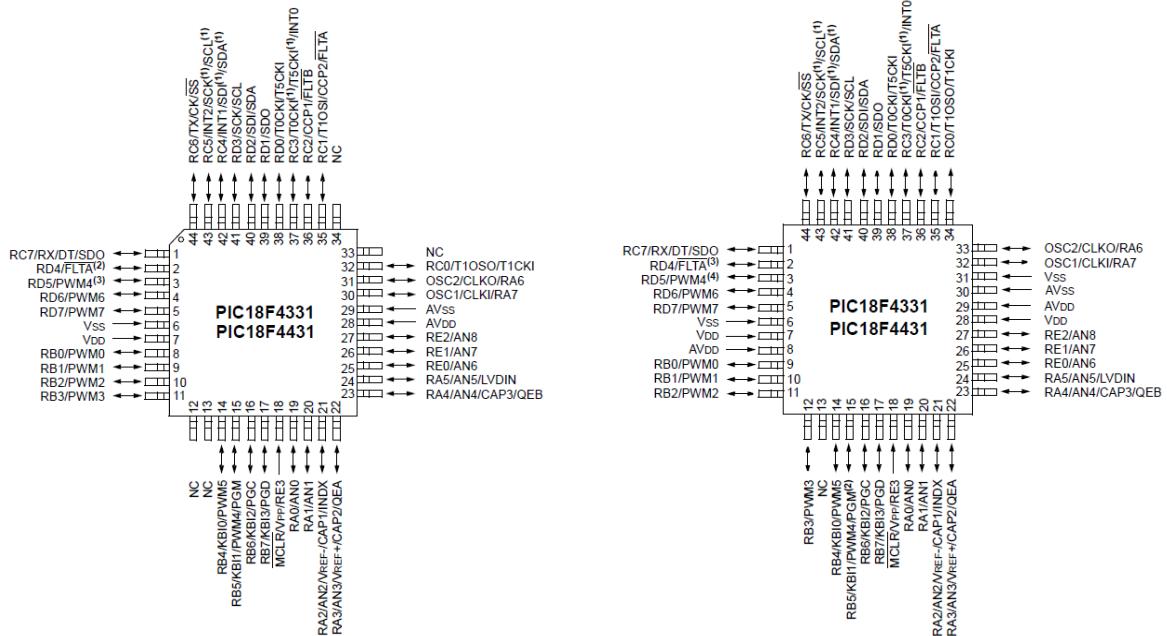
### b. Sơ đồ chân

- Sơ đồ chân dạng PDIP (Lead Plastic Dual In-Line Package) hai hàng chân cắm hai bên.



Hình 1.5. Sơ đồ chân PIC 18F4431 dạng PDIP

- TQFP (Thin Quad Flat Package) bốn hàng chân dán vỏ mỏng hình 1.6. Sơ đồ chân dạng QFN(Quad Flat No-lead) bốn hàng chân dán dưới đế như hình 1.7.

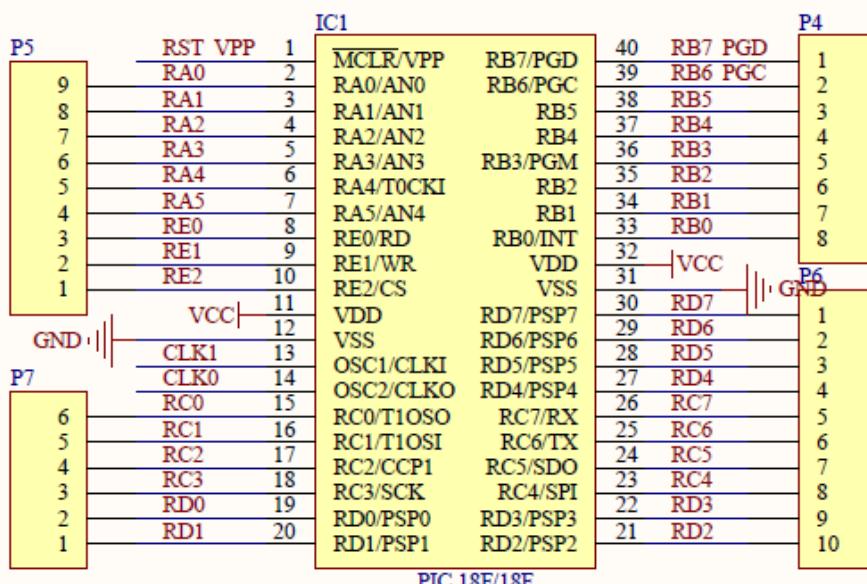


Hình 1.6. Sơ đồ chân PIC 18F4431 dạng TQFP

Hình 1.7. Sơ đồ chân PIC 18F4431 dạng QFN

### 1.3.2. Thiết kế phần cứng

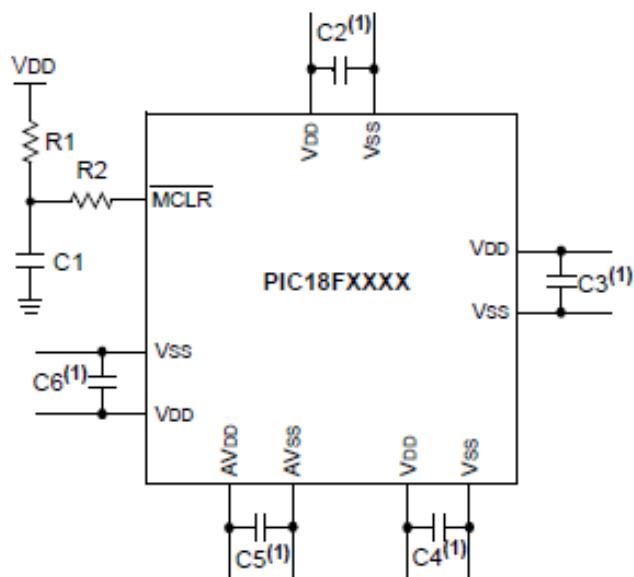
Để thuận tiện cho việc ký hiệu và thiết kế mạch dưới đây, ta có sơ đồ chân và ký hiệu vi điều khiển PIC18F4431 như hình 1.8.



Phải có các tụ decoupling  $0,1\mu F$  đặt gần các chân nguồn (khi vẽ PCB phải đặt càng gần càng tốt): Mặc dù dòng điện tiêu thụ trung bình có thể nhỏ nhưng khi hoạt động ở tần số cao, dòng điện tức thời mà chip cần (để nạp các cổng của các MOSFET) là rất lớn. Do  $di/dt$  lớn nên ảnh hưởng của điện cảm trên đường mạch cũng rất lớn.

Ngoài ra các ồn áp tuyén tính đều không thể cung cấp được dòng điện này cho chip. Vì vậy ta phải cần các tụ điện để tích điện tạm thời trong chu kỳ chip không hoạt động và xả dòng để cung cấp cho chip trong chu kỳ hoạt động. Các tụ điện này phải: ESR thấp để có khả năng xả dòng lớn. Tụ gồm  $0,1\mu F$  là phù hợp. Đôi khi có thể mắc song song thêm một tụ  $0,01\mu F$  thật gần 2 chân nguồn của chip để giảm điện cảm, điện trở của đường mạch. Khi lắp các tụ decoupling, trên các đường cấp nguồn sẽ có nhiều tần số cao do nạp và xả các tụ này cộng thêm nhiều từ các nguồn khác và cần phải triệt tiêu. Nhưng đây là vấn đề rất phức tạp: Sử dụng LDO tốt, dòng tĩnh thấp, nếu phải xài 7805, với một loại chỉ cần có một tụ điện  $0.1\mu F$  sát ngay chân output để tránh bị dao động.

Tăng kích thước đường nguồn để giảm điện cảm. Đặt rải rác các tụ  $10\mu F$  (low ESR) trên các đường cấp nguồn. Nếu phải dùng các tải cảm như role, motor, và dùng chung nguồn nên đi 2 đường mạch VSS – VDD riêng. Dùng diode schottky (1N4148, 1N5817, ...) thay vì các loại nắn dòng (1N4001, 1N4007, ...) để dập dòng cảm ứng trong các tải cảm. Hình 1.10 mô tả mạch cấp nguồn cho vi điều khiển PIC18F. Trong đó C1 tới C6 có giá trị  $0.1 \mu F/20V$ ,  $R1 \leq 10 k\Omega$ ,  $R2 = 100\Omega \div 470\Omega$ .

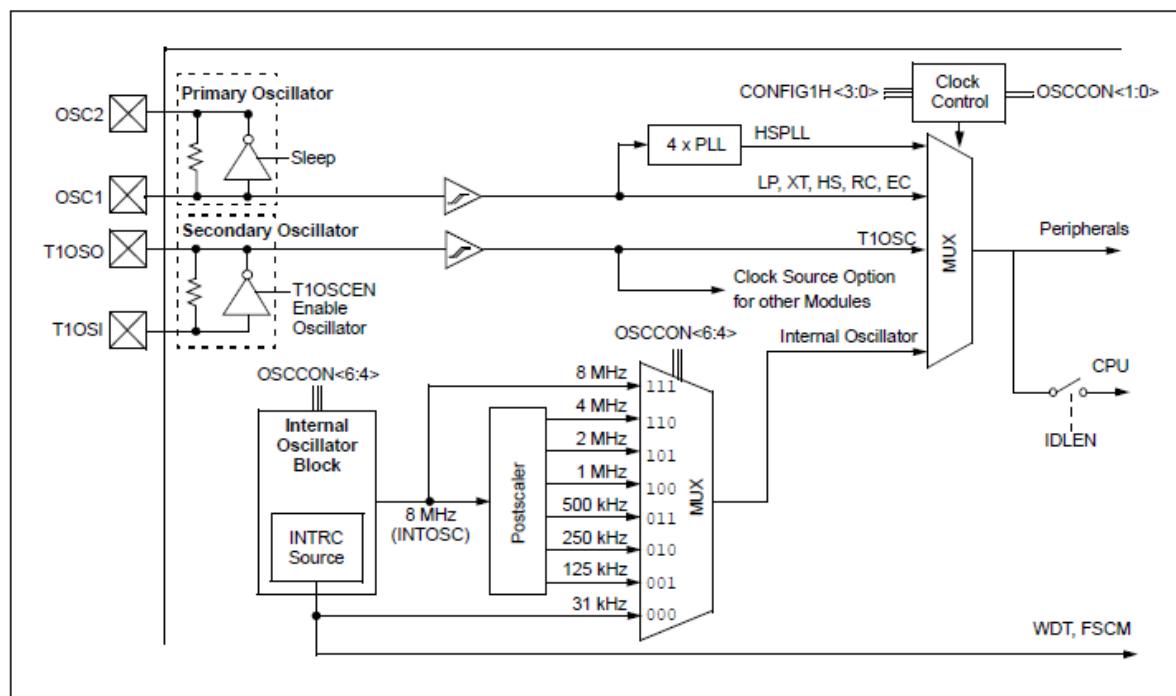


Hình 1.10. Cáp nguồn cho vi điều khiển

### 1.3.2.2. Khởi tạo dao động

PIC 18F4431 có thể hoạt động ở một trong 10 chế độ tạo dao động khác nhau. Việc lựa chọn các chế độ tạo dao động nhờ cấu hình các bit FOSC<3:0>, trong thanh ghi CONFIG1H. Các chế độ tạo dao động:

1. LP (Low-Power Crystal) nguồn xung thạch anh ngoài, nguồn thấp
2. XT (Crystal/Resonator) thạch anh/bộ cộng hưởng bên ngoài
3. HS (High-Speed Crystal/Resonator) thạch anh/bộ cộng hưởng bên ngoài tốc độ cao
4. HSPLL nhân 4 lần tần số HS bằng vòng khóa pha (Phase Locked Loop)
5. RC (External Resistor/Capacitor) tạo dao động bằng mạch RC bên ngoài, phát xung FOSC/4 ra chân RA6.
6. RCIO tạo dao động bằng mạch RC ngoài, vào/ra trên chân RA6.
7. INTIO1 (Internal Oscillator) bộ tạo dao động nội, phát xung FOSC/4 ra chân RA6, vào/ra trên chân RA7.
8. INTIO2 bộ tạo dao động nội, vào/ra trên chân RA6 và RA7.
9. EC (External Clock) bộ phát xung ngoài, phát xung FOSC/4 ra chân RA6.
10. ECIO bộ phát xung ngoài, vào/ra trên chân RA6.

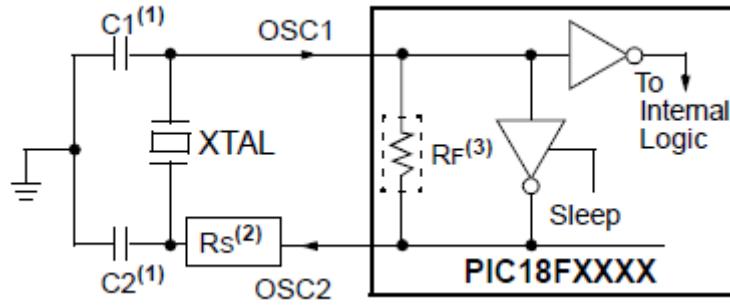


Hình 1.11. Sơ đồ khái bô tạo dao động trên PIC 18F4431

#### ➤ **Tạo dao động bằng thạch anh ngoài (Crystal/ Ceramic Resonator)**

Trong chế độ tạo dao động LP, XT, HS, HSPLL sử dụng thạch anh (Crystal) chưa có tụ điện hoặc mạch cộng hưởng thạch anh bọc gốm đã có tụ điện (Ceramic Resonator). Ở các chế độ này bộ tạo dao động kết nối với vi điều khiển PIC 18F4431 qua hai chân OSC1 và OSC2 .

- Sơ đồ kết nối giữa vi điều khiển với bộ phát xung ngoài:



Hình 1.12. Sơ đồ kết nối với bộ dao động thạch anh/mạch cộng hưởng ngoài

- Lựa chọn giá trị tụ điện khi sử dụng mạch cộng hưởng thạch anh bọc gốm (Ceramic Resonator).

Mode	Freq	OSC1	OSC2
XT	455 kHz	56 pF	56 pF
	2.0 MHz	47 pF	47 pF
	4.0 MHz	33 pF	33 pF
HS	8.0 MHz	27 pF	27 pF
	16.0 MHz	22 pF	22 pF

Bảng 1.2. Lựa chọn giá trị tụ điện khi sử dụng Ceramic Resonator

- Lựa chọn giá trị tụ điện khi sử dụng thạch anh chưa có tụ điện (Crystal).

Osc Type	Crystal Freq	Typical Capacitor Values Tested:	
		C1	C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	1 MHz	33 pF	33 pF
	4 MHz	27 pF	27 pF
HS	4 MHz	27 pF	27 pF
	8 MHz	22 pF	22 pF
	20 MHz	15 pF	15 pF

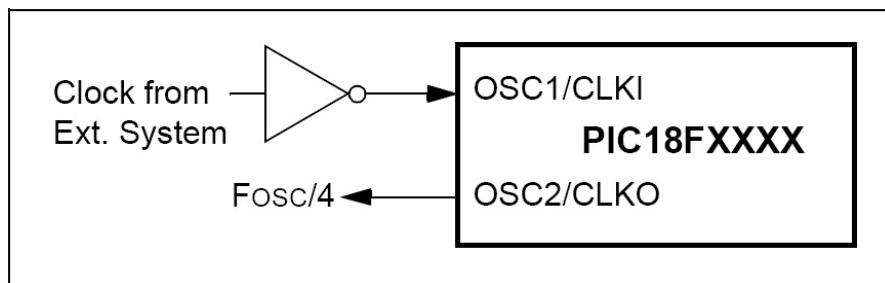
Bảng 1.3. Lựa chọn tụ điện khi sử dụng thạch anh (Crystal)

**Chú ý:** Đặt thạch anh sát gần với PIC: Khi thiết kế PCB, nên đặt thạch anh gần với PIC, nhất là khoảng cách từ chân OSC1 đến chân thạch anh phải càng gần càng tốt, dao động đi vào chân này, càng xa càng nhiều gây mất ổn định cho chip.

#### ➤ Nguồn xung ngoài (External Clock)

Chế độ EC và ECIO sử dụng nguồn xung ngoài làm xung hệ thống và được nối qua cổng NOT trước khi đưa vào chân OSC1/CLKI.

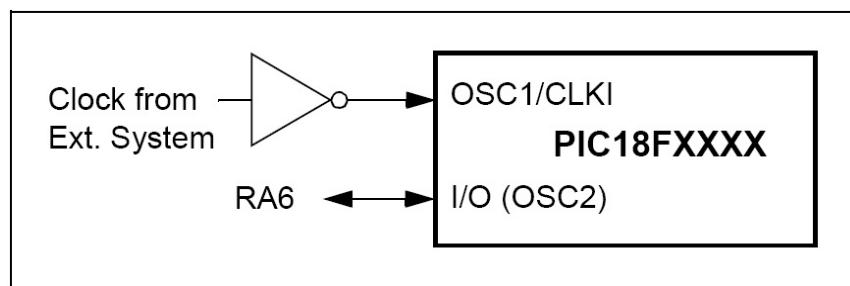
- Chế độ nguồn xung ngoài EC (External Clock), nguồn xung được lấy từ bên ngoài nối qua cổng NOT trước khi đưa vào OSC1/CLKI và chân OSC2/CLKO phát ra tần số bằng  $\frac{1}{4}$  tần số đầu vào.



Hình 1.13. Chế độ dao động EC

➤ **Chế độ nguồn xung ngoài ECIO (External Clock Input Output).**

Nguồn xung được lấy từ bên ngoài nối qua cổng NOT trước khi đưa vào OSC1/CLKI và chân OSC2/CLKO là chân vào/ra RA6.

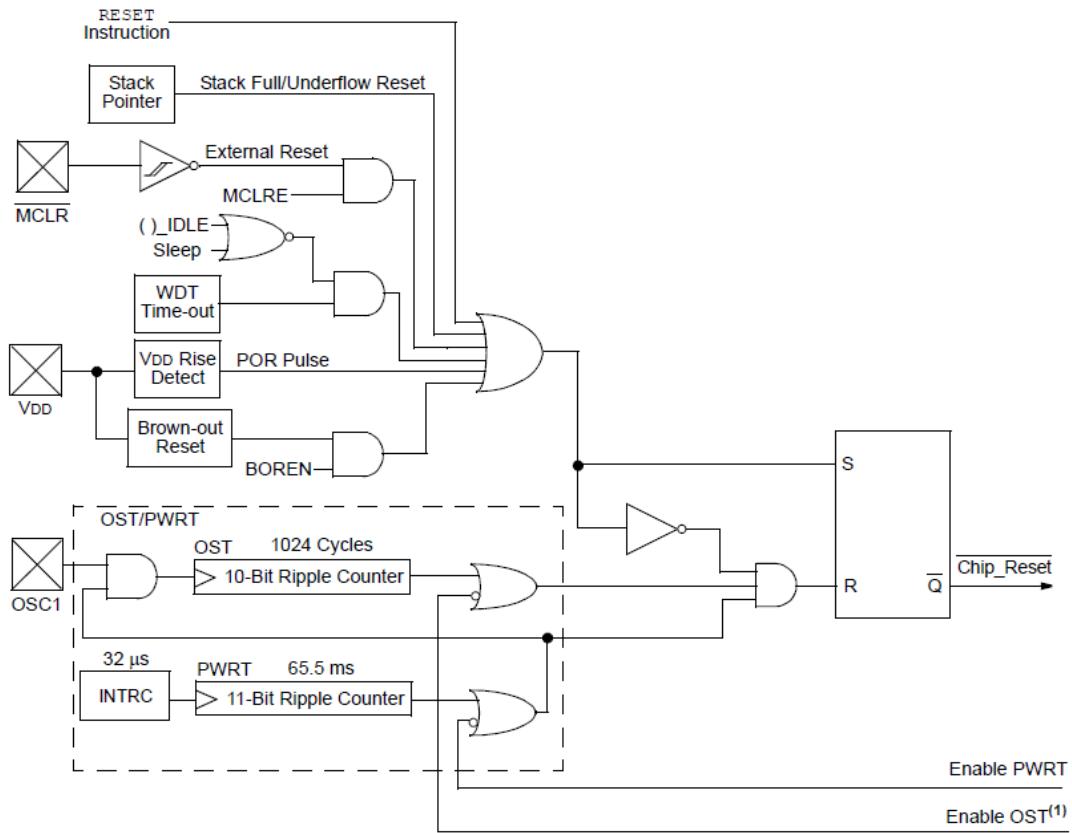


Hình 1.14. Chế độ dao động ECIO

### 1.3.2.3. Hoạt động Reset

Vì điều khiển PIC 18F4431 có 8 nguồn Reset:

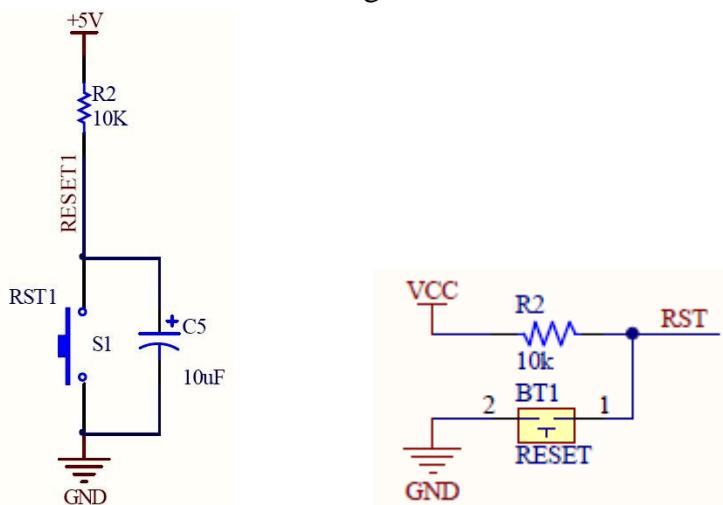
1. Reset do bật nguồn POR (Power-on Reset).
2. Reset từ chân MCLR, sử dụng trong quá trình hoạt động bình thường.
3. Reset từ chân MCLR, sử dụng trong chế độ quản lý nguồn.
4. Reset do Watchdog Timer (WDT).
5. Reset do sụt điện áp nguồn BOR (Brown-out Reset), reset này có thể lập trình được.
6. Reset bằng lệnh RESET.
7. Reset do đầy ngăn xếp (Stack Full Reset).
8. Reset do rỗng ngăn xếp (Stack Underflow Reset).



Hình 1.15. Các chế độ reset vi điều khiển 18F4431

#### ➤ Reset từ chân MCLR

Nguồn reset MCLR được nối từ mạch reset bên ngoài qua chân MCLR/RE3. Nếu bit MCLRE trong thanh ghi CONFIG3H được đặt bằng 1 thì chân MCLR/RE3 là chân reset, đặt MCLRE = “0” chân MCLR/RE3 là chân vào/ra của PORTE. Khi có mức điện áp thấp được đặt lên chân MCLR hệ thống sẽ thực hiện reset.

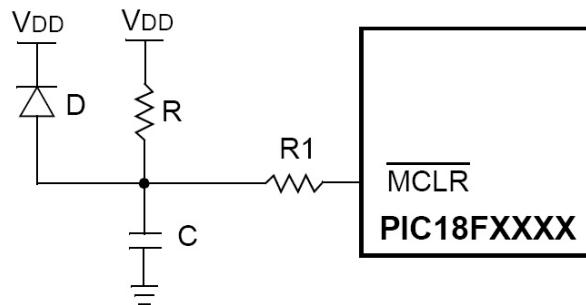


Hình 1.16. Mạch reset ngoài MCLR

### ➤ Reset do bật nguồn POR (Power-on Reset)

Khi nguồn được cấp vào chân VDD, khôi reset POR sẽ phát hiện sùm dương trên chân VDD sau đó phát tín hiệu reset vi điều khiển. Bit POR trong thanh ghi RCON sẽ báo trạng thái của reset POR, POR = '1' là không phát hiện tín hiệu reset, POR = '0' là phát hiện tín hiệu reset POR, bit POR cần được đặt bằng '1' sau reset POR (không được thiết lập bằng phần cứng).

Giá trị linh kiện:  $R < 40 \text{ k}\Omega$ ;  $R1 \geq 1\text{K}\Omega$ .

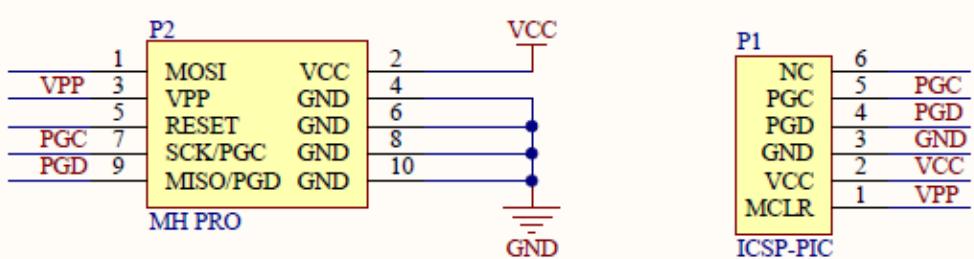


Hình 1.17. Mạch reset do bật nguồn POR.

#### 1.3.2.4. Cổng nạp chương trình

PIC 18F4431 hỗ trợ chuẩn nạp ICSP (In Circuit Serial Programming), nạp trực tiếp cho vi điều khiển PIC kể cả khi PIC gắn trên mạch. Chuẩn nạp này sử dụng chân 1 (VPP), chân 39 (PGC) và chân 40 (PGD) của PIC 18F4431.

Cổng nạp chuẩn ICSP (6 chân) và chuẩn nạp MH PRO (10 chân), minh họa như hình 1.18. Cổng nạp ICSP chỉ cần 5 chân từ 1 đến 5 là có thể nạp được PIC, tuy nhiên có thêm chân số 6 để đề phòng trường hợp nếu như cắm ngược, lúc đó chân MCRL/VPP sẽ đưa vào chân NC, khi thiết bị nạp chân chip, nó sẽ nâng điện áp chân MCRL/VPP lên từ 8-13V, nếu chân MCRL/VPP nối vào một I/O bất kỳ, có thể gây hỏng. Bus nạp cho chuẩn ICSP này ta có thể chọn loại Bus có quy định chiều cắm thì càng tốt (như chuẩn nạp MH PRO).



Hình 1.18. Chuẩn nạp cho PIC

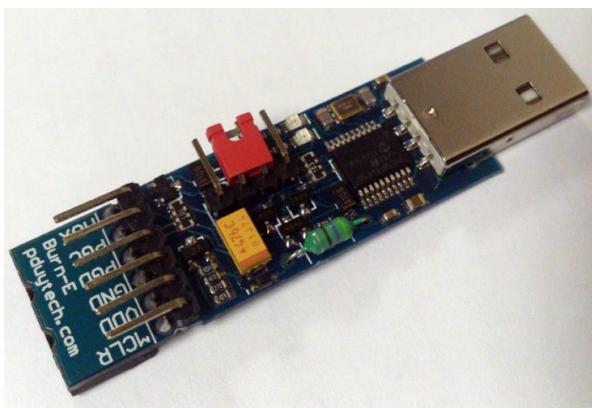
Trên thị trường cung cấp nhiều loại mạch nạp cho PIC, hình 1.19 mô tả một số loại mạch nạp cho PIC.



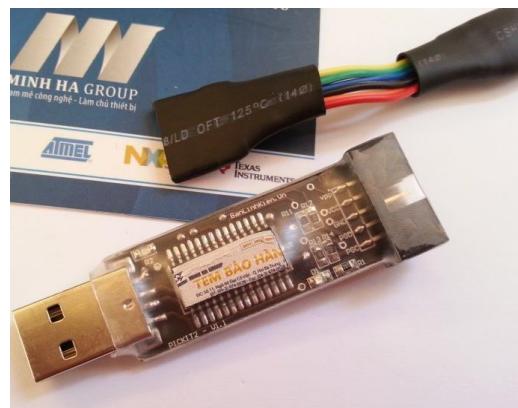
### a) Mạch nạp PIC KIT2



b) Mạch nạp PIC KIT 3



c) Mạch nạp Burn-E



d) Mạch nạp PIC KIT2 MH

Hình 1.19. Một số mạch nạp cho PIC

### 1.3.2.5. Các tính năng đặc biệt, các bit cấu hình của PIC 18F4431

PIC18F4431 có một loạt các tính năng nhằm tối ưu hóa độ tin cậy của hệ thống, giảm thiểu chi phí qua việc loại bỏ các linh kiện bên ngoài, các tính năng bảo vệ mã, tiết kiệm năng lượng. Những tính năng đó là: Reset, Power-on Reser (POR), Power-up timer (PWRT), Oscillator Start – up Timer (OST), Brown – out Reset (BOR), Interrupts, Watchdog timer (WDT), Oscillator selection, Sleep, Code protection, ID Locations, In-Circut Serial Programming, Low-voltage In-Circut Serial Programming, ....

Một vài tính năng trên sẽ luôn luôn sẵn có, nhưng một vài tính năng khác thì cần phải cấu hình trong lúc nạp chương trình cho PIC. Cấu hình là việc bật/tắt các bit trong các thanh ghi cấu hình CONFIGx của PIC, thanh ghi này nằm trong bộ nhớ chương trình nên có độ rộng bằng độ rộng của từ lệnh (14 bits) và được ghi bằng bộ nạp. Các tính năng đó cũng là tính năng cần thiết để PIC có thể hoạt động, do đó trước khi lập trình cần xác định PIC cần những tính năng gì để tiến hành cấu hình cho nó.

Bảng 1.4. Các thanh ghi cấu hình PIC18F4431

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default/ Unprogrammed Value
300000h	CONFIG1L	—	—	—	—	—	—	—	-----
300001h	CONFIG1H	IESO	FCMEN	—	—	FOSC3	FOSC2	FOSC1	FOSC0 11-- 1111
300002h	CONFIG2L	—	—	—	—	BORV1	BORV0	BOREN	PWRREN ----- 1111
300003h	CONFIG2H	—	—	WINEN	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN --11 1111
300004h	CONFIG3L	—	—	T1OSCMX	HPOL	LPOL	PWMPIN	—	— --11 11--
300005h	CONFIG3H	MCLRE <sup>(1)</sup>	—	—	EXCLKMX <sup>(1)</sup>	PWM4MX <sup>(1)</sup>	SSPMX <sup>(1)</sup>	—	FLTAMX <sup>(1)</sup> 1--- 11-1
300006h	CONFIG4L	DEBUG	—	—	—	—	LVP	—	STVREN 1--- -1-1
300007h	CONFIG4H	—	—	—	—	—	—	—	-----
300008h	CONFIG5L	—	—	—	—	CP3 <sup>(1)</sup>	CP2 <sup>(1)</sup>	CP1	CP0 ----- 1111
300009h	CONFIG5H	CPD	CPB	—	—	—	—	—	11-- -----
30000Ah	CONFIG6L	—	—	—	—	WRT3 <sup>(1)</sup>	WRT2 <sup>(1)</sup>	WRT1	WRT0 ----- 1111
30000Bh	CONFIG6H	WRTD	WRTB	WRTC	—	—	—	—	111- -----
30000Ch	CONFIG7L	—	—	—	—	EBTR3 <sup>(1)</sup>	EBTR2 <sup>(1)</sup>	EBTR1	EBTR0 ----- 1111
30000Dh	CONFIG7H	—	EBTRB	—	—	—	—	—	-1-- -----
3FFFFEh	DEVID1 <sup>(2)</sup>	DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0 xxxx xxxx <sup>(2)</sup>
3FFFFFh	DEVID2 <sup>(2)</sup>	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3 0000 0101

Legend: x = unknown, u = unchanged, - = unimplemented. Shaded cells are unimplemented, read as '0'.

Bảng 1.4 mô tả các thanh ghi cấu hình của PIC18F4431, dưới đây sẽ trình bày một số bit cấu hình tiêu biểu.

**IESO (Internal External Switch Over):** Bit này tích cực mức cao, dùng để cấu hình cho chế độ khởi động 2 tốc độ. Đây là tính năng giúp giảm thiểu năng lượng tiêu hao trên PIC do quá trình khởi động gây ra bằng cách giảm thiểu độ trễ dao động bên ngoài và quá trình thực thi mã. Với những ứng dụng thường sử dụng như SLEEP để tiết kiệm năng lượng, TSS thực sự hiệu quả khi mà PIC đi vào SLEEP và thức dậy từ SLEEP một cách nhanh chóng. TSS sẽ sử dụng dao động nội trong quá trình khởi động, ngay sau khi dao động bên ngoài ổn định, PIC sẽ tự chuyển sử dụng dao động bên ngoài.

**FCMEN (Fail safe Clock Monitor Enable):** Tích cực mắc cao, tính năng này cho phép PIC tiếp tục hoạt động nếu nguồn dao động bên ngoài bị hỏng. FSCM có thể dùng với các nguồn dao động ngoài như: LP, XT, HS, EC, RC và RCIO (tham khảo phần dao động). Khi dao động bên ngoài bị hỏng, FSCM sẽ chuyển dao động dùng cho PIC từ từ nguồn dao động bên ngoài sang nguồn dao động nội, đồng thời bật cờ OSFIF trong thanh ghi PIR2, sẽ gây ngắt nếu OSFIE trong thanh ghi PIE2 được cho phép. Dao động nội được dùng cho tới khi nào nguồn dao động bên ngoài được phục hồi và phần mềm trong PIC chuyển sang dao động ngoài.

Dao động nội được FSCM chọn phụ vào vào 3 bits IRCF<2:0> trong thanh ghi OSCCON, nghĩa là FSCM cho phép cấu hình dao động nội trước khi FSCM nhận biết sự hỏng hóc của dao động bên ngoài.

Khi phần mềm PIC thay đổi nguồn dao động, tức là bật/tắt bit SCS, thì coi như FSCM khởi động lại từ đầu, lúc này bit OSFIF mới được phép xóa. Ngoài ra, sau khi PIC khởi động từ RESET hay SLEEP, FSCM cũng khởi động.

**FOSC (Oscillator Selection):** Chọn chế độ dao động chính của PIC. Khi lựa chọn chế độ dao động, cần phải tìm hiểu các chế độ dao động của PIC hoạt động như thế nào.

**BOREN (Brown – out Reset Enable):** Tích cực mức cao, đây là tính năng Reset của PIC khi điện áp nhỏ hơn một mức nào đó đã được định nghĩa ( $V_{BOR}$ ).

**PWRTE (Power-up Timer Enable):** Bit này tích cực mức thấp. Tính năng này cho phép giữ PIC reset trong khoảng 64ms từ khi cấp nguồn, hay bị reset bởi Brown-out.

**WDTEN (Watchdog Timer Enable):** Tích cực mức cao khi được cho phép, Timer Watchdog sẽ chạy, và sẽ gây reset chip mỗi khi tràn. Nếu không được cho phép thì WDT sẽ bị tắt, đồng thời được điều khiển bằng phần mềm trong PIC thông qua thanh ghi WDTCON.

**MCLRE (Master Clear Enable):** Đây là tính năng cho phép PIC có thể bị reset bằng chân MCLR. Khi bit MCLRE (tích cực mức cao) được cho phép, thì chân RE3/MCLR có thể reset PIC khi được cấp mức 0, và khi PIC hoạt động phải cấp mức logic 1. Khi bit MCLRE không được cho phép, thì chân MCLR không có tính năng reset PIC, lúc này chân RE3/MCLR chỉ hoạt động như một ngõ vào.

**DEBUG:** Tích cực mức thấp, khi bit này bị xóa (0) chip sẽ hoạt động ở chế độ gõ rõi, chạy từng bước, truyền thông tin bộ nhớ về thiết bị gõ rõi. Chân RB6/CLK và RB7/DAT hoạt động với chức năng gõ rõi. Ngược lại, khi bit này được bật lên 1, chip không thực hiện tính năng Debug, thay vào đó RB6/CLK và RB7/DAT hoạt động như một I/O thông thường. Chú ý là bít này có thể xóa hay bật bởi MPLAB khi chọn chế độ nạp hay gõ rõi trên MPLAB, do vậy trong quá trình làm việc không nhất thiết quan tâm đến bit này.

**LVB (Low Voltage Programming):** Tính năng lập trình điện áp thấp, nghĩa là nạp chip với điện áp làm việc của nó (5V), khi tính này được cho phép (bit được bật lên 1), có thể nạp chip thông qua ICSP với điện áp nguồn của nó. Khi thực hiện tính năng này, chân RB3/PGM phải được nối lên VDD, điều này có nghĩa không thể thực hiện RB3/PGM như một I/O, nó phục vụ cho tính năng LVB và chân MCPR/VPP phải cấp điện áp VDD. Chú ý: Tính năng lập trình điện áp cao luôn có, với tính năng này, PIC luôn được có thể nạp thông qua cổng ICSP khi điện áp chân MCLR/VPP nâng lên khoảng 13V (tùy loại PIC, có loại chỉ có 8V). Thông thường tính năng này ít sử dụng và được tắt để tận dụng chân RB3 như I/O.

**CP (Code Protect):** Bảo vệ toàn bộ bộ nhớ chương trình, tích cực mức thấp. Khi được xóa về 0, không thể đọc lại nội dung từ bộ nhớ chương trình của PIC. Chú ý: Trong quá trình gỡ rối, CP không cho phép, MPLAB sẽ cảnh báo.

**CPD (Code Protect Data):** Bit này tích cực mức thấp, khi cấu hình bằng 0, cho phép bảo vệ dữ liệu trong EEPROM, sẽ không thể đọc lại dữ liệu EEPROM từ PIC bằng bất cứ chương trình với sự hỗ trợ của thiết bị nào.

## CÂU HỎI HƯỚNG DẪN ÔN TẬP, THẢO LUẬN

**Câu 1:** Trình bày cấu trúc chung của vi điều khiển? Vẽ hình minh họa.

**Câu 2:** Trình bày các kiểu kiến trúc của vi điều khiển? Vi điều khiển PIC18F4431 thuộc kiểu kiến trúc nào?

**Câu 3:** Trình bày cấu trúc, sơ đồ các khôi của vi điều khiển PIC18F4431? Vẽ hình minh họa.

**Câu 4:** Trình bày các chế độ tạo dao động của PIC18F4431? Vẽ hình nguyên lý chế độ tạo dao động bằng thạch anh ngoài?

**Câu 5:** Trình bày chức năng khôi Reset của vi điều khiển và các nguồn reset của PIC18F4431? Vẽ hình nguyên lý minh họa mạch reset ngoài MCLR?

## CHƯƠNG 2

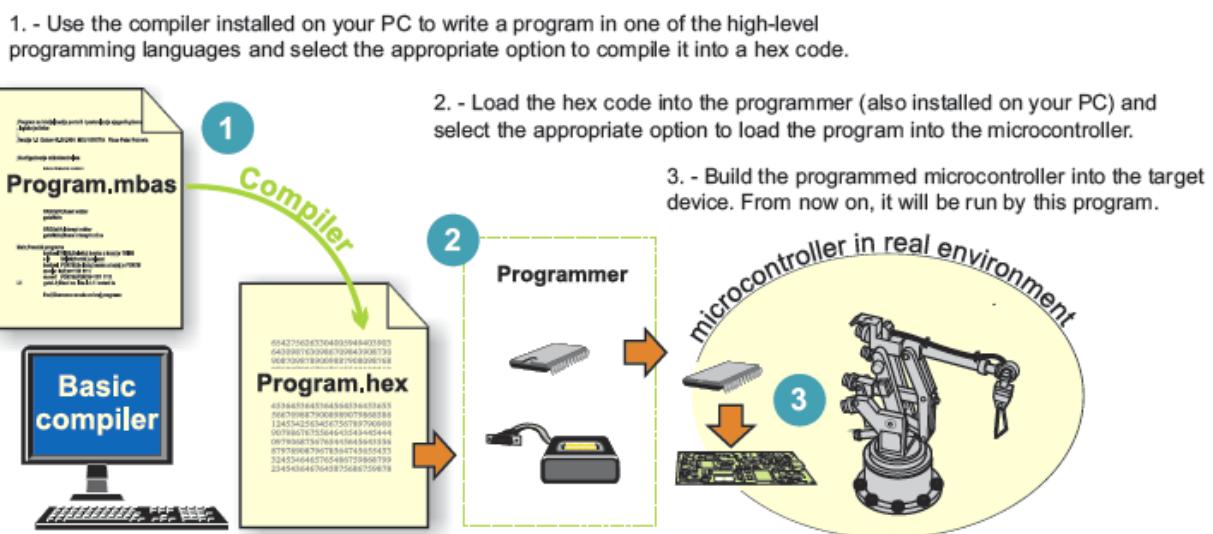
### CÁC TÀI NGUYÊN CƠ BẢN CỦA VI ĐIỀU KHIỂN PIC18F

#### MỤC TIÊU CỦA CHƯƠNG

Nội dung chương 2 yêu cầu sinh viên nắm được các phần mềm lập trình (CCS, MikroC, MPLAB) và các tài nguyên cơ bản của vi điều khiển PIC18F: Hoạt động ngắn của vi điều khiển, hoạt động vào/ra, bộ định thời Timer, khói CCP, bộ biến đổi ADC và bộ truyền thông nối tiếp trong vi điều khiển PIC18F

#### 2.1. CÁC PHẦN MỀM LẬP TRÌNH

Các bước khi thiết lập và làm việc với các ứng dụng sử dụng vi điều khiển minh họa như hình 2.1.



Hình 2.1. Các bước thực hiện dự án với vi điều khiển

Máy tính nói chung và vi điều khiển nói riêng chỉ xử lý, tính toán trên các chuỗi bit “0”, “1” được sắp xếp theo một trật tự nhất định gọi là “mã máy” (*machine language* hay *machine code*). Để máy tính có thể thực hiện được thuật toán, cần phải viết thuật toán dưới dạng mã máy hoặc các dòng “lệnh” theo các quy ước nào đó để sau đó có thể biên dịch ra mã máy. Tập các kí hiệu và các quy tắc viết các lệnh để thể hiện thuật toán được gọi là một ngôn ngữ lập trình (*programming language*). Các quy tắc để viết chương trình được gọi là cú pháp (*syntax*) của ngôn ngữ lập trình. Có rất nhiều ngôn ngữ lập trình khác nhau, tuy nhiên căn cứ vào mức độ hình thức hoá có thể chia thành ba lớp:

*Lớp 1:* Ngôn ngữ máy. Chương trình trong ngôn ngữ máy là dãy các lệnh máy mà CPU có thể thực hiện trực tiếp. Đó là ngôn ngữ lập trình duy nhất mà máy tính “hiểu được”. Trong thang bậc các ngôn ngữ giao tiếp với máy tính, đây là mức thấp nhất nhưng hiệu quả của chương trình sẽ là cao nhất vì ta có thể khai thác triệt để khả

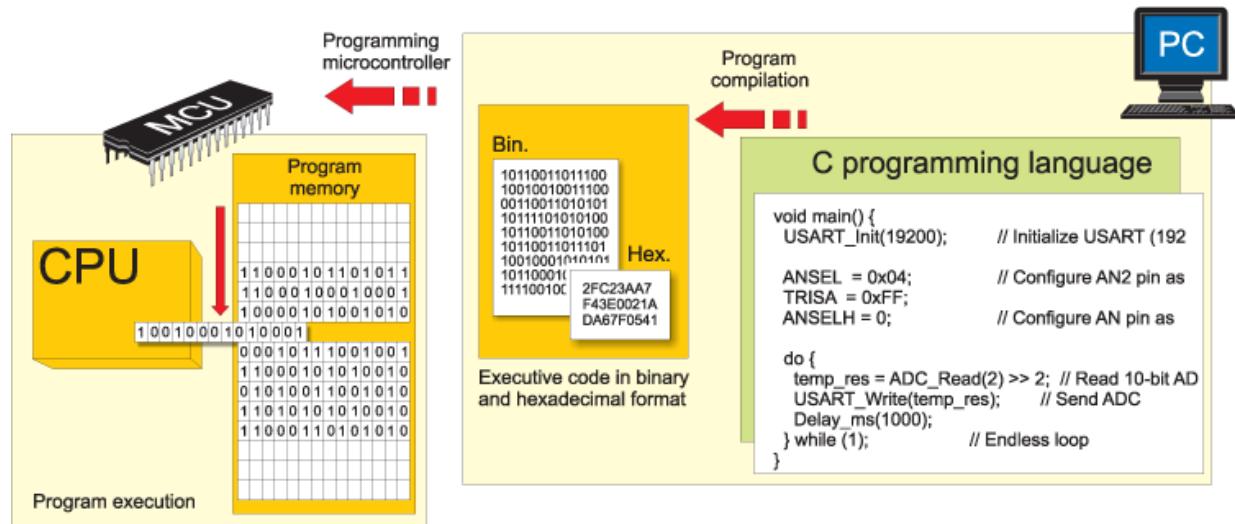
năng của máy. Tuỳ theo thiết kế về phần cứng, mỗi loại máy tính có một ngôn ngữ máy khác nhau. Các lệnh viết bằng ngôn ngữ máy nói chung ở dạng nhị phân hoặc biến thể của chúng trong hệ đếm 16.

*Lớp 2: Hợp ngữ (Assembly).* Hợp ngữ được viết bằng tập hợp các chữ (ký tự), mỗi mã lệnh được thể hiện bằng một mã chữ, địa chỉ các đối tượng trong lệnh cũng được “quy ước” thành các ký tự.

Ví dụ, trong PIC18Fxxx, lệnh hợp ngữ MOVLW 0X0F mang ý nghĩa: Chuyển vào thanh ghi W hằng số 0F(H). Khi chuyển sang mã máy như sau: 0000 1110 0000 1111.

Để một chương trình viết bằng hợp ngữ chạy được trên máy tính cần phải được dịch ra ngôn ngữ máy. Chương trình dịch đổi với hợp ngữ được gọi là hợp dịch (Assembler).

*Lớp 3: Ngôn ngữ thuật toán (Algorithmic language).* Là ngôn ngữ có các câu lệnh gần với ngôn ngữ tự nhiên, ngôn ngữ toán học và độc lập với bất cứ loại máy cụ thể nào. Ngôn ngữ thuật toán còn được gọi là ngôn ngữ lập trình bậc cao (high level programming language). Cũng như đối với hợp ngữ, mỗi ngôn ngữ lập trình bậc cao trên một loại máy cụ thể đều cần có chương trình dịch (còn gọi là trình biên dịch, gọi tắt là *trình dịch*) để dịch các chương trình sang sang ngôn ngữ máy của máy đó mới có thể thực hiện được.



Hình 2.2. Nhiệm vụ của trình biên dịch

Trình dịch (Compiler), là một chương trình cài đặt trên máy tính làm công việc dịch một chuỗi các câu lệnh được viết bằng một ngôn ngữ lập trình (gọi là ngôn ngữ nguồn hay mã nguồn), thành một chương trình tương đương nhưng ở dưới dạng ngôn ngữ máy tính mới (gọi là ngôn ngữ đích) và thường là ngôn ngữ ở cấp thấp hơn, như ngôn ngữ máy. Chương trình mới được dịch này gọi là mã đối tượng.

Đối với vi điều khiển PIC18 cũng có các loại ngôn ngữ lập trình tương ứng với ba lớp trên. Với hợp ngữ, PIC18 có tập lệnh gồm 76 lệnh, đa số các lệnh được mã hóa bằng 16 bit (có 4 lệnh được mã hóa bằng 32 bit). Phần này sẽ khái quát các “Trình biên dịch” dịch từ ngôn ngữ lập trình C cho vi điều khiển PIC. Ứng với mỗi loại trình biên dịch khác nhau cũng sẽ phân tích ưu điểm, nhược điểm và khả năng ứng dụng của trình biên dịch đó. Có thể nói với vi điều khiển PIC, các trình biên dịch phát triển theo 2 hướng sau: Hướng tiện dụng, nhanh chóng và hướng chuyên sâu.

### 2.1.1. Phần mềm MPLAB

MPLAB IDE là phần mềm được hỗ trợ bởi Microchip, dùng để soạn thảo code cho các ứng dụng của PIC. Để lập trình PIC trên phần mềm MPLAB phải cài đặt trình biên dịch C, các trình biên dịch hỗ trợ trên phần mềm MPLAB được trình bày dưới đây.

Hitech bản thân từ đầu là một nhóm phát triển độc lập với Microchip. Với phương pháp tương tác thanh ghi, bit trên thanh ghi được Hi-Tech phát triển trong trình biên dịch Hi-Tech C, điều đó cho phép người lập trình có khả năng tương tác ở cấp độ từng bit trong thanh ghi để cài đặt những tính năng cần thiết cho vi điều khiển. Nhưng để làm được việc này người lập trình không những có kiến thức về kỹ thuật lập trình C mà còn phải hiểu rõ về kiến trúc phần cứng của vi điều khiển. Tuy vậy, người lập trình sẽ hiểu tường tận hoạt động của vi điều khiển thế nào, dễ dàng tối ưu code của họ. Hitech C là trình biên dịch riêng cho dòng PIC16F.

Microchip đã thừa hưởng những ưu điểm của Hi-Tech C và phát triển trình biên dịch C16, C18, C30, C32 (C16 cho dòng vi điều khiển 16F, C18 cho dòng vi điều khiển 18F, C30 cho dòng vi điều khiển DsPIC30F). Cấu trúc chương trình, phương pháp tương tác hoàn toàn giống Hitech C. Điều khác nhau duy nhất là phương pháp CONFIG (tạm hiểu là khai báo phần cứng ban đầu).

Hiện nay Microchip đã ngừng hỗ trợ MPLAB, chỉ có MPLAB IDE X là được Microchip phát triển. Cùng với đó là các trình biên dịch: (Tháng 9/2012 Microchip cho ra đời 3 phiên bản nâng cấp của Hitech C và C18, C30, C32):

- XC8 cho dòng vi điều khiển PIC 8 bit như PIC10, PIC12, PIC16, PIC18.
- XC16 cho dòng vi điều khiển PIC 16 bit như dSPIC30F, dSPIC33F
- XC 32 cho dòng vi điều khiển PIC 32 bit.



Hình 2.3. Phần mềm MPLAB và các trình biên dịch C

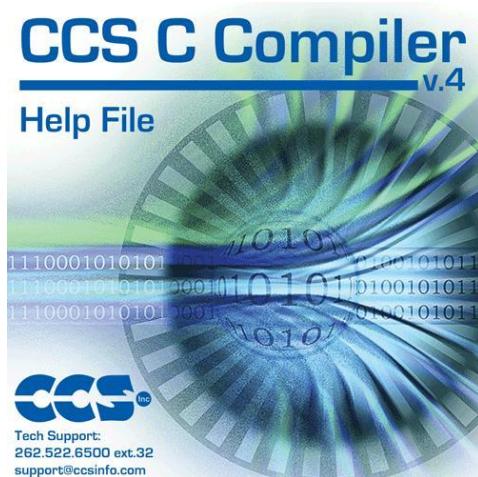
- **Ưu điểm:** Tương tác cấp độ phần cứng, tính mở trong phương pháp lập trình, dễ dàng phát triển, kết nối với các phương pháp lập trình khác. cho phép người lập trình tối ưu hóa về code. Người lập trình có thể tính toán được thời gian hoạt thực thi chương trình một cách chính xác. Rất tương thích với môi trường lập trình MPLAB và vi điều khiển PIC.

- **Nhược điểm:** Đòi hỏi người dùng không những có kiến thức về C, người lập trình còn phải hiểu rõ về cấu trúc máy tính, bộ nhớ...Hỗ trợ ít thư viện, nên người sử dụng phải tự xây dựng các thư viện.

- **Ứng dụng:** Khi muốn tìm hiểu chuyên sâu về vi điều khiển để phát triển tiếp các dòng vi điều khiển khó hơn. Tối ưu hóa code, thời gian thực thi, biên dịch chương trình... Muốn thực hiện một dự án mà yêu cầu khắc khe về thời gian tính toán của chip xử lý.

### 2.1.2. Phần mềm CCS

CCSC là loại trình biên dịch đơn giản cho PIC phổ biến nhất hiện nay. Với ưu điểm dễ sử dụng (người lập trình hầu như không quan tâm tới phần cứng của vi điều khiển ngoại trừ sơ đồ kết nối chân), và ra đời sớm CCSC trở nên quen thuộc đối với những ai đã dùng dòng PIC16F kể từ 2003. Và cho đến ngày nay đây vẫn là một loại trình biên dịch được ưa chuộng.



Hình 2.4. Phần mềm CCS

Có thể khái quát một số ưu, nhược điểm của mảng này như sau:

- **Ưu điểm:** Dễ sử dụng (chỉ cần có kiến thức về C cơ bản), nhiều thư viện hỗ trợ, hỗ trợ nhiều công cụ đi kèm

- **Nhược điểm:** Tính bao đóng của thư viện và các hàm trong thư viện, không thể sửa đổi và phát triển, không thể kết hợp với những cú pháp của trình biên dịch khác. Không tối ưu về code và biên dịch do chương trình được dịch sẽ có rất nhiều đoạn mã không cần thiết. Khó có thể lập trình theo hướng realtime hoặc những ứng dụng yêu cầu khắt khe về thời gian tính toán.

- **Ứng dụng:** Với những ưu điểm, nhược điểm trên ta cũng dễ dàng nhận ra được nên dùng CCSC trong trường hợp nào. Ý kiến của bản thân tôi thì CCSC sẽ rất có ích trong những trường hợp sau:

- + Chưa vững về ngôn ngữ lập trình C
- + Mới tiếp xúc với vi điều khiển lần đầu
- + Muốn thực hiện nhanh một dự án, dĩ nhiên là những dự án, công việc không yêu cầu khắt khe về thời gian tính toán

### 2.1.3. Phần mềm MikroC

Mikro C cũng có những tính chất tương tự CCSC – cùng là hướng tiện dụng và nhanh chóng nhất cho người dùng. Thế mạnh của MikroC là các thư viện hỗ trợ từ nhà sản xuất (tutorial, tài liệu, ứng dụng).



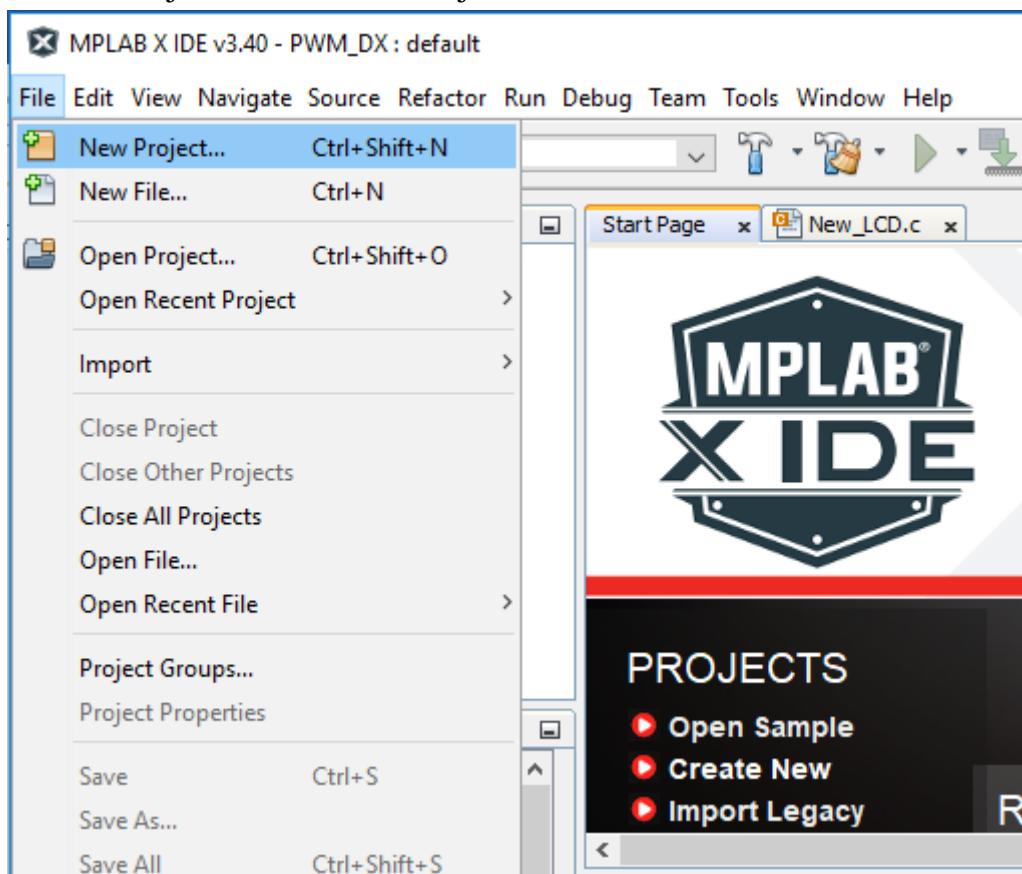
Hình 2.5. Phần mềm MikroC

## 2.2. PHẦN MỀM MPLAB VÀ XC8

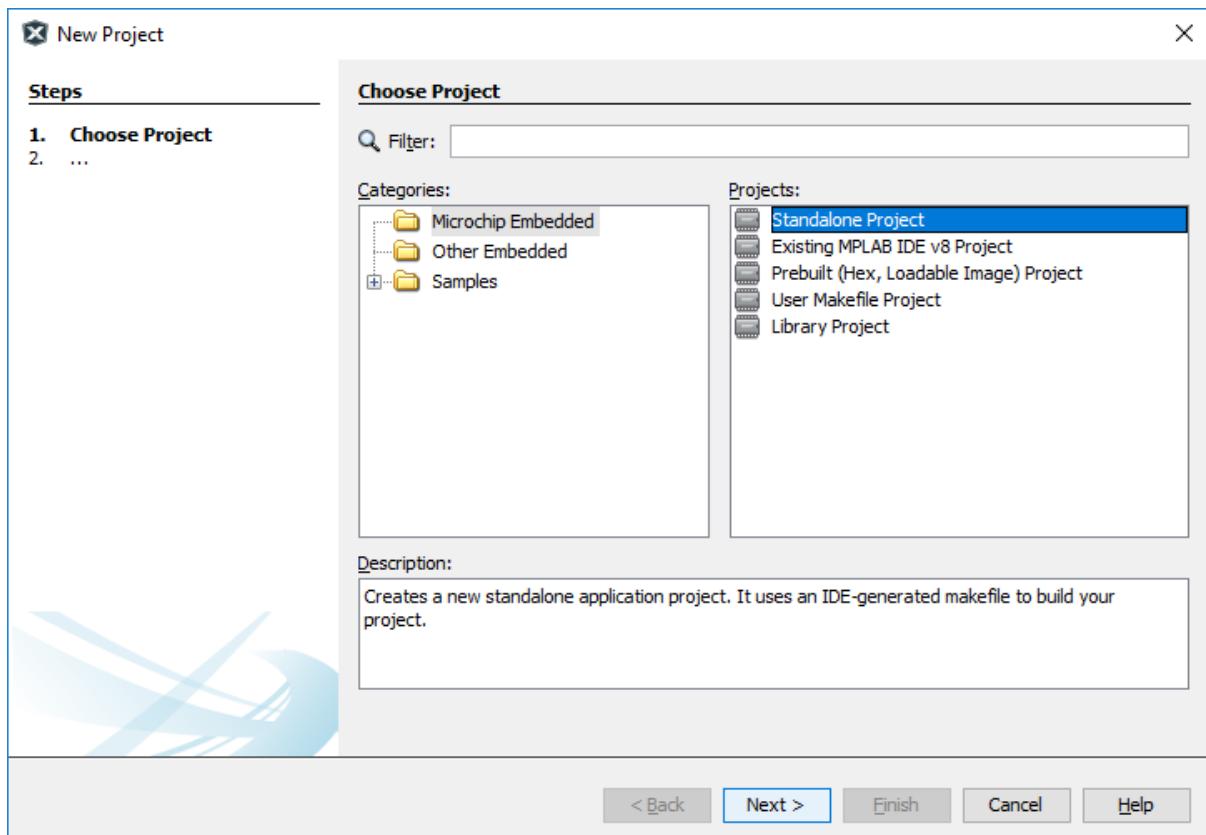
Trong tài liệu này sử dụng phần mềm MPLAB và trình biên dịch C XC8. Phần dưới đây trình bày các tập lệnh ngôn ngữ C trên trình biên dịch XC8.

### 2.2.1. Khởi tạo Project trên phần mềm MPLAB

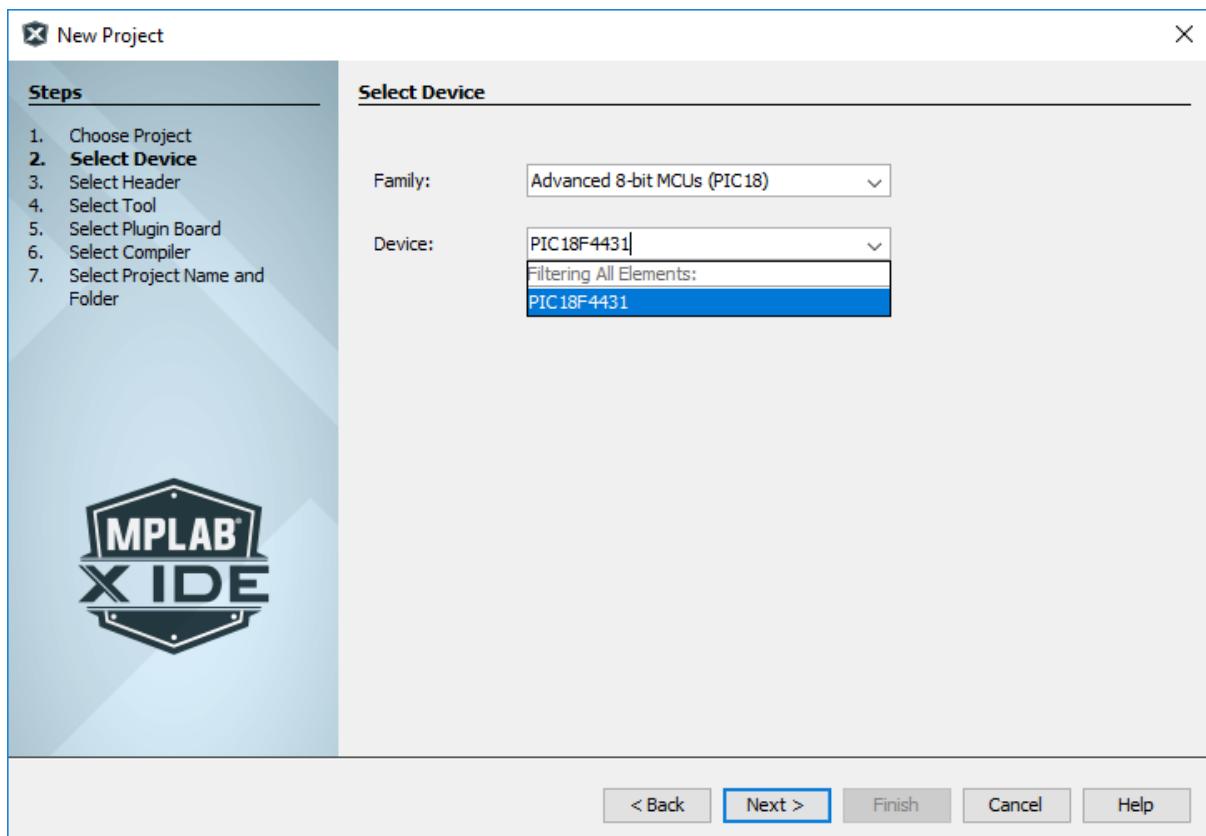
Khởi tạo Project: File → New Project



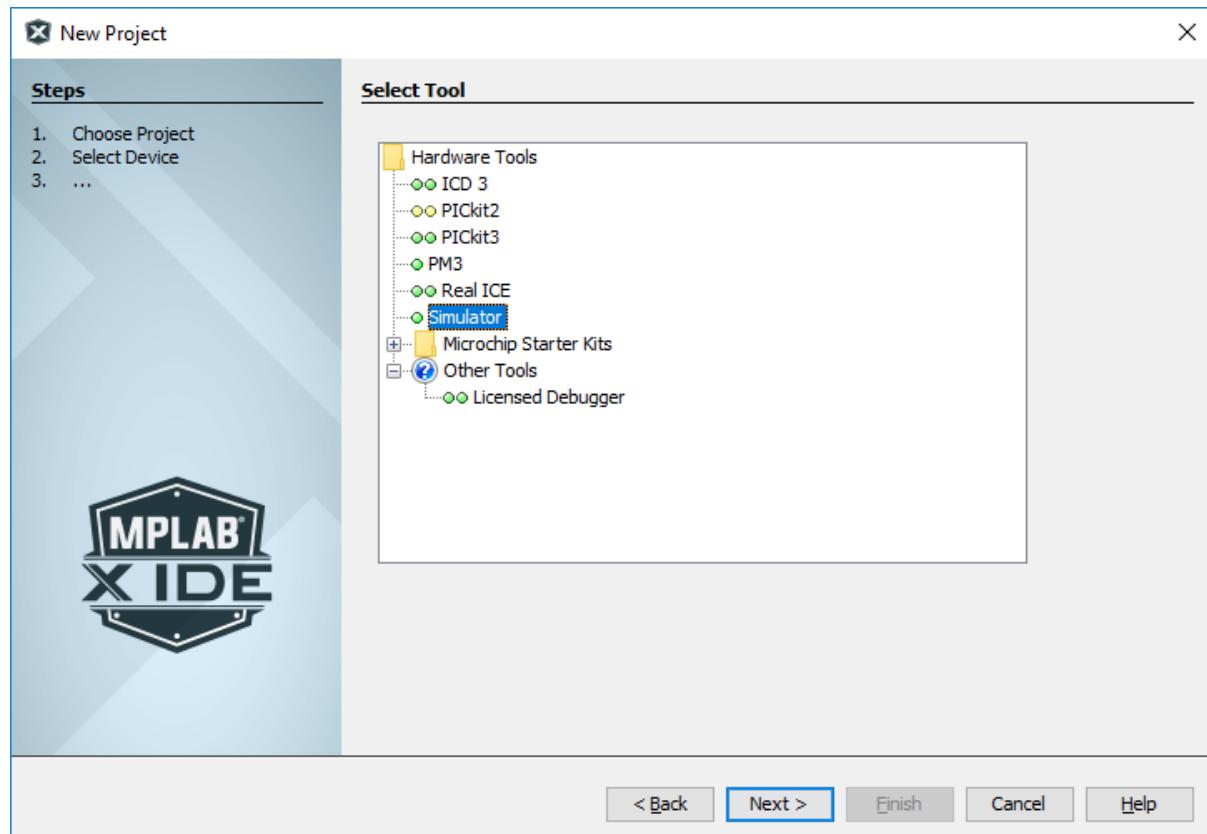
Cửa sổ mới xuất hiện, chọn Standalone Project và chọn Next



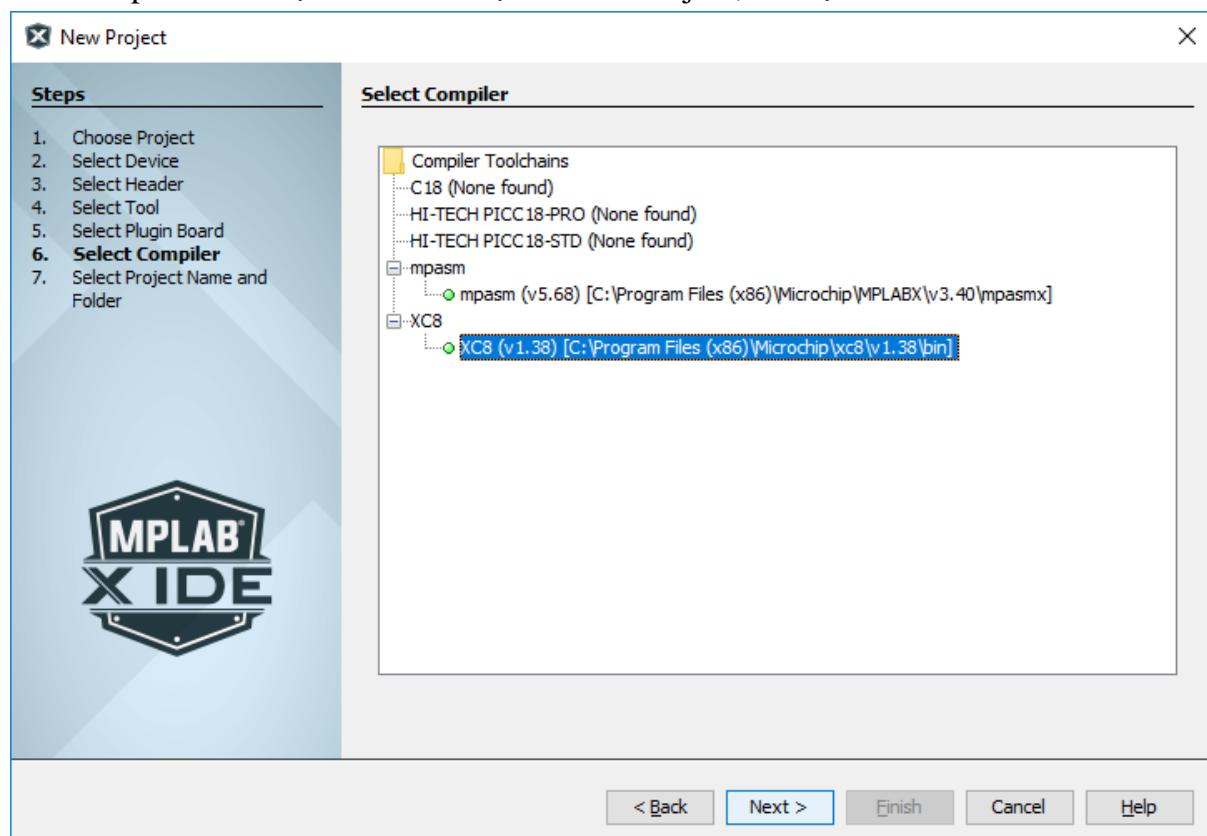
Tiếp theo chọn dòng vi điều khiển 8 bit tại Family và chọn PIC18F4431 tại Device.



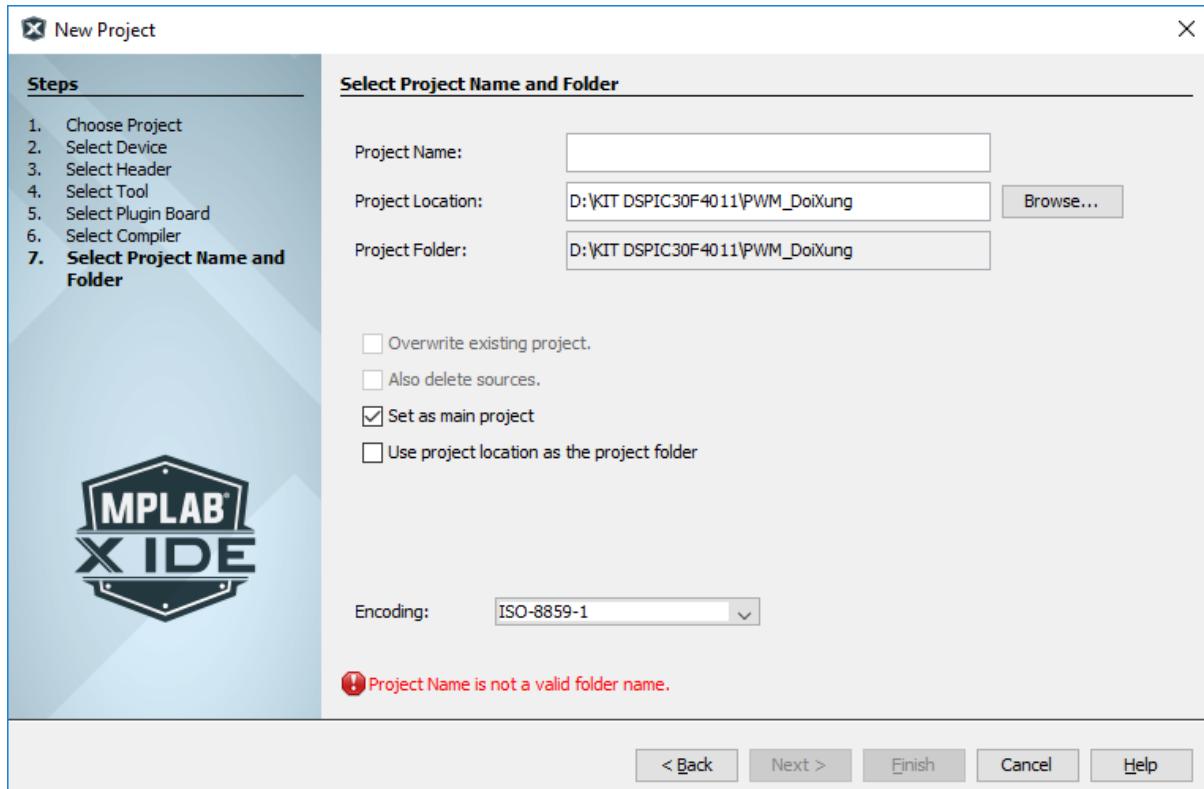
Cửa sổ tiếp theo xuất hiện, yêu cầu chọn phần cứng kèm theo. Ta lựa chọn công cụ mạch nạp hoặc mô phỏng, sau đó nhấn Next.



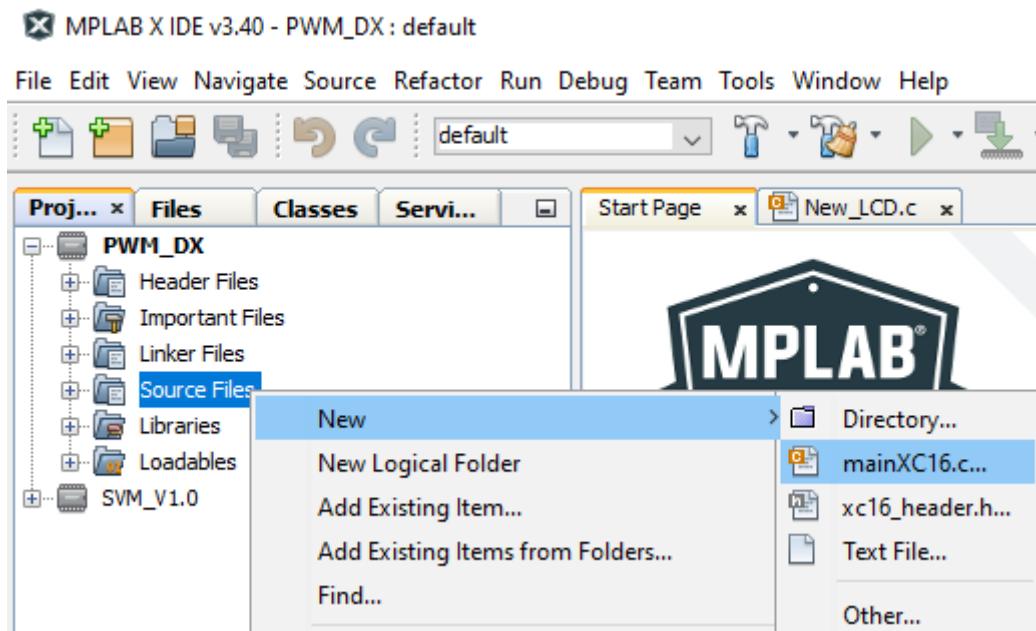
Tiếp theo là chọn trình biên dịch C cho Project, ta chọn XC8.



Tiếp theo là chọn tên dự án và đường dẫn lưu trữ dự án. Ta nhập tên và đường dẫn sau đó chọn Finish để kết thúc quá trình khởi tạo dự án



Sau đó tại cửa sổ phần mềm MPLAB, tạo file C mới như hình dưới. Sau đó viết chương trình các file tạo ra tùy người sử dụng.



## 2.2.2. Ngôn ngữ lập trình C của trình biên dịch XC8

### 2.2.2.1. Khung một chương trình ngôn ngữ C với XC8

Cấu trúc chương trình C bao gồm các khối sau:

1. Khai báo thư viện
2. Cấu hình vi điều khiển
3. Khai báo biến, hằng toàn cục (nếu có)
4. Các chương trình con (nếu có)
5. Chương trình chính
6. Chương trình ngắt (nếu có)
7. Các chú thích (nếu có)

Ví dụ một chương trình C cơ bản như sau:

```
#include <xc.h>

// PIC16F1719 Configuration Bit Settings

// For more on Configuration Bits,
// consult your device data sheet

// CONFIG1
#pragma config FOSC = ECH      // External Clock, 4-20 MHz
#pragma config WDTE = OFF       // Watchdog Timer (WDT) disabled
#pragma config PWRTE = OFF      // Power-up Timer disabled
#pragma config MCLRE = ON       // MCLR/VPP pin function is MCLR
#pragma config CP = OFF         // Flash Memory Code Protection off
#pragma config BOREN = ON        // Brown-out Reset enabled
#pragma config CLKOUTEN = OFF    // Clock Out disabled.
#pragma config IESO = ON         // Internal/External Switchover on
#pragma config FCMEN = ON        // Fail-Safe Clock Monitor enabled

// CONFIG2
#pragma config WRT = OFF        // Flash Memory Self-Write Protect off
#pragma config PPS1WAY = ON       // PPS one-way control enabled
#pragma config ZCDDIS = ON        // Zero-cross detect disabled
#pragma config PLLEN = OFF       // Phase Lock Loop disable
#pragma config STVREN = ON        // Stack Over/Underflow Reset enabled
#pragma config BORV = LO          // Brown-out Reset low trip point
#pragma config LPBOR = OFF        // Low-Power Brown Out Reset disabled
#pragma config LVP = OFF          // Low-Voltage Programming disabled

#define LEDS_ON_OFF 0x55

void main(void) {

    // Port D access

    ANSELD = 0x0; // set to digital I/O (not analog)
    TRISD = 0x0; // set all port bits to be output
    LATD = LEDS_ON_OFF; // write to port latch - RD[0:3] = LED[0:3]

    // Port B access
    ANSELB = 0x0; // set to digital I/O (not analog)
    TRISB = 0x0; // set all port bits to be output
    LATB = LEDS_ON_OFF; // write to port latch - RB[0:3] = LED[4:7]

    return;
}
```

**Chú ý:** Các câu lệnh trong C kết thúc bằng dấu “;”.

Các lời chú thích được đặt trong dấu “/\*” kết thúc bằng “\*/”. Nếu lời giải thích trên một dòng thì có thể dùng dấu: “//”

Khi lập trình nên giải thích các câu lệnh khỏi lệnh làm gì để về sau khi chương trình lớn dễ sửa lỗi.

### ➤ **Chỉ thị tiền xử lý**

Các chỉ thị tiền xử lý không phải là các lệnh của ngôn ngữ C mà là các lệnh giúp cho việc soạn thảo chương trình nguồn C trước khi biên dịch. Khi dịch một chương trình C thì không phải chính bản chương trình nguồn mà ta soạn thảo được dịch. Trước khi dịch, các lệnh tiền xử lý sẽ chỉnh lý bản gốc, sau đó bản chỉnh lý này sẽ được dịch. Có ba cách chỉnh lý được dùng là:

- + Phép thay thế, định nghĩa: **#define**
- + Phép chèn tệp: **#include**

Các chỉ thị tiền xử lý giúp ta viết chương trình ngắn gọn hơn và tổ chức biên dịch, gỡ rối chương trình linh hoạt, hiệu quả hơn.

\* Chỉ thị **#define**: Chỉ thị #define cho phép tạo các macro thay thế đơn giản.

Định nghĩa: #define Tên Địa Chỉ;

Ví dụ:

```
#define RS_PIN LATEbits.LATE0
```

\* Chỉ thị **#include**: Chỉ thị #include báo cho trình biên dịch nhận nội dung của tệp khác và chèn vào tệp chương trình nguồn mà ta soạn thảo.

- Cú pháp:

Cách 1: #include<tên\_tệp>

Cách 2: #include“tên\_tệp”

### **2.2.2. Biến và khai báo biến**

Biến là các vùng nhớ để lưu giá trị khi làm việc trong chương trình. Trong XC8 có các kiểu biến (**kiểu dữ liệu**) biểu diễn số nguyên như sau:

Type	Size (bits)	Arithmetic Type
bit	1	Unsigned integer
signed char	8	Signed integer
unsigned char	8	Unsigned integer
signed short	16	Signed integer
unsigned short	16	Unsigned integer
signed int	16	Signed integer
unsigned int	16	Unsigned integer
signed short long	24	Signed integer
unsigned short long	24	Unsigned integer
signed long	32	Signed integer
unsigned long	32	Unsigned integer
signed long long	32	Signed integer
unsigned long long	32	Unsigned integer

Và trong XC8 có các kiểu biến (**kiểu dữ liệu**) biểu diễn số thực như sau:

Type	Size (bits)	Arithmetic Type
float	24 or 32	Real
double	24 or 32	Real
long double	same as double	Real

### ➤ Khai báo biến

Cấu trúc:      Kiểu\_bien  Tên\_bien

Ví dụ:            char     Bien1;

Khi khai báo biến có thể gán luôn cho biến giá trị ban đầu. VD:

Thay vì: unsigned char x;

      x=0;

Ta chỉ cần: unsigned char x=0;

Có thể khai báo nhiều biến cùng kiểu một lúc. Ví dụ: unsigned int x,y,z;

### 2.2.2.3. Hằng và khai báo hằng

Hằng về cơ bản cũng là một vùng nhớ được cấp cho chương trình, tuy nhiên giá trị của nó không thể thay đổi được sau khi khởi tạo.

Một hằng số thông thường được định nghĩa bởi từ khóa const. Kiểu hằng số và định dạng:

Radix	Format	Example
binary	0b <i>number</i> or 0B <i>number</i>	0b10011010
octal	0 <i>number</i>	0763
decimal	<i>number</i>	129
hexadecimal	0x <i>number</i> or 0X <i>number</i>	0x2F

➤ **Khai báo hằng:**

```
const Kiểu_dữ_liệu Tên_hằng = Giá_trị_hằng;
Kiểu_dữ_liệu Const Tên_hằng = Giá_trị_hằng;
```

Ví dụ: const float pi = 3.14;

#### 2.2.2.4. Mảng và khai báo mảng

Mảng là một tập hợp các biến có cùng tính chất và cùng kiểu dữ liệu. Khai báo mảng:

```
Tên_kiểu_dữ_liệu Tên_mảng[số_phần_tử];
```

Ví dụ1:

```
char tuoi[5]={0,5,10,15,20};
```

Mảng **tuoi** có 5 phần tử được đánh số thứ tự từ 0 đến 5: tuoi[0], tuoi[1],...,tuoi[4].

Để truy xuất giá trị một phần tử trong mảng thì viết: tên\_mảng[chỉ\_số]. Ví dụ: tuoi[0]=0; tuoi[4]=20

Ví dụ 2:

```
unsigned char GiaTri[] ={0x08,0x08,0x00,0x00,0x00,0x09,0x41,0x80,0xC0,0xFF,0x00,0x00,0x13,0x1A,0x26,0x33,0x80,0xFF,0x00,0x00,0x00,0x09,0x41,0x80,0x66,0x66,0x00,0x00,0x00,0x05,0x4A,0x46,0x40,0x40,0x00,0x00,0x00,0x08,0x43,0x43,0x3D,0x3A,0x00,0x00,0x00,0x00,0x2D,0x4D,0x56,0x4D,0x00,0x00,0x00,0x00,0x21,0x56,0x6C,0x6F};
```

#### 2.2.2.5. Khai báo các hàm trong XC8

Hàm trong XC8 có cấu trúc như sau:

Có 2 loại hàm:

➤ **Hàm trả lại giá trị:**

Cấu trúc:

[Kiểu giá trị hàm trả lại] [Tên hàm (Biến truyền vào hàm)]

{

//Các lệnh xử lý ở đây

}

Ví dụ:

```
unsigned char Cong(unsigned char x, unsigned char y)
{
    //Các lệnh xử lý ở đây
}
```

➤ **Hàm không trả lại giá trị**

Cấu trúc:

```
void Tên_hàm (Biến truyền vào hàm)
{
    //Các lệnh xử lý ở đây
}
```

Ví dụ: void Cong(unsigned char x, unsigned char y)

```
{
    // Các câu lệnh xử lý ở đây
}
```

Chú ý: *Hàm có thể truyền vào biến hoặc không*

Ví dụ: Hàm không có biến truyền vào:

```
unsigned char Tên_hàm(void)
{
    //Các câu lệnh xử lý ở đây
}
```

*Hàm có biến truyền vào:*

```
void Tên_hàm(unsigned char x)
{
    // Các câu lệnh xử lý ở đây
}
```

Số biến truyền vào tùy ý (miễn đủ bộ nhớ), ngăn cách bởi dấu “,”

Ví dụ:

```
void Tên_hàm (unsigned char x, unsigned char y, unsigned char z)
{
    //Các câu lệnh xử lý ở đây
}
```

➤ Khai báo hàm chương trình chính trong XC8

```
void main (void){  
    //Các câu lệnh;  
}
```

➤ Khai báo hàm chương trình ngắt trong XC8

```
__interrupt(type) void Tên_Hàm (void) {  
    //Các câu lệnh;  
}
```

<b>&lt;empty&gt;</b>	Implement the default interrupt function.
<b>low_priority</b>	The interrupt function corresponds to the low priority interrupt source. (MPLAB XC8 - PIC18 only)
<b>high_priority</b>	The interrupt function corresponds to the high priority interrupt source. (MPLAB XC8)

Ví dụ:

```
__interrupt(low_priority) void getData(void) {  
    if (TMR0IE && TMR0IF) {  
        TMR0IF=0;  
        ++tick_count;  
    }  
}
```

### 2.2.2.6. Các phép toán cơ bản

#### a. Phép toán số học

Phép toán	Ý nghĩa	Ví dụ
<b>+</b>	Phép cộng	$X = a+b;$
<b>-</b>	Phép trừ	$X = a-b;$
<b>*</b>	Phép nhân	$X = a*b;$
<b>/</b>	Phép chia lấy phần nguyên	$X = a/b;$ $(a=9, b=2 \rightarrow X=4)$
<b>%</b>	Phép chia lấy phần dư	$X = a \% b;$ $(a=9, b=2 \rightarrow X=1)$

#### b. Phép toán Logic

Chức năng	Phép toán
AND	<b>&amp;&amp;</b>
OR	<b>  </b>
NOT	<b>!</b>

### c. Các phép toán so sánh

Phép toán	Ý nghĩa	Ví dụ
>	So sánh lớn hơn	$a>b$ 4>5 sẽ trả ra giá trị 0
$\geq$	So sánh lớn hơn hoặc bằng	$a\geq b$ 6 $\geq$ 2 sẽ trả ra giá trị 1
<	So sánh nhỏ hơn	$a < b$ 6<7 sẽ trả ra giá trị 1
$\leq$	So sánh nhỏ hơn hoặc bằng	$a \leq b$ 8 $\leq$ 5 sẽ trả ra giá trị 0
$=$	So sánh bằng nhau	$a == b$ 6 $=$ 6 sẽ trả ra giá trị 1
$\neq$	So sánh khác nhau	$a \neq b$ 9 $\neq$ 9 sẽ trả ra giá trị 0

### d. Phép toán thao tác Bit

Phép toán	Ý nghĩa	Ví dụ
$\&$	Phép và (AND)	$Bit\_1 \& Bit\_2$
$ $	Phép hoặc (OR)	$Bit\_1   Bit\_2$
!	Phép đảo (NOT)	$!Bit\_1$
$\wedge$	Phép hoặc loại trừ (XOR)	$Bit\_1 \wedge Bit\_2$
$<<$	Dịch trái	$a << 3$
$>>$	Dịch phải	$a >> 4$
$\sim$	Lấy bù theo bit	$\sim a$

### e. Phép toán kết hợp

Phép toán	Ví dụ
$+=$	$a += 5 \Leftrightarrow a = a + 5$
$-=$	$a -= 5 \Leftrightarrow a = a - 5$
$*=$	$a *= 5 \Leftrightarrow a = a * 5$
$/=$	$a /= 5 \Leftrightarrow a = a / 5$
$\%=$	$a \%= 5 \Leftrightarrow a = a \% 5$

#### 2.2.2.7. Các cấu trúc điều khiển, rẽ nhánh

Trong phạm vi tài liệu, chỉ đề cập tới một số cấu trúc cơ bản, các cấu trúc khác trong ngôn ngữ lập trình C cũng hoàn toàn có giá trị trong trình dịch C (ngoại trừ kiểu file)

##### ➤ Cấu trúc While

while (biểu thức điều kiện)

{

Các câu lệnh;

}

Chương trình sẽ kiểm tra biểu thức điều kiện, nếu đúng chương trình sẽ thực hiện các câu lệnh bên trong thân của vòng lặp while, sau đó quay lại kiểm tra biểu thức điều kiện. Còn nếu biểu thức điều kiện sai thì thoát khỏi vòng lặp while.

### ➤ Câu trúc Do...While

```
do  
{  
    // các câu lệnh;  
}  
while(dieu_kien);
```

Chương trình thực hiện các câu lệnh sau đó kiểm tra biểu thức điều kiện nếu đúng thì lặp lại thực hiện các câu lệnh, nếu sai thì thoát khỏi vòng lặp.

### ➤ Câu trúc IF, IF ...ELSE

```
- Câu trúc 1:  
if(Biểu thức điều kiện)  
{  
    // Đoạn chương trình;  
}
```

Chương trình kiểm tra biểu thức điều kiện, nếu đúng thì xử lý các câu lệnh bên trong còn sai thì nhảy qua.

```
- Câu trúc 2:  
if(biểu thức điều kiện)  
{  
    // Đoạn chương trình 1;  
}  
else  
{  
    // Đoạn chương trình 2;  
}
```

Nếu biểu thức điều kiện đúng thì xử lý “Đoạn chương trình 1” bên trong còn sai thì xử lý “Đoạn chương trình 2”.

### ➤ Câu trúc SWITCH

```
switch(biến)  
{
```

```

case gia_tri_1:
//các câu lệnh;
break;
case gia_tri_2:
//các câu lệnh;
break;
.....
case gia_tri_n:
//các câu lệnh;
break;
default:
//các câu lệnh
}

```

Tuỳ vào **biến** có giá trị bằng giá trị của **Case** nào thì thực hiện các câu lệnh tương ứng trong **Case** đó, sau đó thoát khỏi cấu trúc nhờ câu lệnh “**break;**”. Nếu không có **Case** nào phù hợp thì thực hiện các câu lệnh trong **default**.

### ➤ Cấu trúc FOR

```

for( x=n ; điều_kiện ; phép_toán )
{
// các câu lệnh xử lý;
}

```

Giải thích: x là biến, n là giá trị xác định. Trước tiên vòng lặp sẽ gán giá trị ban đầu cho biến: x=n, rồi kiểm tra nếu điều\_kiện đúng thì thực hiện các câu lệnh xử lý, sau đó thực hiện Phép\_toán nhằm tác động đến điều kiện. Sau đó lại kiểm tra lại điều\_kiện, nếu còn đúng thì thực hiện tiếp, nếu sai sẽ thoát khỏi vòng lặp.

## 2.3. HOẠT ĐỘNG NGẮT

### 2.3.1. Giới thiệu

Trong thực tế người ta rất muốn tận dụng khả năng của CPU để làm thêm được nhiều công việc khác nữa, chỉ khi nào có cần trao đổi dữ liệu thì mới yêu cầu CPU tạm dừng công việc hiện tại để phục vụ việc trao đổi dữ liệu. Sau khi hoàn thành việc trao đổi dữ liệu thì CPU lại phải quay về để làm tiếp công việc hiện đang bị gián đoạn. Cách làm việc theo kiểu này gọi là *ngắt CPU (gián đoạn hoạt động của CPU)*. Một hệ thống sử dụng *ngắt* có thể đáp ứng rất nhanh các yêu cầu trao đổi dữ liệu trong khi vẫn có thể làm được các công việc khác.

Cần phải phân biệt sự giống và khác nhau giữa ngắt và gọi chương trình con.

- Giống nhau:**

Khi xảy ra điều kiện tương ứng thì CPU sẽ tạm dừng chương trình chính đang thực thi để thực thi một chương trình khác (chương trình con/chương trình xử lý ngắt) rồi sau đó CPU sẽ quay về thực thi tiếp tục chương trình chính đang bị tạm dừng.

- Khác nhau:**

	<b>Ngắt</b>	<b>Chương trình con</b>
<b>Thời điểm xảy ra sự kiện</b>	Không biết trước (hay xảy ra không đồng bộ với chương trình chính)	Biết trước (hay xảy ra đồng bộ với chương trình chính)
<b>Nguyên nhân dẫn đến sự kiện</b>	Do các tín hiệu điều khiển từ Timer, Serial port và bên ngoài chip	Do lệnh gọi chương trình con

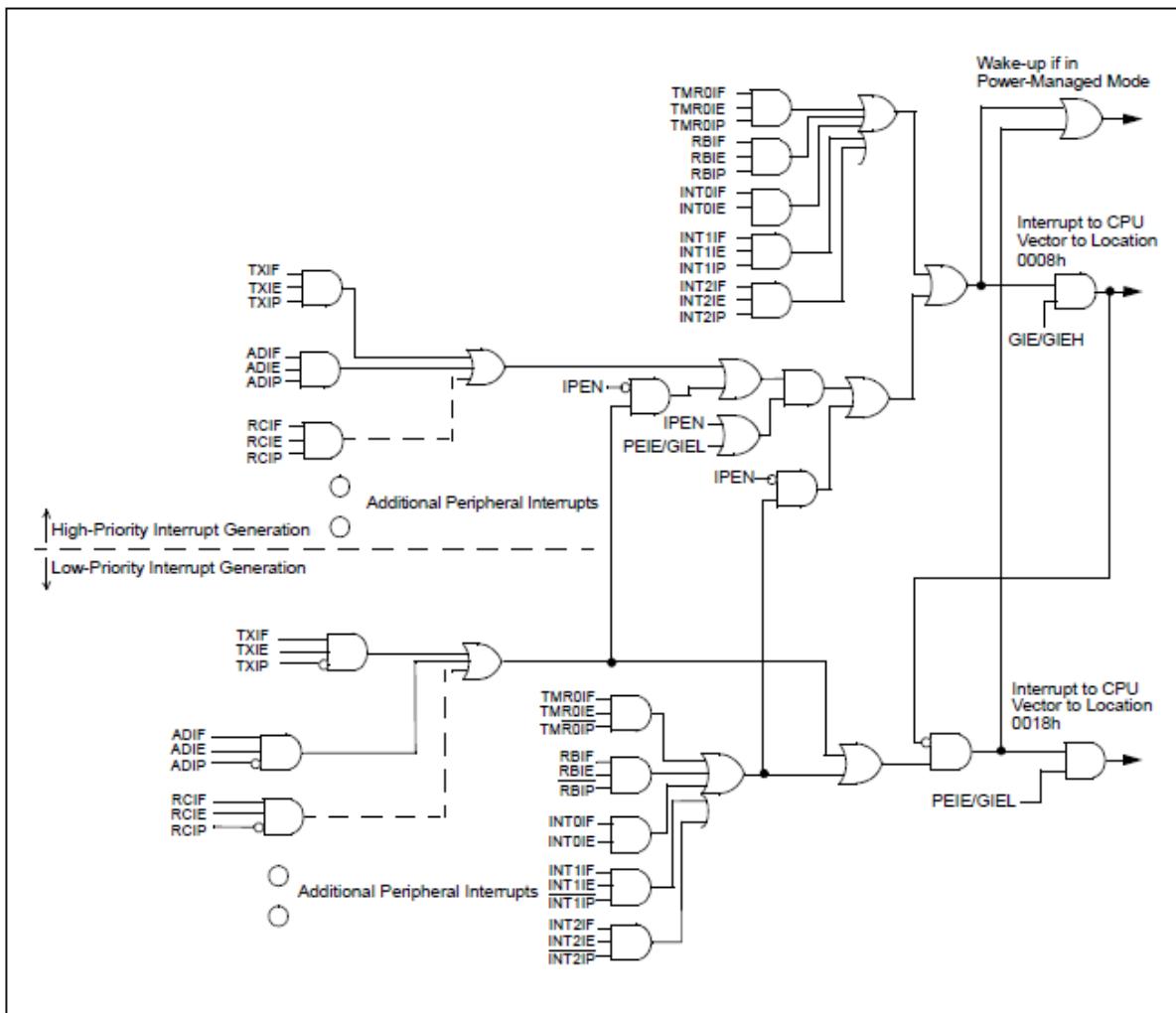
### 2.3.2. Tổ chức ngắt của PIC 18F4431

Vì điều khiển PIC18F4431 có hai vector ưu tiên ngắt: Vector ưu tiên ngắt cao (địa chỉ 000008H) và vector ưu tiên ngắt thấp (địa chỉ 000018H). Các nguồn ngắt (hình 2.6) đều có thể đặt mức ưu tiên cao hoặc thấp (trừ ngắt INT0 -ngắt ngoài 0). Khi xảy ra đồng thời cả hai ngắt ưu tiên cao và ngắt ưu tiên thấp thì ngắt ưu tiên cao được thực hiện trước, ngắt ưu tiên thấp thực hiện sau. Khi đang thực hiện chương trình con phục vụ ngắt của ngắt ưu tiên thấp mà có ngắt ưu tiên cao xảy ra thì chương trình con phục vụ ngắt thấp được tạm dừng để thực hiện chương trình con phục vụ ngắt cao, khi chương trình phục vụ ngắt ưu tiên cao thực thi xong, CPU sẽ quay lại thực hiện tiếp chương trình con phục vụ ngắt thấp.

Các nguồn ngắt của PIC18F4431:

- Ngắt ngoài trên các chân RC3/INT0, RC4/INT1 và RC5/INT2.
- Ngắt do các bộ timer bao gồm: Timer0, Timer1, Timer2, Timer3.
- Ngắt do PORTB.
- Ngắt từ bộ biến đổi A/D.
- Ngắt từ PORT nối tiếp EUSART gồm ngắt truyền và ngắt nhận.
- Ngắt từ module MSSP.
- Ngắt Capture, Compare từ CCP1 và CCP2.
- Ngắt do hỏng bộ phát xung hệ thống.
- Ngắt từ module so sánh tương tự.
- Ngắt từ hoạt động ghi vào bộ nhớ dữ liệu EEPROM/Flash.

- Ngắt do sự xung đột bus.
- Ngắt do phát hiện được mức điện áp cao/thấp.



Hình 2.6. Logic ngắt của PIC 18F4431

Để sử dụng một nguồn ngắt nào đó, ta phải quan tâm đến các bit sau:

- Bit cờ ngắt (Flag Bit): Khi xảy ra ngắt từ một nguồn ngắt nào đó thì bit cờ ngắt tương ứng được đặt (set) bằng 1. Kiểm tra (đọc) cờ ngắt sẽ biết được ngắt xảy ra từ đâu, sau đó xóa cờ ngắt để thực hiện lần ngắt tiếp theo (các bit cờ không được xóa bằng phần cứng).

- Bit cho phép ngắt (Interrupt Enable Bit): Để cho phép một nguồn ngắt nào đó, ta phải đặt bit cho phép ngắt tương ứng bằng 1, ngược lại, để cấm một nguồn ngắt thì sẽ đặt bit cho phép ngắt của nguồn ngắt đó bằng 0. Mỗi nguồn ngắt có một bit cho phép tương ứng, ngoài ra tất cả các nguồn ngắt còn được cho phép/cấm bởi bit cho phép ngắt toàn cục (GIE- Global Interrupt Enable); các nguồn ngắt ngoại vi được cho phép/cấm bởi bit cho phép ngắt ngoại vi (PEIE-Peripheral Interrupt Enable).

- Bit ưu tiên ngắt (Interrupt priority bit): Bit này cho phép đặt một nguồn ngắt nào đó ở mức ưu tiên cao hoặc thấp.

- Bit lựa chọn cách tác động ngắt. Với các nguồn ngắt ngoài INT0, INT1 và INT2 có 02 cách tác động ngắt: Tác động bằng sườn dương (Rising Edge) hoặc sườn âm (Falling Edge).

### ➤ Ngắt ngoài

Ngắt ngoài INT0, INT1 và INT2 được tích cực bằng sườn xung bên ngoài đặt lên các chân RB0/INT0, RB1/INT1 và RB2/INT2 tương ứng. Nếu bit INTEDGx trên thanh ghi INTCON2 được thiết lập (đặt bằng 1), ngắt ngoài tương ứng sẽ được tích cực bằng sườn dương. Ngược lại xóa INTEDGx (đặt bằng 0), ngắt ngoài ngoài tương ứng sẽ tích cực bằng sườn âm. Khi có sườn xung trên chân RBx/INT, bit cờ ngắt INTxFIF sẽ được thiết lập. Bit cờ ngắt INTxFIF phải được xóa bằng phần mềm trước khi lặp lại quá trình cho phép ngắt. Tất cả ngắt ngoài có thể “đánh thức” vi xử lý từ chế độ Idle hoặc Sleep nếu bit INTxIE được thiết lập trước khi thực hiện các chế độ quản lý nguồn này. Ngắt ngoài INT1 và ngắt ngoài INT2 có thể lựa chọn mức ưu tiên cao hoặc thấp thông qua các bit INTxIP trên thanh ghi INTCON3 (=1 mức ưu tiên cao, =0 mức ưu tiên thấp), riêng ngắt ngoài INT0 luôn được đặt ở mức ưu tiên cao.

### ➤ Các thanh ghi liên quan

Vì điều khiển PIC 18F4431 có 13 thanh ghi điều khiển hoạt động ngắt:

- RCON – thanh ghi điều khiển Reset.
- INTCON – thanh ghi điều khiển ngắt.
- INTCON2 – thanh ghi điều khiển ngắt 2.
- INTCON3 – thanh ghi điều khiển ngắt 3.
- PIR1, PIR2, PIR3 – thanh ghi yêu cầu ngắt ngoại vi 1, thanh ghi yêu cầu ngắt ngoại vi 2.
- PIE1, PIE2, PIE3 – thanh ghi cho phép ngắt ngoại vi 1, thanh ghi cho phép ngắt ngoại vi 2.
- IPR1, IPR2, IPR3 – thanh ghi ưu tiên ngắt ngoại vi 1, thanh ghi ưu tiên ngắt ngoại vi 2.

#### **1. Thanh ghi RCON**

R/W-0	U-0	U-0	R/W-1	R-1	R-1	R/W-0	R/W-0
IPEN	—	—	RI	TO	PD	POR	BOR
bit 7							bit 0

bit 7    **IPEN:** Bit cho phép ưu tiên ngắt

1 = Cho phép ưu tiên ngắt.

0 = Không cho phép ưu tiên ngắt.

Các bít khác: Tham khảo datasheet.

## 2. Các thanh ghi điều khiển ngắt INTCON

Các thanh ghi INTCON (Interrupt Control) có thể ghi/đọc theo cả byte hoặc từng bit. Các thanh ghi này chứa các bit cho phép/cấm các nguồn ngắt, các bit đặt mức ưu tiên, các bit cờ ngắt.

### - Thanh ghi điều khiển ngắt: INTCON

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/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							

Ghi chú: R = Cho phép đọc W = Cho phép ghi U = Không sử dụng, đọc bằng '0' -n = Reset - POR '1' = Được thiết lập '0' = Được xóa -x = Reset không xác định

bit 7 **GIE/GIEH:** Bit cho phép ngắt toàn cục

Khi bit IPEN = 0:

1= Cho phép tắt cả các ngắt không sử dụng mặt nạ (không ưu tiên ngắt)

0 = Cấm tắt cả các ngắt

Khi bit IPEN = 1:

1 = Cho phép tắt cả các ngắt ưu tiên cao

0 = Cấm tắt cả các ngắt

bit 6 **PEIE/GIEL:** Bit cho phép ngắt ngoại vi

Khi bit IPEN = 0:

1 = Cho phép tắt cả các ngắt ngoại vi không sử dụng mặt nạ

0 = Cấm tắt cả các ngắt ngoại vi

Khi bit IPEN = 1:

1 = Cho phép tắt cả các ngắt ngoại vi ưu tiên thấp

0 = Cấm tắt cả các ngắt ngoại vi được ưu tiên thấp

bit 5 **TMR0IE:** Bit cho phép ngắt tràn Timer0 (TMR0)

1 = Cho phép ngắt tràn Timer0(TMR0)

0 = Cấm ngắt tràn Timer0(TMR0)

bit 4 **INT0IE:** Bit cho phép ngắt ngoài INT0

1 = Cho phép ngắt ngoài INT0

0 = Cấm ngắt ngoài INT0

bit 3 **RBIE:** Bit cho phép ngắt do thay đổi mức trên PortB

1 = Cho phép Ngắt khi có thay đổi mức trên PortB

- 0 = Cảm Ngắt khi có thay đổi mức trên PortB
- bit 2 **TMR0IF:** Bit cờ báo ngắt tràn Timer0 (TMR0)  
 1 = Tràn thanh ghi TMR0 của Time0 (phải được xóa bằng phần mềm)  
 0 = không xảy ra tràn thanh ghi TMR0
- bit 1 **INT0IF:** Bit cờ báo ngắt ngoài INT0  
 1 = Có ngắt ngoài INT0 (phải được xóa bằng phần mềm)  
 0 = Chưa phát hiện ngắt ở chân INT0
- bit 0 **RBIF:** Bit cờ ngắt báo đã thay đổi mức trên PortB  
 1 = Ít nhất một bit từ RB7:RB4 có sự thay đổi trạng thái (bit này phải được xóa bằng phần mềm)  
 0 = Không có sự thay đổi trạng thái trên các chân RB7:RB4

- *Thanh ghi điều khiển ngắt 2: INTCON2*

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

- bit 7 **RBPU:** Bit cho phép treo các chân PORTB (Pull-up)  
 1 = Cảm treo PORTB  
 0 = Cho phép treo PORTB
- bit 6 **INTEDG0:** Bit lựa chọn sườn xung cho ngắt ngoài INT0  
 1 = Ngắt bằng sườn dương  
 0 = Ngắt bằng sườn âm
- bit 5 **INTEDG1:** Bit lựa chọn sườn xung cho ngắt ngoài INT1  
 1 = Ngắt bằng sườn dương  
 0 = Ngắt bằng sườn âm
- bit 4 **INTEDG2:** Bit lựa chọn sườn xung cho ngắt ngoài INT2  
 1 = Ngắt bằng sườn dương  
 0 = Ngắt bằng sườn âm
- bit 3 Bit không được định nghĩa
- bit 2 **TMR0IP:** Bit lưu chọn mức ưu tiên ngắt cho ngắt tràn Timer0 (TMR0)  
 1 = Ưu tiên cao  
 0 = Ưu tiên thấp
- bit 1 Bit không được định nghĩa
- bit 0 **RBIP:** Bit lựa chọn mức ưu tiên ngắt do thay đổi PortB

1 = Ưu tiên cao

0 = Ưu tiên thấp

- *Thanh ghi điều khiển ngắt 3: INTCON3*

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP:** Bit lưu chọn mức ưu tiên ngắt cho ngắt ngoài INT2

1 = Ưu tiên cao

0 = Ưu tiên thấp

bit 6 **INT1IP:** Bit lưu chọn mức ưu tiên ngắt cho ngắt ngoài INT1

1 = Ưu tiên cao

0 = Ưu tiên thấp

bit 5 Không được định nghĩa

bit 4 **INT2IE:** Bit cho phép ngắt ngoài INT2

1 = Cho phép ngắt ngoài INT2

0 = Cấm ngắt ngoài INT2

bit 3 **INT1IE:** Bit cho phép ngắt ngoài INT1

1 = Cho phép ngắt ngoài INT1

0 = Cấm ngắt ngoài INT1

bit 2 Không được định nghĩa

bit 1 **INT2IF:** Cờ báo ngắt ngoài INT2

1= Xuất hiện ngắt ngoài trên chân INT2 (phải được xóa bằng phần mềm)

0 = Không xảy ra ngắt trên chân INT2

bit 0 **INT1IF:** Cờ báo ngắt ngoài INT1

1= Xuất hiện ngắt ngoài trên chân INT1 (phải được xóa bằng phần mềm)

0 = Không xảy ra ngắt trên chân INT1

**3. Các thanh ghi yêu cầu ngắt PIR**

Các thanh ghi yêu cầu ngắt ngoại vi PIR (Peripheral Interrupt Request) chứa các bit cờ phản ánh trạng thái có/không có các yêu cầu ngắt ngoại vi.

- *Thanh ghi yêu cầu ngắt ngoại vi 1: PIR1*

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7	bit 0						

- bit 7 Không được định nghĩa
- bit 6 **ADIF:** Bit cờ ngắt của bộ biến đổi A/D  
 1 = Biến đổi A/D đã hoàn thành (phải xóa bằng phần mềm)  
 0 = Biến đổi A/D chưa hoàn thành
- bit 5 **RCIF:** Bit cờ ngắt nhận port nối tiếp EUSART  
 1 = khi bộ đếm nhận RCREG nhận đủ dữ liệu (được xóa khi đọc RCREG)  
 0 = bộ đếm nhân rỗng
- bit 4 **TXIF:** Bit cờ ngắt truyền port nối tiếp EUSART  
 1 = khi bộ đếm truyền TXREG rỗng (xóa khi TXREG được ghi)  
 0 = khi bộ đếm truyền được ghi EUSART
- bit 3 **SSPIF:** Bit cờ ngắt Port nối tiếp đồng bộ chủ MSSP  
 1 = khi việc truyền và nhận ở port nối tiếp hoàn thành (xóa bằng phần mềm)  
 0 = chờ truyền/nhận
- bit 2 **CCP1IF:** Bit cờ ngắt module CCP1 (CAPTURE/COMPARE/PWM)  
Chế độ Capture:  
 1 = Xảy ra hiện tượng Capture ở thanh ghi TMR1 (xóa bằng phần mềm)  
 0 = Không xảy ra hiện tượng chụp ở thanh ghi TMR1  
Chế độ so sánh (Compare):  
 1 = Giá trị thanh ghi đếm TMR1 đếm bằng thanh ghi so sánh (xóa bằng phần mềm)  
 0 = Giá trị TMR1 chưa đếm bằng thanh ghi so sánh  
Chế độ PWM:  
 Không sử dụng ở chế độ này
- bit 1 **TMR2IF:** Bit cờ ngắt khi bộ đếm TMR2 so sánh giá trị ở PR2 của Timer2.  
 1 = Khi giá trị TMR2 bằng giá trị PR2 (được xóa bằng phần mềm)  
 0 = TMR2 không phù hợp PR2
- bit 0 **TMR1IF:** Bit cờ ngắt tràn Timer1 (TMR1)  
 1 = Khi tràn thanh ghi TMR1 (Phải được xóa bằng phần mềm)

0 = Chưa phát hiện tràn trên thanh ghi TMR1

- *Thanh ghi yêu cầu ngắt ngoại vi 2: PIR2*

R/W-0	U-0	U-0	R/W-0	U-0	R/W-0	U-0	R/W-0
OSCFIF	—	—	EEIF	—	LVDIF	—	CCP2IF
bit 7							bit 0

bit 7 **OSCFIF:** Bit cờ ngắt lỗi bộ phát xung (Oscillator Fail)

1 = Bộ phát xung đã vị lỗi, đầu vào tần số INTOSC đã bị thay đổi (phải được xóa bằng phần mềm)

0 = Thiết bị phát xung hoạt động bình thường

bit 6 Không được định nghĩa

bit 5 Không được định nghĩa

bit 4 **EEIF:** Bit cờ ngắt do hoạt động ghi dữ liệu vào EEPROM/Flash

1 = Hoàn thành việc ghi dữ liệu (xóa bằng phần mềm)

0 = Hoạt động chưa hoàn thành hoặc chưa được bắt đầu

bit 3 Không được định nghĩa

bit 2 **LVDIF:** Bit cờ ngắt phát hiện điện áp Cao/thấp (High/Low-Voltage Detect)

1 = Điện áp dưới điện áp LVD (phải xóa bằng phần mềm)

0 = Điện áp cao hơn điện áp LVD

bit 1 Không được định nghĩa

bit 0 **CCP2IF:** Bit cờ ngắt của module CCP2 (Capture/Compare/PWM 2)

Chế độ Capture:

1 = Sau khi chụp thanh ghi TMR1 (xóa bằng phần mềm)

0 = Không xuất hiện chụp TMR1

Chế độ so sánh (Compare):

1 = Khi giá trị trong thanh ghi TMR1 bằng (match) giá trị trong thanh ghi CCPR2 (xóa bằng phần mềm)

0 = Khi giá trị trong thanh ghi TMR1 chưa bằng giá trị trong thanh ghi CCPR2

- *Thanh ghi yêu cầu ngắt ngoại vi 3: PIR3*

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	PTIF	IC3DRIF	IC2QEIF	IC1IF	TMR5IF
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as ‘0’

bit 4 **PTIF:** PWM Time Base Interrupt bit

- 1 = PWM time base matched the value in the PTPER registers. Interrupt is issued according to the postscaler settings. PTIF must be cleared in software.
- 0 = PWM time base has not matched the value in the PTPER registers
- bit 3 **IC3DRIF:** IC3 Interrupt Flag/Direction Change Interrupt Flag bit
- IC3 Enabled (CAP3CON<3:0>):
- 1 = TMR5 value was captured by the active edge on CAP3 input (must be cleared in software)
- 0 = TMR5 capture has not occurred
- QEI Enabled (QEIM<2:0>):
- 1 = Direction of rotation has changed (must be cleared in software)
- 0 = Direction of rotation has not changed
- bit 2 **IC2QEIF:** IC2 Interrupt Flag/QEI Interrupt Flag bit
- IC2 Enabled (CAP2CON<3:0>):
- 1 = TMR5 value was captured by the active edge on CAP2 input (must be cleared in software)
- 0 = TMR5 capture has not occurred
- QEI Enabled (QEIM<2:0>):
- 1 = The QEI position counter has reached the MAXCNT value, or the index pulse, INDX, has been detected. Depends on the QEI operating mode enabled. Must be cleared in software.
- 0 = The QEI position counter has not reached the MAXCNT value or the index pulse has not been detected
- bit 1 **IC1 Enabled (CAP1CON<3:0>):**
- 1 = TMR5 value was captured by the active edge on CAP1 input (must be cleared in software)
- 0 = TMR5 capture has not occurred
- QEI Enabled (QEIM<2:0>), Velocity Measurement Mode Enabled (VELM = 0 in QEICON register):
- 1 = Timer5 value was captured by the active velocity edge (based on PHA or PHB input). CAP1REN bit must be set in CAP1CON register. IC1IF must be cleared in software.
- 0 = Timer5 value was not captured by the active velocity edge
- bit 0 **TMR5IF:** Timer5 Interrupt Flag bit

1 = Timer5 time base matched the PR5 value (must be cleared in software)

0 = Timer5 time base did not match the PR5 value

#### **4. Các thanh ghi cho phép ngắt ngoại vi PIE**

Các thanh ghi cho phép ngắt ngoại vi PIE (Peripheral Interrupt Enable) chứa các bit cho phép các nguồn ngắt ngoại vi. Có 3 thanh ghi cho phép ngắt ngoại vi là PIE1, PIE2 và PIE3. Khi bit IPEN (bit 7 thanh ghi RCON) được đặt bằng 0, muốn cho phép các nguồn ngắt ngoại vi bit PEIE (bit 6 trong thanh ghi INTCON) phải được đặt bằng 1.

##### *- Thanh ghi cho phép ngắt ngoại vi 1: PIE1*

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 Không được định nghĩa

bit 6 **ADIE:** Bit cho phép ngắt do biến đổi A/D

1 = Cho phép ngắt biến đổi A/D

0 = Cấm ngắt biến đổi A/D

bit 5 **RCIE:** Bit cho phép ngắt do nhận ở PORT nối tiếp (EUSART)

1 = Cho phép ngắt

0 = Cấm ngắt

bit 4 **TXIE:** Bit cho phép ngắt do truyền ở PORT nối tiếp (EUSART)

1 = Cho phép ngắt

0 = Cấm ngắt

bit 3 **SSPIE:** Bit cho phép ngắt PORT nối tiếp đồng bộ chủ - MSSP

1 = Cho phép ngắt (MSSP)

0 = Cấm ngắt

bit 2 **CCP1IE:** Bit cho phép ngắt của module CCP1 (Capture/Compare/PWM

1)

1 = Cho phép ngắt CCP1

0 = Cấm

bit 1 **TMR2IE:** Bit cho phép ngắt do so sánh TMR2 và PR2 của Timer2

1 = Cho phép ngắt

0 = Cấm

bit 0 **TMR1IE:** Bit cho phép ngắt tràn của Timer1

1 = Cho phép ngắt do tràn TMR1

0 = Cấm ngắt

- Thanh ghi cho phép ngắt ngoại vi 2: PIE2

R/W-0	U-0	U-0	R/W-0	U-0	R/W-0	U-0	R/W-0
OSCFIE	—	—	EEIE	—	LVDIE	—	CCP2IE
bit 7							bit 0

bit 7 **OSCFIE:** Bit cho phép ngắt do lỗi bộ phát xung (Oscillator Fail)

1 = Cho phép

0 = Cấm

bit 6 Không được định nghĩa

bit 5 Không được định nghĩa

bit 4 **EEIE:** Bit cho phép ngắt ghi dữ liệu vào EEPROM/Flash

1 = Cho phép

0 = Cấm

bit 3 Không được định nghĩa

bit 2 **LVDIE:** Bit cho phép ngắt do module LVD

1 = Cho phép

0 = Cấm

bit 1 Không được định nghĩa

bit 0 **CCP2IE:** Bit cho phép ngắt từ module CCP2 (Capture/Compare/PWM

2)

1 = Cho phép

0 = Cấm

- Thanh ghi cho phép ngắt ngoại vi 3: PIE3

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	PTIE	IC3DRIE	IC2QEIE	IC1IE	TMR5IE
bit 7							bit 0

bit 7-5 **Không sử dụng:** Read as ‘0’

bit 4 **PTIE:** PWM Time Base Interrupt Enable bit

1 = PTIF enabled

0 = PTIF disabled

bit 3 **IC3DRIE:** IC3 Interrupt Enable/Direction Change Interrupt Enable bit

IC3 Enabled (CAP3CON<3:0>):

1 = IC3 interrupt enabled

0 = IC3 interrupt disabled

QEI Enabled (QEIM<2:0>):

1 = Change of direction interrupt enabled

0 = Change of direction interrupt disabled

bit 2 **IC2QEIE:** IC2 Interrupt Flag/QEI Interrupt Flag Enable bit

IC2 Enabled (CAP2CON<3:0>):

1 = IC2 interrupt enabled

0 = IC2 interrupt disabled

QEI Enabled (QEIM<2:0>):

1 = QEI interrupt enabled

0 = QEI interrupt disabled

bit 1 **IC1IE:** IC1 Interrupt Enable bit

1 = IC1 interrupt enabled

0 = IC1 interrupt disabled

bit 0 **TMR5IE:** Timer5 Interrupt Enable bit

1 = Timer5 interrupt enabled

0 = Timer5 interrupt disabled

## 5. Các thanh ghi ưu tiên ngắt IPR

Các thanh ghi ưu tiên ngắt IPR(Interrupt Priority Registers) chứa các bit cho phép đặt các nguồn ngắt ở các mức ưu tiên cao hoặc thấp. Muốn đặt được các mức ưu tiên cho các nguồn ngắt, bit IPEN (bit 7 thanh ghi RCON) phải được đặt bằng 1. Có 3 thanh ghi ưu tiên ngắt, lần lượt là IPR1, IPR2 và IPR3.

### - Thanh ghi ưu tiên ngắt 1: IPR1

U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	ADIP	RCIP	TXIP	SSPIP	CCPIP	TMR2IP	TMR1IP
bit 7							bit 0

bit 7 Không được định nghĩa

bit 6 **ADIP:** Bit ưu tiên ngắt biến đổi A/D (Analog Digital Converter)

1 = Uy tiên cao

0 = Uy tiên thấp

bit 5 **RCIP:** Bit ưu tiên ngắt nhận PORT nối tiếp (EUSART Receive)

1 = Uy tiên cao

0 = Uy tiên thấp

bit 4 **TXIP:** Bit ưu tiên ngắt truyền PORT nối tiếp (EUSART Transmit)

- 1 = Uu tiên cao  
0 = Uu tiên thấp
- bit 3 **SSPIP:** Bit ưu tiên ngắt PORT nối tiếp đồng bộ chủ MSSP  
1 = Uu tiên cao  
0 = Uu tiên thấp
- bit 2 **CCP1IP:** Bit ưu tiên ngắt của module CCP1  
1 = Uu tiên cao  
0 = Uu tiên thấp
- bit 1 **TMR2IP:** Bit ưu tiên ngắt so sánh giữa TMR2 và PR2  
1 = Uu tiên cao  
0 = Uu tiên thấp
- bit 0 **TMR1IP:** Bit ưu tiên ngắt tràn do Timer1  
1 = Uu tiên cao  
0 = Uu tiên thấp

- Thanh ghi ưu tiên ngắt 2: IPR2

R/W-1	U-0	U-0	R/W-1	U-0	R/W-1	U-0	R/W-1
OSCFIP	—	—	EEIP	—	LVDIP	—	CCP2IP
bit 7							bit 0

- bit 7 **OSCFIP:** Bit ưu tiên ngắt do lỗi bộ phát xung (Oscillator Fail)  
1 = Uu tiên cao  
0 = Uu tiên thấp
- bit 6 Không được định nghĩa
- bit 5 Không được định nghĩa
- bit 4 **EEIP:** Bit ưu tiên ngắt đọc bộ nhớ EEPROM/Flash  
1 = Uu tiên cao  
0 = Uu tiên thấp
- bit 3 Không được định nghĩa
- bit 2 **LVDIP:** Bit ưu tiên ngắt phát hiện điện áp cao/thấp  
1 = Uu tiên cao  
0 = Uu tiên thấp
- bit 1 Không được định nghĩa
- bit 0 **CCP2IP:** Bit ưu tiên ngắt CCP2  
1 = Uu tiên cao

0 = Uu tiên thấp

- Thanh ghi ưu tiên ngắt 3: IPR3

U-0	U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	—	—	PTIP	IC3DRIP	IC2QEIP	IC1IP	TMR5IP
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as ‘0’

bit 4 **PTIP:** PWM Time Base Interrupt Priority bit

1 = High priority

0 = Low priority

bit 3 **IC3DRIP:** IC3 Interrupt Priority/Direction Change Interrupt Priority bit

IC3 Enabled (CAP3CON<3:0>):

1 = IC3 interrupt high priority

0 = IC3 interrupt low priority

QEI Enabled (QEIM<2:0>):

1 = Change of direction interrupt high priority

0 = Change of direction interrupt low priority

bit 2 **IC2QEIP:** IC2 Interrupt Priority/QEI Interrupt Priority bit

IC2 Enabled (CAP2CON<3:0>):

1 = IC2 interrupt high priority

0 = IC2 interrupt low priority

QEI Enabled (QEIM<2:0>):

1 = High priority

0 = Low priority

bit 1 **IC1IP:** IC1 Interrupt Priority bit

1 = High priority

0 = Low priority

bit 0 **TMR5IP:** Timer5 Interrupt Priority bit

1 = High priority

0 = Low priority

#### ➤ Lập trình sử dụng ngắt

Với các thiết kế không cần sử dụng ưu tiên ngắt, khi đó bit IPEN (RCON<7>) sẽ được đặt bằng 0, thứ tự các bit cần tác động như sau:

- Đặt bit GIE/GIEH (RCON<7>) bằng 1 để cho phép ngắt toàn cục.

- Đặt bit PEIE/GIEL (RCON<6>) bằng 1 để cho phép các ngắt ngoại vi.
- Đặt bit cho phép ngắt tương ứng với nguồn ngắt được sử dụng bằng 1.

Với các thiết kế sử dụng ưu tiên ngắt, khi đó bit IPEN (RCON<7>) sẽ được đặt bằng 1, thứ tự các bit cần tác động như sau:

- Đặt bit GIE/GIEH (RCON<7>) bằng 1 để cho phép ngắt ưu tiên cao.
- Đặt bit PEIE/GIEL (RCON<6>) bằng 1 để cho phép ngắt ưu tiên thấp.
- Đặt bit cho phép ngắt tương ứng với nguồn ngắt được sử dụng bằng 1.
- Lựa chọn mức ưu tiên ngắt qua các bit ưu tiên ngắt tương ứng.

Khi xảy ra ngắt, cờ ngắt tương ứng sẽ được thiết lập bằng 1. Vì điều khiển PIC 18F4431 chỉ có 2 vector ngắt nên người lập trình cần phải kiểm tra cờ ngắt để thực hiện chương trình con phục vụ ngắt tương ứng.

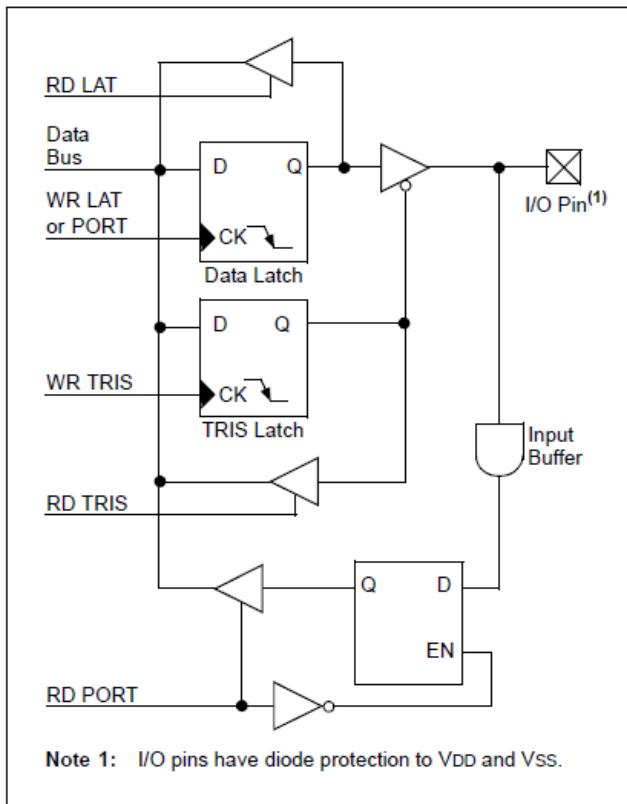
## 2.4. HOẠT ĐỘNG VÀO/RA (I/O)

Vi điều khiển PIC 18F4431 có 36 chân vào/ra được chia thành 5 cổng là PORTA, PORTB, PORTC, PORTD có 8 chân và PORTE có 4 chân. Các chân vào/ra của vi điều khiển PIC 18F4431 mang nhiều chức năng, nó có thể được thiết lập là chân vào/ra dữ liệu hay là các chân chức năng đặc biệt của các bộ ngoại vi, sử dụng các thanh ghi điều khiển của ngoại vi để lựa chọn chức năng cho các chân.

Mỗi cổng vào/ra của vi điều khiển PIC 18F4431 có 3 thanh ghi để điều khiển hoạt động:

- Thanh ghi TRIS (Data Direction register) là thanh ghi lựa chọn hướng dữ liệu (bằng ‘0’ chiều ra, bằng ‘1’ chiều vào).
- Thanh ghi PORT (reads the levels on the pins of the device) là thanh ghi dữ liệu, được định địa chỉ theo byte và theo bit, sử dụng để đếm đọc/ghi dữ liệu trên các cổng.
- Thanh ghi LAT (Data Latch) là thanh ghi chốt dữ liệu đầu ra (bằng ‘1’ đầu ra chốt mức 1, bằng ‘0’ đầu ra chốt mức 0).

Hình 2.7 thể hiện cấu trúc chung của một chân vào/ra.



Hình 2.7. Cấu trúc chung của một chân vào/ra.

#### 2.4.1. PORTA

PORTA gồm 8 bit tương ứng với 8 chân được ký hiệu từ RA0 đến RA7. Các chân của PORTA có thể đọc/ghi theo từng bit hoặc cả byte.

Bảng 2.1. Chức năng các chân trên PORTA.

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
RA0/AN0	RA0	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<0>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTA<0>. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN0	1	<b>I</b>	ANA	Đầu vào kênh 0 của bộ biến đổi A/D. Mặc định khi cấu hình POR.
RA1/AN1	RA1	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<1>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTA<1>, Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN1	1	<b>I</b>	ANA	Đầu vào kênh 1 của bộ biến đổi A/D. Mặc định khi cấu hình POR.

<b>Chân</b>	<b>Chức năng</b>	<b>Thanh ghi TRIS</b>	<b>I/O</b>	<b>Kiểu I/O</b>	<b>Mô tả</b>
RA2/AN2/VREF-/CAP1/INDX	RA2	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<2>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTA<2>. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN2	1	<b>I</b>	ANA	Đầu vào kênh 2 của bộ biến đổi A/D. Mặc định khi cấu hình POR.
	VREF-	1	<b>I</b>	ANA	Đầu vào điện áp tham chiếu mức thấp bộ A/D.
	CAP1	1	<b>I</b>	ST	Đầu vào Input Capture 1. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	INDX	1	<b>I</b>	ST	Đầu vào Quadrature Encoder Interface index. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
RA3/AN3/VREF+/CAP2/QEA	RA3	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<3>. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTA<3>. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN3	1	<b>I</b>	ANA	Đầu vào kênh 3 của bộ biến đổi A/D. Mặc định khi cấu hình POR.
	VREF+	1	<b>I</b>	ANA	Đầu vào điện áp tham chiếu mức cao bộ A/D.
	CAP2	1	<b>I</b>	ST	Đầu vào Input Capture 2. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép
	QEA	1	<b>I</b>	ST	Đầu vào kênh A bộ Quadrature Encoder Interface. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép
RA4/AN4/CAP3/QEB	RA4	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<4>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	<b>I</b>	ST	Chiều vào dữ liệu sử dụng bit PORTA<4>. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN4	1	<b>I</b>	ANA	Đầu vào kênh 4 của bộ biến đổi A/D. Mặc định khi cấu hình POR.
	CAP3	1	<b>I</b>	ST	Đầu vào Input Capture 3. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép
	QEB	1	<b>I</b>	ST	Đầu vào kênh B bộ Quadrature Encoder Interface. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
RA5/AN5/LVDIN	RA5	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<5>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTA<5>. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN5	1	<b>I</b>	ANA	Đầu vào kênh 5 của bộ biến đổi A/D. Mặc định khi cấu hình POR.
	LVDIN	1	<b>I</b>	ANA	Low-Voltage Detect external trip point input.
OSC2/CLKO/RA6	OSC2	x	<b>O</b>	ANA	Kết nối với bộ phát xung chính (XT, HS và LP).
	CLKO	x	<b>O</b>	DIG	Chân phát xung hệ thống (FOSC/4) ở chế độ dao động RC, INTIO1 và EC.
	RA6	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<6>. Chức năng này chỉ được cho phép ở các chế độ RCIO, INTIO2 và ECIO.
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTA<6>. Chức năng này chỉ được cho phép ở các chế độ RCIO, INTIO2 và ECIO.
OSC1/CLKI/RA7	OSC1	x	<b>I</b>	ANA	Kết nối với bộ dao động ngoài
	CLKI	x	<b>I</b>	ANA	Kết nối với nguồn xung bên ngoài
	RA7	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATA<7>. Chức năng này bị cấm ở chế độ dao động ngoài.
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTA<7>. Chức năng này bị cấm ở chế độ dao động ngoài.

Chú thích: DIG = Digital level output (đầu ra số); TTL = đệm đầu vào chuẩn TTL(Transistor-Transistor Logic ); ST = đệm đầu vào sử dụng Schmitt Trigger; ANA = vào/ra tương tự; x= không xác định; I=Input (vào); O=Output (ra).

Chú ý: Hai chân RA6 và RA7 còn phụ thuộc vào cấu hình bộ phát xung hệ thống.

#### ➤ Các thanh ghi liên quan đến PORTA

Các thanh ghi liên quan đến PORTA gồm 6 thanh ghi sau:

- PORTA: Thanh ghi dữ liệu PORTA.
- LATA: Thanh ghi chốt dữ liệu đầu ra của PORTA.
- TRISA: Thanh ghi lựa chọn hướng dữ liệu của PORTA (bit tương ứng trên thanh ghi đặt bằng ‘0’ thì chân tương ứng có chiều ra, bằng ‘1’ là chiều vào).

- ADCON1: Thanh ghi điều khiển A/D, thiết lập các chân vào/ra là số hay tương tự.
- ANSEL0, ANSEL1: Thanh ghi lựa chọn các kênh tín hiệu vào tương tự

*Bảng 2.2. Các thanh ghi liên quan đến PORTA.*

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTA	RA7 <sup>(1)</sup>	RA6 <sup>(1)</sup>	RA5	RA4	RA3	RA2	RA1	RA0
LATA	LATA7 <sup>(1)</sup>	LATA6 <sup>(1)</sup>	LATA Data Output Register					
TRISA	TRISA7 <sup>(1)</sup>	TRISA6 <sup>(1)</sup>	PORTA Data Direction Register					
ADCON1	VCFG1	VCFG0	—	FIFOEN	BFEMT	BFOVFL	ADPNT1	ADPNT0
ANSEL0	ANS7 <sup>(2)</sup>	ANS6 <sup>(2)</sup>	ANS5 <sup>(2)</sup>	ANS4	ANS3	ANS2	ANS1	ANS0
ANSEL1	—	—	—	—	—	—	—	ANS8 <sup>(2)</sup>

**Chú thích:** — = Không xác định, đọc là '0'. Không sử dụng

#### 2.4.2. PORTB

PORTB gồm 8 bit, tương ứng với 8 chân được ký hiệu từ RB0 đến RB7. Các chân của PORTB có thể đọc/ghi theo từng bit hoặc cả byte.

*Bảng 2.3. Chức năng các chân của PORTB.*

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
RB0/PWM0	RB0	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<0>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<0>. Có điện trở weak pull-up khi RBPU bị xóa. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	PWM0	0	O	DIG	Đầu ra kênh PWM0
RB1/PWM1	RB1	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<1>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<1>. Có điện trở weak pull-up khi RBPU bị xóa. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	PWM1	0	O	DIG	Đầu ra kênh PWM1
RB2/PWM2	RB2	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<2>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<2>. Có điện trở weak pull-up khi

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
					<i>RBPU</i> bị xóa. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	PWM2	0	O	DIG	Đầu ra kênh PWM2
RB3/PWM3	RB3	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<3>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<3>. Có điện trở weak pull-up khi <i>RBPU</i> bị xóa. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	PWM3	0	O	DIG	Đầu ra kênh PWM3
RB4/KBI0/PWM5	RB4	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<4>, không bị ảnh hưởng bởi đầu vào tương tự.
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<4>. Có điện trở weak pull-up khi <i>RBPU</i> bị xóa. Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	KBI0	1	I	TTL	Ngắt khi có sự thay đổi tại chân (Interrupt-on-change pin.)
	PWM5	0	O	DIG	Đầu ra kênh PWM3
RB5/KBI1/PWM4 /PGM	RB5	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<5>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<5>. Có điện trở weak pull-up khi <i>RBPU</i> bị xóa.
	KBI1	1	I	TTL	Ngắt khi có sự thay đổi tại chân (Interrupt-on-change pin.)
	PWM4 <sup>(3)</sup>	0	O	DIG	Đầu ra kênh PWM4
	PGM <sup>(2)</sup>	x	I	ST	Tín hiệu nối mạch nạp nối tiếp (ICSP™). Cho phép bởi bit cấu hình LVP. Tất cả các chức năng khác bị cấm.
RB6/KBI2/PGC	RB6	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<6>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<6>. Có điện trở weak pull-up khi <i>RBPU</i> bị xóa.
	KBI2	1	I	TTL	Ngắt khi có sự thay đổi tại chân (Interrupt-on-change pin)

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
	PGC	x	I	ST	Đầu vào xung từ (ICSP™) từ mạch nạp tích hợp ICSP và mạch nạp ICD. <sup>(1)</sup>
RB7/KBI3/PGD	RB7	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATB<7>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTB<7>. Có điện trở weak pull-up khi RBPU bị xóa.
	KBI3	1	I	TTL	Ngắt khi có sự thay đổi tại chân (Interrupt-on-change pin)
	PGD	x	O	DIG	Đầu ra dữ liệu nối tiếp từ ICSP và ICD. <sup>(1)</sup>
		x	I	ST	Đầu vào dữ liệu nối tiếp từ ICSP và ICD. <sup>(1)</sup>

### ➤ Các thanh liên quan đến PORTB

Có 7 thanh ghi được sử dụng để điều khiển và chọn chức năng cho PORTB:

- PORTB: Thanh ghi dữ liệu của PORTB.
- LATB: Thanh ghi chốt dữ liệu của PORTB.
- TRISB: Thanh ghi hướng dữ liệu của PORTB.
- INTCON: Thanh điều khiển ngắt.
- INTCON2: Thanh ghi điều khiển ngắt 2.
- INTCON3: Thanh điều khiển ngắt 3.

Bảng 2.4. Các thanh ghi liên quan đến PORTB.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
LATB	LATB Data Output Register							
TRISB	PORTB Data Direction Register							
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF

### 2.4.3. PORTC

PORTC có độ rộng 8 bit, tương ứng với 8 chân được ký hiệu từ RC0 đến RC7. Các chân của PORTC có hai chiều dữ liệu và người lập trình có thể đọc/ghi theo từng bit hoặc cả byte.

Bảng 2.5. Chức năng các chân của PORTC.

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
RC0/T1OSO/T1CKI	RC0	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATC<0>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTC<0>.
	T1OSO	x	O	ANA	Đầu ra bộ phát xung Timer1, chức năng này được cho phép khi cho phép bộ phát xung Timer1. Vô hiệu hóa chức năng vào/ra số.
	T1CKI	1	I	ST	Cấp xung cho Timer1/Timer3 trong chế độ đếm sự kiện
RC1/T1OSI/CCP2/ <u>FLTA</u>	RC1	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATC<1>.
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTC<1>.
	T1OSI	x	I	ANA	Đầu vào bộ dao động Timer1, chức năng này được cho phép khi cho phép bộ dao động Timer1. Cảm chức năng vào/ra số.
	CCP2	0	O	DIG	CCP2 của bộ so sánh và PWM; chức năng này ưu tiên hơn vào/ra dữ liệu
		1	I	ST	Đầu vào CCP2 capture.
	<u>FLTA</u>	1	I	ST	Fault Interrupt Input Pin A.
RC2/CCP1/ <u>FLTB</u>	RC2	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATC<2>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTC<2>.
	CCP1	0	O	DIG	CCP1 của bộ so sánh và PWM; chức năng này ưu tiên hơn vào/ra dữ liệu
		1	I	ST	Đầu vào CCP1 capture.
	<u>FLTB</u>	1	I	ST	Fault Interrupt Input Pin B.
RC3/T0CKI/T5CKI/ INT0	RC3	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATC<3>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTC<3>.
	T0CKI <sup>(1)</sup>	1	I	ST	Timer0 alternate clock input.
	T5CKI <sup>(1)</sup>	1	I	ST	Timer5 alternate clock input.
	INT0	1	I	ST	External Interrupt 0.
RC4/INT1/SDI/SDA	RC4	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATC<4>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTC<4>.
	INT1	1	I	ST	External Interrupt 0.
	SDI <sup>(1)</sup>	1	I	ST	Chân đầu vào dữ liệu của giao tiếp (SSP module).
	SDA <sup>(1)</sup>	0	O	DIG	Đầu ra dữ liệu của giao tiếp I2C; chức năng này ưu tiên hơn vào/ra dữ liệu.
		1	I	I2C	Đầu vào dữ liệu của giao tiếp I2C (SSP

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
					module).
RC5/INT2/SCK/SC L	RC5	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATC<5>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTC<5>.
	INT2	1	<b>I</b>	ST	External Interrupt 1.
	SCK <sup>(1)</sup>	0	<b>O</b>	DIG	Chân phát xung clock của SPI (SSP module); chức năng này ưu tiên hơn vào/ra dữ liệu.
		1	<b>I</b>	ST	Đầu vào xung của giao tiếp SPI (SSP module).
	SCL <sup>(1)</sup>	0	<b>O</b>	DIG	Chân phát xung (clock) của giao tiếp I2C (SSP module); chức năng này ưu tiên hơn vào/ra dữ liệu.
		1	<b>I</b>	I2C	Đầu vào xung (clock) của giao tiếp I2C (SSP module); chuẩn tín hiệu đầu vào phụ thuộc vào cấu hình MSSP.
RC6/TX/CK/ <b>SS</b>	RC6	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATC<6>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTC<6>.
	TX	0	<b>O</b>	DIG	Chân truyền dữ liệu nối tiếp không đồng bộ của module EUSART; chức năng này ưu tiên hơn vào/ra dữ liệu. Chức năng này phải được cấu hình hướng dữ liệu ra.
	CK	0	<b>O</b>	DIG	Chất phát xung (clock) đồng bộ nối tiếp của module EUSART; chức năng này ưu tiên hơn vào/ra dữ liệu.
		1	<b>I</b>	ST	Chất nhận xung (clock) đồng bộ nối tiếp của module EUSART.
	<b>SS</b>	1	<b>I</b>	ST	SPI slave select input.
RC7/RX/DT/SDO	RC7	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATC<7>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTC<7>..
	RX	1	<b>I</b>	ST	Chân nhận dữ liệu nối tiếp không đồng bộ của module EUSART.
	DT	0	<b>O</b>	DIG	Chân truyền dữ liệu nối tiếp đồng bộ của module EUSART; chức năng này ưu tiên hơn vào/ra dữ liệu.
		1	<b>I</b>	ST	Chân nhận dữ liệu nối tiếp đồng bộ của module EUSART. Phải được cấu hình dữ liệu hướng đầu vào
	SDO <sup>(1)</sup>	0	<b>O</b>	DIG	SPI data out; takes priority over port data.

### ➤ Các thanh ghi liên quan đến PORTC

Có 3 thanh ghi được sử dụng để điều khiển và chọn chức năng cho PORTC:

- PORTC: Thanh ghi dữ liệu của PORTC.
- LATC: Thanh ghi chốt dữ liệu của PORTC.
- TRISC: Thanh ghi hướng dữ liệu của PORTC.
- INTCON: Thanh ghi điều khiển ngắt.
- INCON2: Thanh ghi điều khiển ngắt 2.
- INCON3: Thanh ghi điều khiển ngắt 3.

Bảng 2.6. Các thanh ghi liên quan đến PORTC.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
LATC	LATC Data Output Register							
TRISC	PORTC Data Direction Register							
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIFF
INTCON2	RBPU	INTEGD0	INTEGD1	INTEGD2	—	TMR0IP	—	RBIP
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF

#### 2.4.4. PORTD

PORTD có độ rộng 8 bit, tương ứng với 8 chân được ký hiệu từ RD0 đến RD7. Các chân của PORTD có hai chiều dữ liệu và người lập trình có thể đọc/ghi theo từng bit hoặc cả byte.

Bảng 2.7. Özellik các chân của PORTD.

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
RD0/T0CKI/T5CKI	RD0	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATD<0>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTD<0>.
	T0CKI <sup>(1)</sup>	1	I	ST	Timer0 alternate clock input.
	T5CKI <sup>(1)</sup>	1	I	ST	Timer5 alternate clock input.
RD1/SDO	RD1	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATD<1>.
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTD<1>.
	SDO <sup>(1)</sup>	0	O	DIG	SPI data out; takes priority over port data.
RD2/SDI/SDA	RD2	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATD<2>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTD<2>.

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
	SDI <sup>(1)</sup>	1	<b>I</b>	ST	Chân đầu vào của dữ liệu của giao tiếp SPI (SSP module).
	SDA <sup>(1)</sup>	0	<b>O</b>	DIG	Đầu ra dữ liệu của giao tiếp I2C (SSP module); chức năng này được ưu tiên hơn chức năng vào/ra dữ liệu.
		1	<b>I</b>	I2C	Đầu vào dữ liệu của giao tiếp I2C (SSP module).
RD3/SCK/SCL	RD3	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATD<3>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTD<3>.
	SCK <sup>(1)</sup>	0	<b>O</b>	DIG	Chân phát xung clock của SPI (module SSP); chức năng này được ưu tiên hơn chức năng vào/ra dữ liệu.
		1	<b>I</b>	ST	Đầu vào xung clock SPI (SSP module).
	SCL <sup>(1)</sup>	0	<b>O</b>	DIG	Chân phát xung clock của I2C (SSP module); chức năng này được ưu tiên hơn chức năng vào/ra dữ liệu.
		1	<b>I</b>	I2C	Đầu vào xung clock I2C (SSP module); chuẩn tín hiệu đầu vào phụ thuộc vào cấu hình MSSP.
RD4/ <u>F<sub>LTA</sub></u>	RD4	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATD<4>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTD<4>.
	<u>F<sub>LTA</sub></u> <sup>(2)</sup>	1	<b>I</b>	ST	Fault Interrupt Input Pin A.
RD5/PWM4	RD5	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATD<5>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTD<5>.
	PWM4 <sup>(3)</sup>	0	<b>O</b>	DIG	Đầu ra PWM 4; chức năng này được ưu tiên hơn chức năng vào/ra dữ liệu.
RD6/PWM6	RD6	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit LATD<6>
		1	<b>I</b>	TTL	Chiều vào dữ liệu sử dụng bit PORTD<6>.
	PWM6 <sup>(3)</sup>	0	<b>O</b>	DIG	Đầu ra PWM 6; chức năng này được ưu tiên hơn chức năng vào/ra dữ liệu.
RD7/PWM7	RD7	0	<b>O</b>	DIG	Chiều ra dữ liệu sử dụng bit

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
					LATD<7>
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTD<7>.
	PWM7 <sup>(3)</sup>	0	O	DIG	Đầu ra PWM 7; chức năng này được ưu tiên hơn chức năng vào/ra dữ liệu.

### ➤ Các thanh liên quan đến PORTD

Có 5 thanh ghi được sử dụng để điều khiển và chọn chức năng cho PORTD:

- PORTD: Thanh ghi dữ liệu của PORTD.
- LATD: Thanh ghi chốt dữ liệu của PORTD.
- TRISD: Thanh ghi hướng dữ liệu của PORTD.

Bảng 2.8. Các thanh ghi liên quan đến PORTD.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
LATD	LATD Data Output Register							
TRISD	PORTD Data Direction Register							

### 2.4.5. PORTE

PORTE có độ rộng 4 bit, tương ứng với 4 chân được ký hiệu từ RE0 đến RE3. Chân MCLR/VPP/RE3 chỉ có chiều vào.

Bảng 2.9. Chức năng các chân của PORTE.

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
RE0/AN6	RE0	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATE<0>, không bị ảnh hưởng bởi đầu vào tương tự
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTE<0>, Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN6	1	I	ANA	Đầu vào kênh 6 của bộ biến đổi A/D. Mặc định khi cấu hình POR.
RE1/AN7	RE1	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATE<1>, không bị ảnh hưởng bởi đầu vào tương tự
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTE<1>, Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN7	1	I	ANA	Đầu vào kênh 7 của bộ biến đổi A/D. Mặc định khi cấu hình POR.

Chân	Chức năng	Thanh ghi TRIS	I/O	Kiểu I/O	Mô tả
RE2/AN8	RE2	0	O	DIG	Chiều ra dữ liệu sử dụng bit LATE<2>, không bị ảnh hưởng bởi đầu vào tương tự
		1	I	TTL	Chiều vào dữ liệu sử dụng bit PORTE<2>, Chức năng này sẽ bị cấm khi chức năng đầu vào tương tự được cho phép.
	AN8	1	I	ANA	Đầu vào kênh 8 của bộ biến đổi A/D. Mặc định khi cấu hình POR.
<i>MCLR/VPP/RE3<sup>(1)</sup></i>	<i>MCLR</i>	-	I	ST	Đầu vào Reset ngoài; cho phép khi bit cấu hình MCLRE được đặt bằng 1.
	VPP	-	I	ANA	Phát hiện điện áp cao, sử dụng cho ICSP
	RE3	- <sup>(2)</sup>	I	ST	Chiều vào dữ liệu sử dụng bit PORTE<3>; cho phép khi cấu hình bit MCLRE được đặt bằng 0.

### ➤ Các thanh liên quan đến PORTE

Có 4 thanh ghi được sử dụng để điều khiển và chọn chức năng cho PORTE:

- PORTE: Thanh ghi dữ liệu của PORTE.
- LATE: Thanh ghi chốt dữ liệu đầu ra của PORTE.
- TRISE: Thanh ghi hướng dữ liệu của PORTE, chân RE3 mặc định có chiều vào.
- ANSEL0, ANSEL1: Thanh ghi lựa chọn cổng vào tương tự.

Bảng 2.10. Các thanh ghi liên quan đến PORTE.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTE	—	—	—	—	RE3 <sup>(1)</sup>	RE2	RE1	RE0
LATE	—	—	—	—	—	LATE Data Output Register		
TRISE	—	—	—	—	—	PORTE Data Direction Register		
ANSEL0	ANS7 <sup>(2)</sup>	ANS6 <sup>(2)</sup>	ANS5 <sup>(2)</sup>	ANS4	ANS3	ANS2	ANS1	ANS0
ANSEL1	—	—	—	—	—	—	—	ANS8 <sup>(2)</sup>

## 2.5. BỘ ĐỊNH THỜI TIMER

Các bộ định thời (timer) được sử dụng rất rộng rãi trong các ứng dụng đo lường và điều khiển. Có thể coi một bộ định thời n bit là một bộ đếm n bit được tạo ra bởi n flip-flop mắc nối tiếp với nhau. Đầu vào của bộ định thời chính là đầu vào của flip-flop đầu tiên, đầu ra báo tràn (over flow) của bộ định thời phản ánh trạng thái tràn của nó. Đầu ra của các flip-flop phản ánh giá trị hiện thời của bộ đếm. Tuỳ thuộc vào ứng dụng, đầu vào bộ định thời có thể là nguồn xung lấy từ xung nhịp của vi điều khiển

hoặc nguồn xung từ bên ngoài đưa đến. Vi điều khiển PIC18F4431 có bốn bộ định thời Timer0, Timer1, Timer2, Timer5. Các bộ định thời được dùng để định khoảng thời gian hoặc đếm các sự kiện.

Trong các ứng dụng định khoảng thời gian, timer được lập trình sao cho sau một khoảng thời gian timer sẽ tràn. Khi tràn, cờ tràn của timer được đặt bằng 1. Căn cứ vào trạng thái của cờ tràn, người lập trình có thực hiện một thao tác điều khiển hoặc đo lường nào đó để tạo ra các ứng dụng đo lường/điều khiển theo thời gian.

Đếm sự kiện dùng để xác định số lần xảy ra của một sự kiện. Trong ứng dụng này người ta tìm cách quy các sự kiện thành sự chuyển mức từ ‘1’ xuống ‘0’ hoặc ngược lại trên các lối vào của các bộ timer. Sau mỗi sự chuyển mức, giá trị của timer sẽ thay đổi (tăng hoặc giảm đi 1 tùy theo chế độ đếm tiến hoặc lùi). Giá trị hiện thời của timer sẽ chính là số sự kiện đã xảy ra.

Dựa trên hai chức năng chính này, trong thực tế các bộ timer có thể được dùng để tạo xung, hẹn thời gian bật/tắt của một thiết bị, điều chế độ rộng xung (PWM – Pulse Width Modulation), đo tần số, đếm sản phẩm, tạo tốc độ baud cho PORT nối tiếp của vi điều khiển.

### 2.5.1. Timer0

Timer0 là bộ timer 16 bit tuy nhiên bộ timer có thể hoạt động được cả ở chế độ 8 bit. Timer0 có thể dùng làm bộ đếm hoặc định thời với nguồn xung từ bộ dao động trên chip hoặc từ ngoài đưa vào qua chân T0CLKI.

#### ➤ Các thanh ghi của Timer0

- Thanh ghi điều khiển Timer0: T0CON

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T016BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Bit điều khiển Bật/Tắt Timer.

1 = Cho phép Timer0

0 = Dừng Timer0

bit 6 **T016BIT:** Bit lựa chọn 8-bit /16-bit của Timer0

1 = Timer0 được cấu hình là bộ đếm 8-bit

0 = Timer0 được cấu hình là bộ đếm 16-bit

bit 5 **T0CS:** Bit lựa chọn nguồn xung cấp cho Timer0

1 = Nguồn xung từ chân T0CKI

0 = Nguồn xung hệ thống (CLKO)

bit 4 **T0SE:** Bit lựa chọn sườn xung đếm cho Timer0

- 1= Lựa chọn sườn âm trên chân T0CKI  
 0 = Lựa chọn sườn dương trên chân T0CKI
- bit 3 **PSA:** Bit thiết lập bộ chia tần đầu vào  
 1 = Xung cấp vào Timer0 không qua bộ chia tần.  
 0 = Xung cấp vào Timer0 qua bộ chia tần (Prescaler).
- bit 2 **T0PS<2 :0>:** Bit lựa chọn hệ số chia tần  
 111 = 1:256  
 110 = 1:128  
 101 = 1:64  
 100 = 1:32  
 011 = 1:16  
 010 = 1:8  
 001 = 1:4  
 000 = 1:2

- Thanh ghi chứa byte thấp của Timer0: TMR0L (8 bit, không định địa chỉ bit)
- Thanh ghi chứa byte cao của Timer0: TMR0H (8 bit, không định địa chỉ bit)
- Thanh ghi điều khiển ngắt : INTCON (xem phần ngắt và xử lý ngắt)

*Các thanh ghi liên quan đến Timer0:*

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TMR0L	Timer0 Register Low Byte							
TMR0H	Timer0 Register High Byte							
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
T0CON	TMR0ON	T016BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
TRISA	TRISA7 <sup>(1)</sup>	TRISA6 <sup>(1)</sup>	PORTA Data Direction Register					

## ➤ Chế độ hoạt động của Timer0

### Chế độ 8 bit

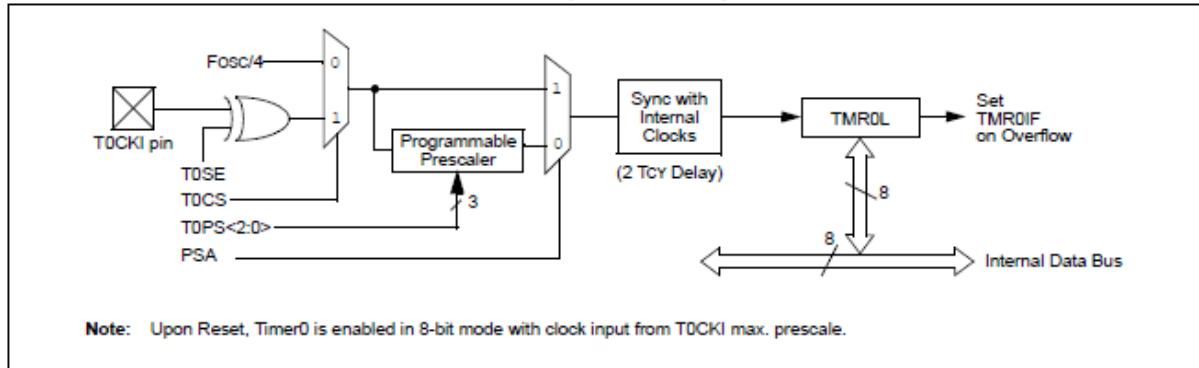
Ở chế độ 8 bit, giá trị của Timer0 chỉ chứa trong thanh ghi TMR0L, khi tràn (khi có sự chuyển giá trị đếm từ 255 sang 0), cờ ngắt tràn TMR0IF được đặt bằng 1.

Nguồn xung cấp cho Timer0 có thể lấy từ 2 nguồn:

- Khi bit T0CS=0, xung được lấy từ bộ dao động trên chíp, tần số của xung bằng  $\frac{1}{4}$  tần số của bộ dao động thạch anh được sử dụng (Fosc/4).
- Khi bit T0CS=1, xung được lấy từ ngoài đưa vào qua chân T0CLKI (T0CLKI pin). Trong trường hợp này Timer0 thường được sử dụng để đếm xung đưa vào từ chân T0CLKI.

Bit PSA trong thanh ghi T0CON được dùng để lựa chọn việc sử dụng/không sử dụng bộ chia tần số:

- Khi PSA=1, giá trị của Timer0 sẽ tăng lên 1 đơn vị sau mỗi chu kỳ lệnh (tương đương 4 chu kỳ thạch anh) hoặc sau 1 chu kỳ xung ngoài đưa đến chân T0CLKI.



Hình 2.8. Mô tả hoạt động Timer0 ở chế độ 8 bit

- Khi PSA=0, xung từ 2 nguồn trên sẽ qua bộ chia tần số với 8 hệ số chia từ 2 đến 256 tùy thuộc vào giá trị của các bit T0PS2, T0PS1, T0PS0 trong thanh ghi T0CON.

Chẳng hạn khi  $T0PS2\ T0PS1\ T0PS0 = 0\ 1\ 1$ , hệ số chia sẽ là 1/16, khi đó sau 16 chu kỳ xung ngoài đưa đến chân T0CLKI hoặc sau 16 chu kỳ lệnh giá trị trong TMR0L mới tăng lên 1 đơn vị.

Bit T0SE dùng để chọn kiểu tác động của xung ngoài:

- Khi T0SE=0, giá trị của TMR0L sẽ tăng 1 đơn vị sau mỗi sườn xung dương đưa tới chân T0CLKI.

- Khi T0SE=1, giá trị của TMR0L sẽ tăng 1 đơn vị sau mỗi sườn xung âm đưa tới chân T0CLKI.

Khi sử dụng nguồn xung ngoài (sử dụng Timer0 như bộ đếm), nguồn xung này cần có một quá trình đồng bộ với xung nội (internal clock), quá trình này thường mất 2 chu kỳ lệnh (tương đương 8 chu kỳ thạch anh).

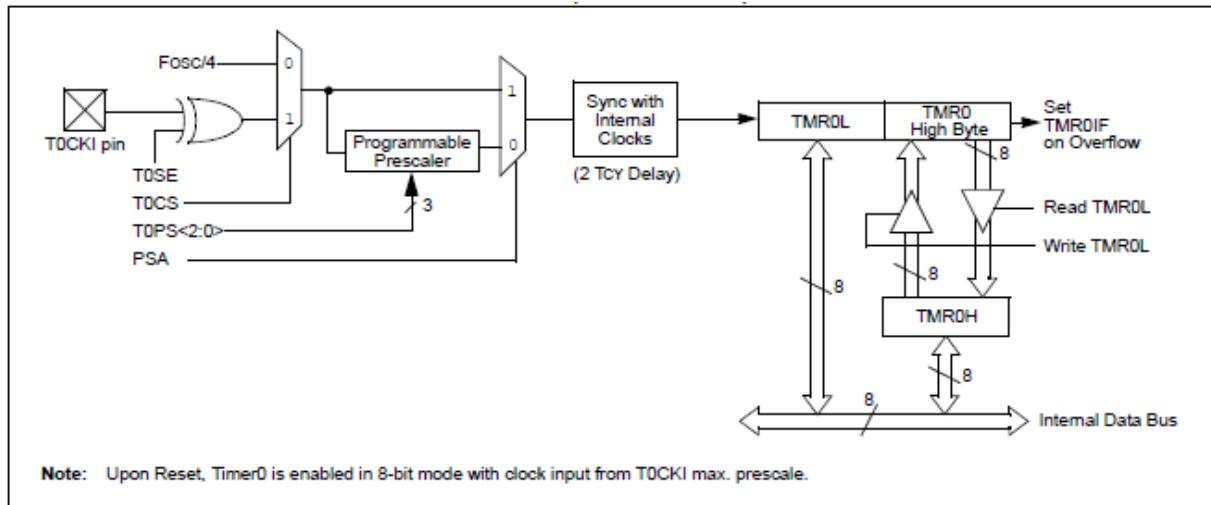
### Chế độ 16 bit

Ở chế độ 16 bit, giá trị của Timer0 được chứa trong thanh ghi TMR0L (byte thấp) và TMR0 High Byte (byte cao), TMR0H đóng vai trò là bộ đếm của byte cao trong quá trình ghi đọc. Khi tràn (khi có sự chuyển giá trị đếm từ 65535 sang 0), cờ ngắt tràn TMR0IF được đặt bằng 1.

Việc lựa chọn nguồn xung, kiểu tác động của xung và bộ chia tần hoàn toàn giống chế độ 8 bit.

Khi ghi giá trị cho Timer0, byte thấp được ghi vào TMR0L còn byte cao sẽ được ghi từ TMR0H ngay khi byte thấp được ghi. Khi đọc giá trị của Timer0, byte

thấp được đọc từ TMR0L, byte cao sẽ xuất hiện trong TMR0H ngay khi đọc byte thấp. Với thiết kế này byte thấp và byte cao của Timer được ghi/đọc đồng thời.



Hình 2.9. Mô tả hoạt động Timer0 ở chế độ 16 bit

### ➤ Ngắt Timer0.

Ngắt Timer0 xảy ra khi Timer0 tràn. Ở chế độ đếm 8 bit sự kiện tràn xảy ra khi có sự chuyển số đếm từ FFH sang 00H và FFFFH sang 0000H ở chế độ 16 bit. Khi tràn cờ ngắt tương ứng của Timer0 (TMR0IF) được thiết lập. Ngắt Timer0 được cho phép ngắt bởi bit TMR0IE.

### 2.5.2. Timer1

Timer1 là bộ Timer 16 bit. Ngoài chức năng đếm và định thời thông thường của một bộ Timer; Timer1 còn có thể được dùng như một bộ phát xung thứ hai, cấp nguồn xung đồng hồ cho toàn bộ hệ thống; Reset từ module CCP Special Event Trigger.

### ➤ Các thanh ghi của Timer1

- Thanh ghi điều khiển Timer1: T1CON

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON
bit 7	bit 0						

bit 7    **RD16:** Bit lựa chọn chế độ ghi/đọc Timer1

1 = Ghi/đọc 1 lần 16 bit.

0 = Ghi/đọc 2 lần mỗi lần 8 bit.

bit 6    **T1RUN:** Bit cho phép hệ thống lấy xung từ Timer1

1 = Hệ thống hoạt động bằng nguồn xung cấp từ Timer1

0 = Hệ thống hoạt động bằng nguồn xung khác

bit 5-4    **T1CKPS1:T1CKPS0:** Các bit đặt hệ số chia tần số của xung cấp cho Timer1

11 = Hệ số chia là 1:8

10 = Hệ số chia là 1:4

01 = Hệ số chia là 1:2

00 = Hệ số chia là 1:1

bit 3 **T1OSCEN:** Bit cho phép/cấm chức năng phát xung cho hệ thống

1 = Cho phép

0 = Cấm

bit 2 **T1SYNC:** Bit lựa chọn sự đồng bộ giữa xung ngoài cấp cho Timer1 và xung trên chip.

Khi bit TMR1CS = 1 (xung clock nội):

1 = Không đồng bộ

0 = Đồng bộ xung ngoài với xung trên chip

Khi bit TMR1CS = 0 (xung clock ngoại):

Bit T1SYNC không có giá trị. (khi đó Timer1 sử dụng nguồn xung nội).

bit 1 **TMR1CS:** Bit lựa chọn nguồn xung cấp cho Timer1

1 = Timer1 được cấp xung từ ngoài qua chân RC0/T1OSO/T13CKI

0 = Timer1 được cấp xung nội (tần số bằng FOSC/4)

bit 0 **TMR1ON:** Bit điều khiển hoạt động của Timer1

1 = Timer1 hoạt động

0 = Dừng Timer1

- Thanh ghi chứa giá trị đếm byte thấp của Timer1: TMR1L

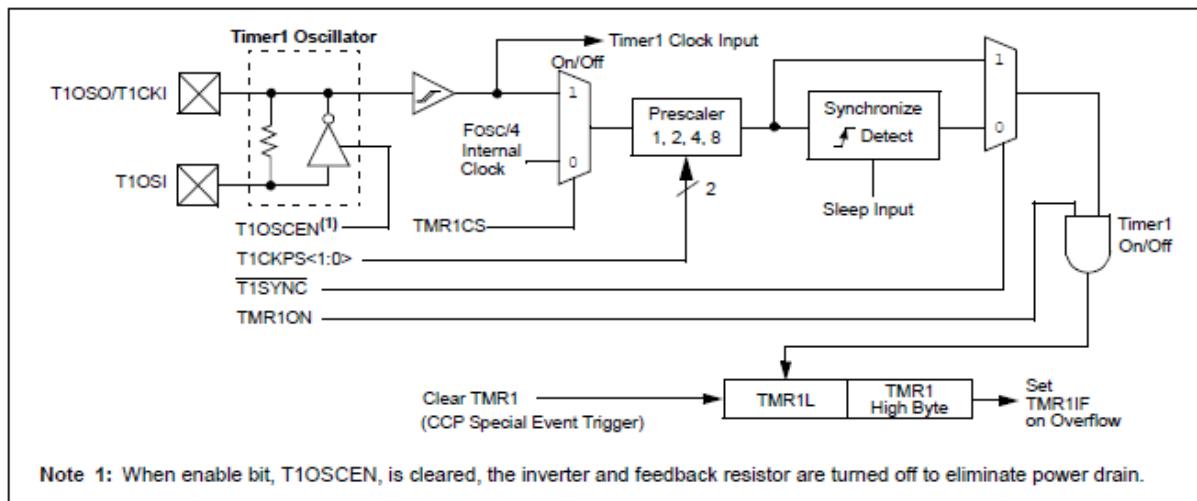
- Thanh ghi chứa giá trị đếm byte cao của Timer1: TMR1H

*Các thanh ghi liên quan đến TIMER1*

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	—	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
TMR1L	Timer1 Register Low Byte							
TMR1H	Timer1 Register High Byte							
T1CON	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

## ➤ Chế độ hoạt động của Timer1

### 1. Chế độ ghi/đọc 2 lần 8bit



Hình 2.10. Timer1 hoạt động ở chế độ ghi/đọc 2 lần 8bit

Trong chế độ ghi/đọc 2 lần 8bit, giá trị của Timer1 được chứa trong 2 thanh ghi TMR1H (byte cao) và TMR1L (byte thấp). Quá trình ghi đọc được chia làm 2 lần: Ghi/đọc thanh ghi TMR1H và ghi/đọc thanh ghi TMR1L. Khi có sự chuyển số đếm từ FFFFH sang 0 (tràn), cờ tràn của Timer1 (bit TMR1IF) được đặt bằng 1 và xảy ra ngắt Timer1 nếu nguồn ngắt này được cho phép. Timer1 được điều khiển hoạt động/dừng bởi bit TMR1ON.

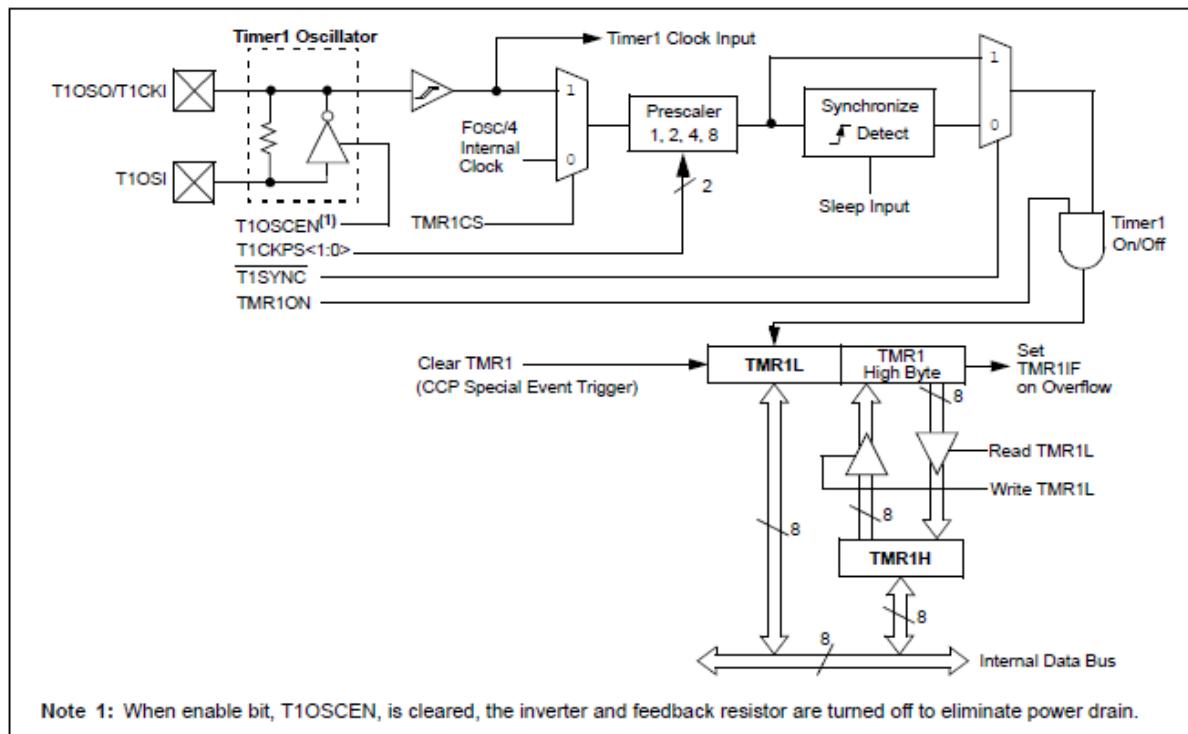
Nguồn xung cấp cho Timer1 có thể lấy từ 1 trong 3 cách sau:

- Lấy xung từ bộ dao động trên chíp, xung này có tần số bằng Fosc/4. Khi đó bit TMR1CS phải được đặt bằng 0.
- Lấy xung từ một nguồn xung bên ngoài đưa vào qua chân T13CKI. Khi đó bit T1OSCEN phải được đặt bằng 0 và bit TMR1CS phải được đặt bằng 1. Trong trường hợp này Timer1 thường được dùng như một bộ đếm (đếm số xung đưa vào qua chân T13CKI).
- Lấy xung từ mạch tạo dao động bao gồm bộ tạo dao động của Timer1. Khi đó bit T1OSCEN và bit TMR1CS phải được đặt bằng 1.

Cả 3 nguồn xung này đều được đưa qua bộ chia tần số với các hệ số chia là 1,2,4,8 (tùy thuộc vào giá trị của các bit T1CKPS1, T1CKPS0).

Khi sử dụng xung từ ngoài cấp cho Timer1 thì có thể lựa chọn việc sử dụng bộ đồng bộ để đồng bộ hóa nguồn xung này với xung của hệ thống bằng cách đặt bit T1SYNC bằng 0 (=1 nếu không sử dụng bộ đồng bộ).

## 2. Chế độ ghi/đọc 1 lần 16bit

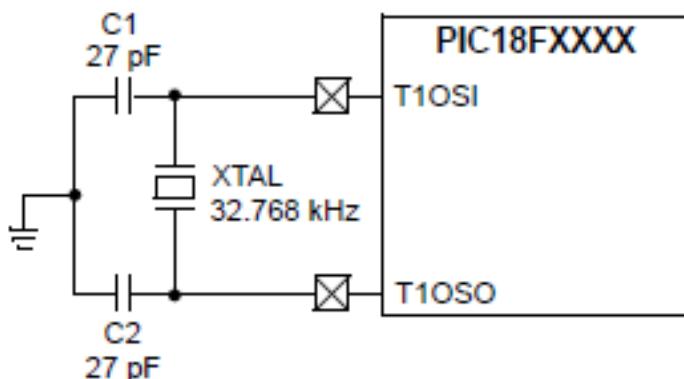


Hình 2.11. Timer1 hoạt động ở chế độ ghi/đọc 1 lần 16bit

Trong chế độ ghi/đọc 16 bit 1 lần, byte thấp của Timer1 vẫn là thanh ghi TMR1L, thanh ghi TMR1H trở thành bộ đếm của byte cao. Khi đọc TMR1L, giá trị của byte cao sẽ tự động chuyển vào TMR1H, khi ghi TMR1L, giá trị của TMR1H sẽ tự động chuyển vào byte cao. Hai hoạt động ghi/đọc TMR1L và chuyển giá trị giữa byte cao của Timer1 với TMR1H diễn ra ở cùng thời điểm.

Điều khiển sự hoạt động của Timer1 và các nguồn xung cấp cho Timer1 trong chế độ này hoàn toàn giống chế độ ghi/đọc 2 lần 8 bit.

### 3. Chế độ phát xung cho toàn hệ thống.



Hình 2.12. Timer1 hoạt động như một bộ phát xung cho hệ thống

Thông thường PIC18F4431 dùng nguồn dao động từ bộ dao động thạch anh mắc trên 2 chân OSC1 và OSC2. Tuy nhiên khi nguồn dao động này bị hỏng, bộ vi điều khiển vẫn có thể hoạt động nhờ việc chuyển sang dùng nguồn dao động thứ hai, khi

đó bit T1RUN và bit T1OSCEN phải được đặt bằng 1 để cho phép hệ thống dùng nguồn xung từ Timer1 và cho phép mạch phát xung trên Timer1 hoạt động.

### ➤ Ngắt Timer1.

Ngắt Timer1 được cho phép bởi bit TMR1IE, được đặt mức ưu tiên cao/thấp bởi bit TMR1IP. Ngắt Timer1 xảy ra khi Timer1 tràn, khi đó cờ ngắt tràn TMR1IF được đặt bằng 1.

### 2.5.3. Timer2

Timer2 có thanh ghi Timer 8-bit (TMR2), thanh ghi Period 8-bit (PR2), cả hai thanh khi có thể đọc và viết, bộ chia prescaler (1:1, 1:4, 1:16), postscaler (1:1 to 1:16). Ngắt Timer1 khi giá trị thanh ghi TMR2 bằng PR2. Timer2 còn dùng để tạo xung clock cho module và tạo tín hiệu điều chế độ rộng xung PWM

### ➤ Các thanh ghi của Timer2

- Thanh ghi điều khiển Timer1: T2CON

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 sử dụng:** Read as ‘0’

bit 6-3 **TOUTPS<3:0>:** Lựa chọn Output Postscale cho Timer2

0000 = 1:1 Postscale

0001 = 1:2 Postscale

•

•

•

1111 = 1:16 Postscale

bit 2 **TMR2ON:** Bit điều khiển Timer2 hoạt động

1 = Timer2 là on

0 = Timer2 là off

bit 1-0 **T2CKPS<1:0>:** Bit chọn Clock Prescale cho Timer2

00 = Prescaler is 1

01 = Prescaler is 4

1x = Prescaler is 16

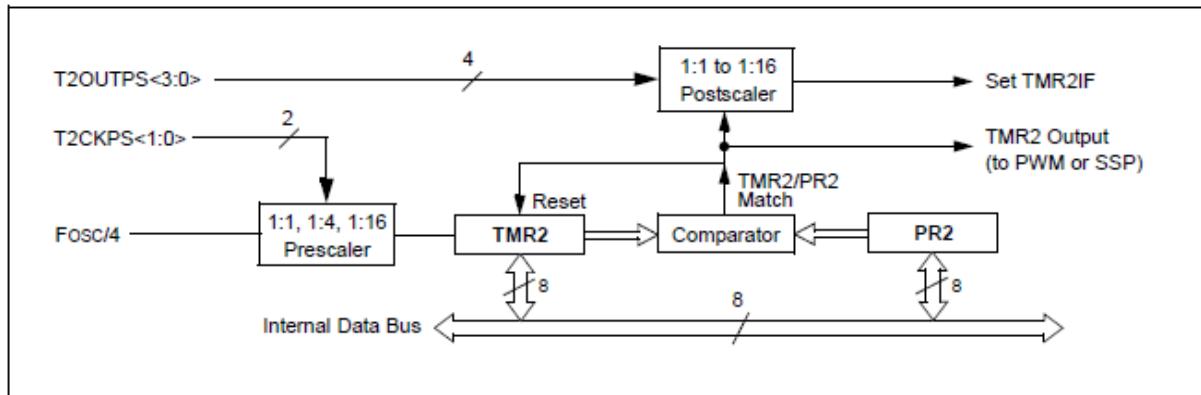
Các thanh ghi liên quan đến Timer2:

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	—	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
TMR2	Timer2 Register							
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
PR2	Timer2 Period Register							

### ➤ Ngắt Timer2

Ngắt Timer2 xảy ra khi giá trị thanh ghi TMR2 bằng giá trị thanh ghi PR2, cò báo ngắt là TMR2IF (PIR1<1>), bit cho phép ngắt là TMR2IE (PIE1<1>).

### ➤ Chế độ hoạt động Timer2



Hình 2.13. Chế độ hoạt động Timer2

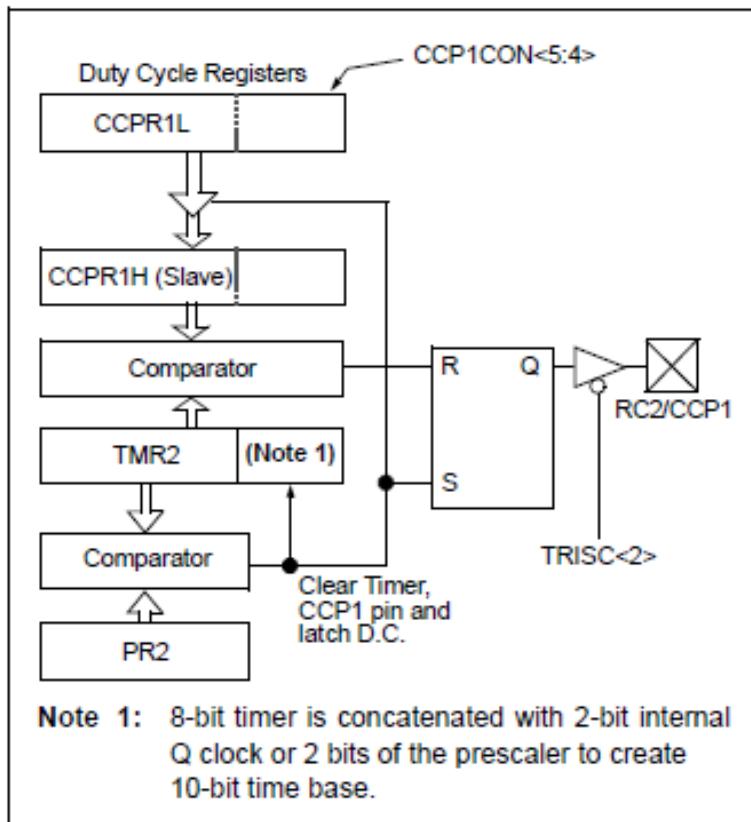
## 2.6. KHÓI CCP (Capture – Compare – PWM)

Vì điều khiển PIC18F4431 có 2 bộ CCP (Capture/Compare/PWM). Mỗi bộ CCP có thể hoạt động ở chế độ chụp (Capture), so sánh (Compare) hoặc điều chỉnh chế độ rộng xung PWM. Bộ CCP sử dụng thanh 16 bit để chụp, so sánh và tạo độ rộng của xung PWM. Trong phần này sẽ đề cập đến hoạt động của CCP ở chế độ tạo xung PWM.

### 2.6.1. Chế độ PWM

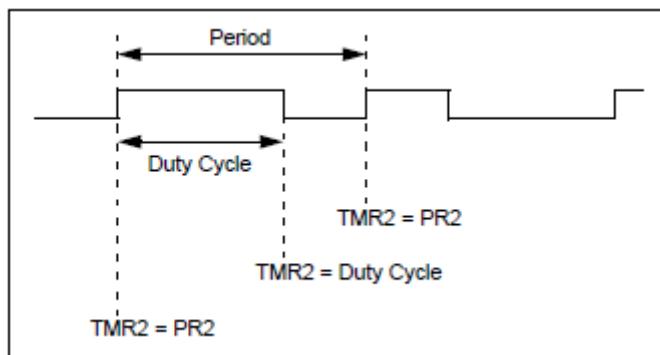
Trong chế độ PWM, xung PWM được tạo ra trên các chân CCPx với độ phân giải lên đến 10 bit. Chân CCP1 được ghép với RC2 (RC2/CCP1) và chân CCP2 có thể được ghép với RC1 (RC1/CCP2) hoặc RB3 (RB3/CCP2). Các chân CCPx cần phải được chọn là hướng chiều ra.

Trên hình 1.26 là sơ đồ khái của chế độ PWM, trong chế độ này CCP sử dụng bộ Timer2 để tạo xung PWM. Thanh ghi PR2 sẽ được so sánh với thanh ghi đếm TMR2 để tạo chu kỳ (Period) và 10 bit CCPRXL:CCPXCON<5:4> được so sánh với thanh ghi TMR2 để tạo độ rộng xung (Duty Cycle).



Hình 2.14. Sơ đồ khối của CCP ở chế độ PWM.

Trên hình 1.27 là dạng xung PWM trên chân CCPx.



Hình 2.15. Dạng xung PWM.

Khi TMR2 bằng PR2 :

- TMR2 sẽ được xóa.
- Chân CCPx sẽ được thiết lập ở mức ‘1’ (nếu độ rộng xung là 0%, chân CCPx sẽ không được thiết lập).
- Giá trị thiết lập độ rộng xung trong thanh ghi CCPxL được nạp vào thanh ghi CCPxH.

Khi TMR2 bằng CCPRxL:CCPxCON<5:4> : Chân CCPx sẽ được xóa về ‘0’.

## 2.6.2. Các thanh ghi liên quan

### - Thanh ghi điều khiển CCPx: CCPxCON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Không được sử dụng**: đọc sẽ được ‘0’

bit 5-4 **DCxB1:DCxB0**: Bit lựa chọn độ rộng xung 1:0 của chế độ PWM.

Chế độ Capture: Không sử dụng.

Chế độ Compare: Không sử dụng.

Chế độ PWM:

Là 2 bit thấp DCxB1: DCxB0 của thanh ghi lựa chọn độ rộng xung cho PWM.

bit 3-0 **CCPxM3:CCPxM0**: Bit lựa chọn chế độ hoạt động cho bộ CCPx  
0000 = Cảm CCPx hoạt động.

0001: 1011 = Không được sử dụng ở chế độ này.

11xx = Chế độ PWM.

### - Thanh ghi chu kỳ (Period) : PR2

PR2 là thanh ghi 8 bit của bộ Timer2, thanh ghi này được sử dụng để tạo chu kỳ cho xung PWM.

### - Thanh ghi độ rộng xung (Duty Cycle) : CCPRx

CCPRx là thanh ghi 8 bit của CCPx, thanh ghi này chứa 8 bit cao DCxB9:DCxB2 sử dụng để tạo độ rộng xung(2 bit thấp trong thanh CCPxCON).

### Các thanh ghi liên quan đến chế độ PWM của bộ CCPx:

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	—	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
TRISC	PORTC Data Direction Register							
TMR2	Timer2 Register							
PR2	Timer2 Period Register							
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
CCPR1L	Capture/Compare/PWM Register 1 Low Byte							
CCPR1H	Capture/Compare/PWM Register 1 High Byte							
CCP1CON	—	—	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
CCPR2L	Capture/Compare/PWM Register 2 Low Byte							
CCPR2H	Capture/Compare/PWM Register 2 High Byte							
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0

### 2.6.3. Sử dụng các bộ PWM

Để sử dụng các bộ PWM, cần thực hiện các công việc sau:

- Chọn chế độ OSC (các ví dụ trong tài liệu này thường chọn chế độ HS).
- Lựa chọn chân RCx có chiều ra, tín hiệu sô.
- Đặt giá trị cho thanh ghi CCPxCON, cấu hình mô-đun CCPx là bộ PWM.
- Thiết lập hệ số chia tần (Prescale) cho bộ Timer2, cho phép bộ Timer2 bằng cách ghi giá trị vào T2CON.
- Thiết lập chu kỳ.
- Thiết lập độ rộng xung dương

#### 1. Thiết lập chu kỳ

Công thức tính chu kỳ:

$$\text{PWM Period} = [(PR2) + 1] * 4 * \text{TOSC} * (\text{TMR2 Prescale Value})$$

Trong đó:

PWM Period: Chu kỳ xung PWM cần tạo

TOSC: Chu kỳ thạch anh

TMR2 Prescale Value: Hệ số chia tần của Timer 2.

Theo công thức trên, ngoài Tosc do phần cứng quy định, chu kỳ xung PWM có thể thay đổi nhờ giá trị nạp vào PR2 và hệ số chia tần của Timer 2.

Ví dụ:

PR2=199; // nạp cho PR2 giá trị bằng 99.

T2CONbits.T2CKPS1=0; // Hệ số chia tần của Timer 2 bằng 1

T2CONbits.T2CKPS0=0;

Khi đó, PWM Period = [199 + 1] \* 4 \* TOSC \* 1. Nếu sử dụng thạch anh có tần số 8Mhz, bộ tạo dao động hoạt động ở chế độ HS, PWM Period=  $800/(8 \cdot 10^{-6}) = 100$  uS, tương đương tần số 10Khz.

#### 2. Thiết lập độ rộng xung

Công thức tính độ rộng xung dương:

$$\text{PWM Duty Cycle} = (\text{CCPRxL:CCPxCON<5:4>} * \text{Tosc} * (\text{TMR2 Prescale Value}))$$

Trong đó:

PWM Duty Cycle: Độ rộng của xung dương.

CCPRxL:CCPxCON<5:4> : 10 bit chứa giá trị của bộ đếm (CCPRxL chứa 8 bit cao và CCPxCON <5:4> chứa 2 bit thấp).

Theo công thức trên, ngoài Tosc do phần cứng quy định, độ rộng của xung dương có thể thay đổi nhờ giá trị nạp vào CCPRxL:CCPxCON<5:4> và hệ số chia tần của Timer 2.

Ví dụ: Với bộ CCP1, các câu lệnh sau:

```
T2CONbits.T2CKPS1=0; //TMR2 Prescaler Value =1  
T2CONbits.T2CKPS0=0;  
PR2=199; // PWM Period = 100uS (F=10Khz)  
CCPR1L=400/4;//Đưa 8bit cao chúa giá trị bộ đếm vào CCPR1L  
CCP1CON|=(400%4)<<4; // Đưa 2 bit thấp vào CCP1CON<5:4> sẽ  
tạo ra xung PWM có chu kỳ 100uS (PWM Duty Cycle=100uS), độ rộng  
xung dương (PWM Duty Cycle) =400/8= 50 uS. Khi đó D=50% (Nếu sử  
dụng thạch anh có tần số 8Mhz, bộ tạo dao động hoạt động ở chế độ  
HS).
```

## 2.7. BỘ ĐIỀU KHIỂN ADC

Các bộ vi điều khiển chỉ có khả năng xử lý và tính toán được với tín hiệu số, trong khi các đại lượng vật lý cần đo như điện áp, dòng điện, nhiệt độ, áp suất v.v... là các tín hiệu tương tự. Vì vậy các tín hiệu tương tự cần phải chuyển đổi thành tín hiệu số bằng bộ biến đổi tương tự - số trước khi đưa vào vi điều khiển.

Bộ chuyển đổi tương tự - số (ADC- Analog to Digital Converter) có chức năng chuyển đổi mức điện áp tương tự (đầu vào) thành tín hiệu số tương ứng (đầu ra).

Các thông số kỹ thuật quan trọng của ADC:

- Độ phân giải (resolution): độ phân giải của ADC là số bit của tín hiệu đầu ra số và được gọi tắt là n. Với độ phân giải là n bit, bộ biến đổi tương tự - số sẽ có  $2^n$  mức điện áp khác nhau tương ứng với  $2^n$  giá trị số tương ứng ở đầu ra. Độ phân giải n càng lớn thì sai số lượng tử sẽ càng nhỏ.

- Tốc độ chuyển đổi: tốc độ chuyển đổi của ADC được xác định bằng thời gian hoàn thành một lần chuyển đổi. Thời gian này được tính từ khi có lệnh điều khiển chuyển đổi đến khi có tín hiệu số đầu ra ổn định.

- Điện áp tham chiếu (reference voltage): điện áp tham chiếu VREF là điện áp giới hạn điện áp tương tự đầu vào của bộ biến đổi ADC. Điện áp tham chiếu mức cao được ký hiệu VREF+, điện áp tham chiếu mức thấp được ký hiệu là VREF-. Bộ biến đổi ADC có thể chuyển đổi tín hiệu tương tự trong giải từ VREF- đến VREF+, các bộ biến đổi ADC có thể tùy chỉnh được điện áp tham chiếu cho phù hợp để đạt được độ chính xác là cao nhất.

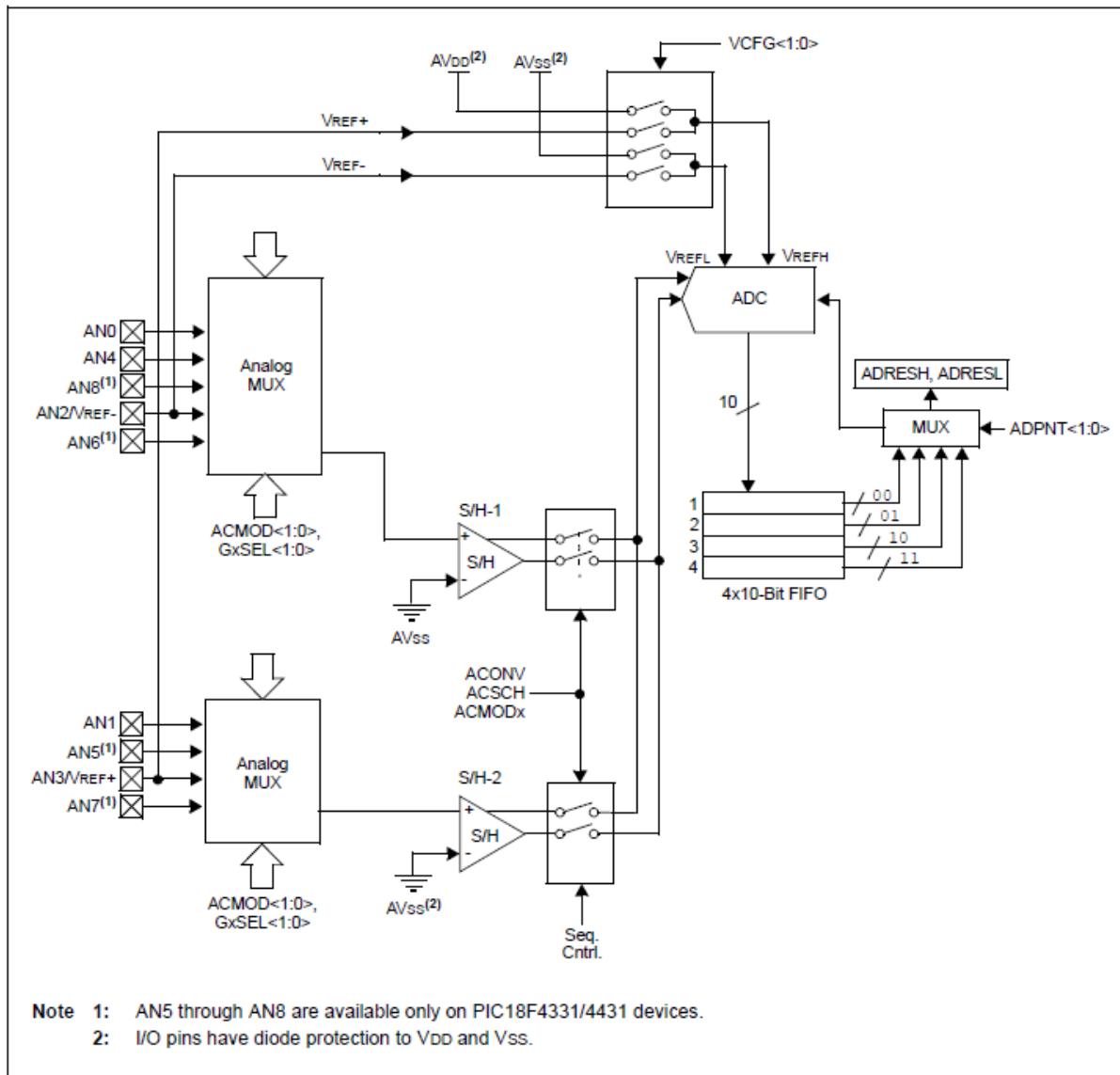
Vì điều khiển PIC18F4431 được tích hợp bộ chuyển đổi ADC với 9 kênh đầu vào tương tự từ AN0 đến AN8, độ phân giải là 10 bit, và được chia thành 4 nhóm.

Nhóm A (Group A): AN0, AN4.

Nhóm B (Group B): AN1, AN5.

Nhóm C (Group C): AN2, AN6.

Nhóm D (Group D): AN3, AN7.



Hình 2.16. Cấu trúc bộ biến đổi AD của PIC18F4431

Việc chuyển đổi tín hiệu từ dạng tương tự sang dạng số bao gồm hai giai đoạn: giai đoạn lấy mẫu tín hiệu (**sampling**) và giai đoạn chuyển đổi (**conversion**). Thời điểm chấm dứt lấy mẫu (và bắt đầu chuyển đổi) có thể do người dùng xác định. Như vậy thời gian lấy mẫu là khác nhau với các thiết lập khác nhau. Giai đoạn chuyển đổi dữ liệu cần có 12 xung clock cho module ADC, với một chu kỳ xung clock  $T_{AD}$  có thể được chọn từ phần mềm.

Module ADC của PIC18F4431 có 2 bộ khuếch đại S/H (Sample and Hold), được đánh địa chỉ là kênh 1 đến kênh 2. Bạn có thể chỉ dùng kênh 1, hay dùng kênh 2, và cũng có thể dùng cả 2 kênh cho việc thu thập dữ liệu. Bộ đếm của module ADC có thể chứa được tối đa 4 kết quả.

### **2.7.1. Các thanh ghi liên quan**

Module A/D có 9 thanh ghi:

- A/D Result High Register (ADRESH)
  - A/D Result Low Register (ADRESL)
  - A/D Control Register 0 (ADCON0)
  - A/D Control Register 1 (ADCON1)
  - A/D Control Register 2 (ADCON2)
  - A/D Control Register 3 (ADCON3)
  - A/D Channel Select Register (ADCHS)
  - Analog I/O Select Register 0 (ANSEL0)
  - Analog I/O Select Register 1 (ANSEL1)

#### ➤ **Thanh ghi ADCON0: Thanh ghi điều khiển 0**

bit 7-6      **Không sử dụng:** Đọc là ‘0’

bit 5 **ACONV**: Bit lựa chọn chế độ Auto-Conversion Continuous Loop hay Single-Shot

## 1 = Chế độ Continuous Loop

0 = Chế độ Single-Shot

bit 4 ACSCH: Bit lựa chọn chế độ Auto-Conversion Single hay Multi-Channel

## 1 = Chế độ Multi-Channel

0 = Chế độ Single Channel

bit 3-2      **ACMOD<1:0>**: Bit chon ché đô Auto-Conversion Mode Sequence

If ACSCH = 1:

00 = Trình tự chế độ 1 (SE)

## Mẫu thí 1: Group A

## Mẫu thử 2: Group B

Trình tự chế độ 2 (S)

### Mô hình 1: Group A

#### **REFERENCES**

Mẫu thứ 2: Group B

Mẫu thứ 3: Group C

Mẫu thứ 4: Group D

10 = Simultaneous Mode 1 (STNM1); 2 mẫu được lấy đồng thời:

Mẫu thứ 1: Group A và Group B

11 = Simultaneous Mode 2 (STNM2); 2 mẫu được lấy đồng thời:

Mẫu thứ 1: Group A và Group B

Mẫu thứ 2: Group C và Group D

If ACSCH = 0, Tư động chuyển đổi kênh đơn theo trình tự:

00 = Kênh đơn chế độ 1 (SCM1); Group A được lấy mẫu và chuyển đổi

01 = Kênh đơn chế độ 2 (SCM2); Group B được lấy mẫu và chuyển đổi

10 = Kênh đơn chế độ 3 (SCM3); Group C được lấy mẫu và chuyển đổi

11 = Kênh đơn chế độ 4 (SCM4); Group D được lấy mẫu và chuyển đổi

bit 1 **GO/DONE** : Bit báo trạng thái chuyển đổi A/D

1 = Đang chuyển đổi A/D. Cài đặt bit này khi bắt đầu chu kỳ chuyển đổi A/D. Nếu chế độ Auto-Conversion Single-Shot được kích hoạt (ACONV = 0), thì bit này sẽ tự động được xóa bởi phần cứng khi chuyển đổi A/D (single hay multi-channel tùy thuộc vào cài đặt ACMOD) đã hoàn thành. Nếu chế độ Auto-Conversion Continuous Loop được kích hoạt (ACONV = 1), bit này được đặt sau khi người dùng hoặc tín hiệu trigger đã đặt nó (continuous conversions). Người dùng có thể xóa thủ công để dừng việc chuyển đổi.

0 = A/D conversion / multiple conversions completed/not in progress

bit 0 **ADON**: A/D On bit

1 = Chuyển đổi A/D được kích hoạt

0 = A/D Converter không được kích hoạt.

#### ➤ Thanh ghi ADCON1: Thanh ghi điều khiển 1

R/W-0	R/W-0	U-0	R/W-0	R-0	R-0	R-0	R-0
VCFG1	VCFG0	—	FIFOEN	BFEMT	BFOVL	ADPNT1	ADPNT0
bit 7							bit 0

bit 7-6 **VCFG<1:0>**: Bit chọn nguồn điện áp tham chiếu A/D  $V_{REF+}$  và A/D  $V_{REF-}$

00 =  $V_{REF+} = AV_{DD}$ ,  $V_{REF-} = AV_{SS}$  (AN2 và AN3 là I/O)

01 =  $V_{REF+} = \text{External } V_{REF+}$ ,  $V_{REF-} = AV_{SS}$  (AN2 là I/O)

- 10 =  $V_{REF+} = AV_{DD}$ ,  $V_{REF-} = \text{External } V_{REF-}$  (AN3 là I/O)
- 11 =  $V_{REF+} = \text{External } V_{REF-}$ ,  $V_{REF-} = \text{External } V_{REF}$
- bit5 **Không sử dụng:** Đọc là ‘0’
- bit 4 **FIFOEN:** Bit kích hoạt FIFO Buffer  
1 = Kích hoạt FIFO.  
0 = Không kích hoạt FIFO .
- bit 3 **BFEMT:** Bit báo Buffer rỗng  
1 = FIFO là rỗng.  
0 = FIFO is không rỗng.
- bit 2 **BFOVFL:** Bit báo Buffer Overflow  
1 = Kết quả A/D bị ghi đè, trong khi dữ liệu chưa được đọc  
0 = Kết quả A/D không bị ghi đè.
- bit 1-0 **ADPNT<1:0>:** Con trỏ đọc bộ đệm Buffer  
00 = Buffer Address 0  
01 = Buffer Address 1  
10 = Buffer Address 2  
11 = Buffer Address 3

#### ➤ Thanh ghi ADCON2: Thanh ghi điều khiển 2

| R/W-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| ADFM  | ACQT3 | ACQT2 | ACQT1 | ACQT0 | ADCS2 | ADCS1 | ADCS0 |
| bit 7 |       |       |       |       |       |       | bit 0 |

- bit 7 **ADFM:** Bit định dạng kết quả A/D  
1 = Right justified  
0 = Left justified
- bit 6-3 **ACQT<3:0>:** Bit lựa chọn thời gian A/D Acquisition  
0000 = Không trễ (Bắt đầu chuyển đổi khi bit GO/DONE đặt mức 1)  
0001 =  $2 T_{AD}$   
0010 =  $4 T_{AD}$   
0011 =  $6 T_{AD}$   
0100 =  $8 T_{AD}$   
0101 =  $10 T_{AD}$   
0110 =  $12 T_{AD}$   
0111 =  $16 T_{AD}$

1000 = 20  $T_{AD}$

1001 = 24  $T_{AD}$

1010 = 28  $T_{AD}$

1011 = 32  $T_{AD}$

1100 = 36  $T_{AD}$

1101 = 40  $T_{AD}$

1110 = 48  $T_{AD}$

1111 = 64  $T_{AD}$

bit 2-0 **ADCS<2:0>**: Bit lựa chọn xung lock chuyển đổi A/D

000 =  $F_{osc}/2$

001 =  $F_{osc}/8$

010 =  $F_{osc}/32$

011 =  $F_{RC}/4$

100 =  $F_{osc}/4$

101 =  $F_{osc}/16$

110 =  $F_{osc}/64$

111 =  $F_{RC}$  (Internal A/D RC Oscillator)

#### ➤ Thanh ghi ADCON3: Thanh ghi điều khiển 3

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADRS1	ADRS0	—	SSRC4 <sup>(1)</sup>	SSRC3 <sup>(1)</sup>	SSRC2 <sup>(1)</sup>	SSRC1 <sup>(1)</sup>	SSRC0 <sup>(1)</sup>
bit 7							bit 0

bit 7-6 **ADRS<1:0>**: Lựa chọn tín hiệu báo ngắt khi có kết quả chuyển đổi, A/D ở chế độ Continuous Loop

Bit ADRS được bỏ qua trong chế độ Single-Shot.

00 = Ngắt xảy ra khi mỗi từ được viết lên bộ đệm buffer

01 = Ngắt xảy ra khi từ thứ 2 và 4 được viết lên bộ đệm buffer

10 = Ngắt được tạo ra khi từ thứ 4 được viết lên bộ đệm buffer

11 = Không thực hiện

bit 5 **Không sử dụng**: Đọc là ‘0’

bit 4-0 **SSRC<4:0>**: Chọn nguồn A/D Trigger

00000 = Vô hiệu hóa

xxxx1 = External interrupt RC3/INT0 starts A/D sequence

xxx1x = Timer5 starts A/D sequence

xx1xx = Input Capture 1 (IC1) starts A/D sequence  
 x1xxx = CCP2 compare match starts A/D sequence  
 1xxxx = Power Control PWM module rising edge starts A/D sequence

#### ➤ Thanh ghi ADCHS: Thanh ghi lựa chọn kênh A/D

| R/W-0  |
|--------|--------|--------|--------|--------|--------|--------|--------|
| GDSEL1 | GDSEL0 | GBSEL1 | GBSEL0 | GCSEL1 | GCSEL0 | GASEL1 | GASEL0 |
| bit 7  |        |        |        |        |        |        |        |

bit 7-6 **GDSEL<1:0>**: Bit chọn Group D

S/H-2 positive input.

00 = AN3

01 = AN7

1x = Reserved

bit 5-4 **GBSEL<1:0>**: Bit chọn Group B

S/H-2 positive input.

00 = AN1

01 = AN5

1x = Reserved

bit 3-2 **GCSEL<1:0>**: Bit chọn Group C

S/H-1 positive input.

00 = AN2

01 = AN6

1x = Reserved

bit 1-0 **GASEL<1:0>**: Bit chọn Group A

S/H-1 positive input.

00 = AN0

01 = AN4

10 = AN8<sub>(1)</sub>

11 = Reserved

#### ➤ Thanh ghi ANSEL0: Lựa chọn chân Analog 0

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
ANS7 <sup>(2)</sup>	ANS6 <sup>(2)</sup>	ANS5 <sup>(2)</sup>	ANS4	ANS3	ANS2	ANS1	ANS0
bit 7							

bit 7-0 **ANS<7:0>**: Bit lựa chọn đầu vào Analog cho các pins, AN<7:0>.

1 = Analog input

0 = Digital I/O

➤ Thanh ghi ANSEL1: Lựa chọn chân Analog 1

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-1
—	—	—	—	—	—	—	ANS8 <sup>(2)</sup>
bit 7							bit 0

bit 7-1 Không sử dụng: Đọc là ‘0’

bit 0 ANS8: Bit lựa chọn đầu vào Analog cho pin

1 = Analog input

0 = Digital I/O

➤ Các thành ghi liên quan đến module A/D

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	—	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
PIR2	OSCFIF	—	—	EEIF	—	LVDIF	—	CCP2IF
PIE2	OSCFIE	—	—	EEIE	—	LVDIE	—	CCP2IE
IPR2	OSCFIP	—	—	EEIP	—	LVDIP	—	CCP2IP
ADRESH	A/D Result Register High Byte							
ADRESL	A/D Result Register Low Byte							
ADCON0	—	—	ACONV	ACSCH	ACMOD1	ACMOD0	GO/DONE	ADON
ADCON1	VCFG1	VCFG0	—	FIFOEN	BFEMLT	BFOVFL	ADPNT1	ADPNT0
ADCON2	ADFM	ACQT3	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
ADCON3	ADRS1	ADRS0	—	SSRC4	SSRC3	SSRC2	SSRC1	SSRC0
ADCHS	GDSEL1	GDSEL0	GBSEL1	GBSEL0	GCSEL1	GCSEL0	GASEL1	GASEL0
ANSEL0	ANS7 <sup>(6)</sup>	ANS6 <sup>(6)</sup>	ANS5 <sup>(6)</sup>	ANS4	ANS3	ANS2	ANS1	ANS0
ANSEL1	—	—	—	—	—	—	—	ANS8 <sup>(5)</sup>
PORTA	RA7 <sup>(4)</sup>	RA6 <sup>(4)</sup>	RA5	RA4	RA3	RA2	RA1	RA0
TRISA	TRISA7 <sup>(4)</sup>	TRISA6 <sup>(4)</sup>	PORTA Data Direction Register					
PORTE <sup>(2)</sup>	—	—	—	—	RE3 <sup>(1,3)</sup>	RA2 <sup>(3)</sup>	RA1 <sup>(3)</sup>	RA0 <sup>(3)</sup>
TRISE <sup>(3)</sup>	—	—	—	—	—	PORTE Data Direction Register		
LATE <sup>(3)</sup>	—	—	—	—	—	LATE Data Output Register		

### 2.7.2. Điều khiển hoạt động chuyển đổi A/D

➤ Lựa chọn điện áp tham chiếu

Điện áp tham chiếu tương tự VREF+, VREF- được lựa chọn bằng phần mềm bởi hai bit VCFG<1:0> của thanh ghi ADCON1. Nó có thể lựa chọn là điện áp nguồn cấp AVDD, AVSS hoặc V<sub>REF+</sub> nối với chân AN3, V<sub>REF-</sub> nối với chân AN2. Thông thường

lựa chọn điện áp tham chiếu  $V_{REF-} = 0(V)$ ,  $V_{REF+}$  lớn hơn hoặc bằng với điện áp lớn nhất cần đo.

#### ➤ Lựa chọn nguồn xung clock và thời gian $T_{ACQ}$

$T_{AD}$  là thời gian chuyển đổi tín hiệu từ tương tự sang số của mỗi bit ADC. Nguồn xung tạo  $T_{AD}$  được lấy từ xung hệ thống Fosc hoặc từ bộ dao động nội. Sử dụng các bit ADCS2:ADCS0 của thanh ghi ADCON2 để lựa chọn  $T_{AD}$  từ  $2T_{OSC}$  ( $F_{OSC}/2$ ) đến  $64 T_{OSC}$  ( $F_{OSC}/64$ ) hoặc là  $F_{RC}$  (bộ phát xung RC nội).  $T_{ACQ}$  là thời gian chờ cần thiết để có thể thu nhận được 10 bit tín hiệu số. Sử dụng các bit ACQT3:ACQT0 của thanh ghi ADCON2 để lựa chọn  $T_{ACQ}$  từ  $2T_{AD}$  đến  $64T_{AD}$ . Có thể lựa chọn  $T_{ACQ}$  bằng 0  $T_{AD}$  với trường hợp sử dụng nguồn xung từ  $F_{RC}$  và được tạo trễ bằng một lệnh trước khi biến đổi A/D.

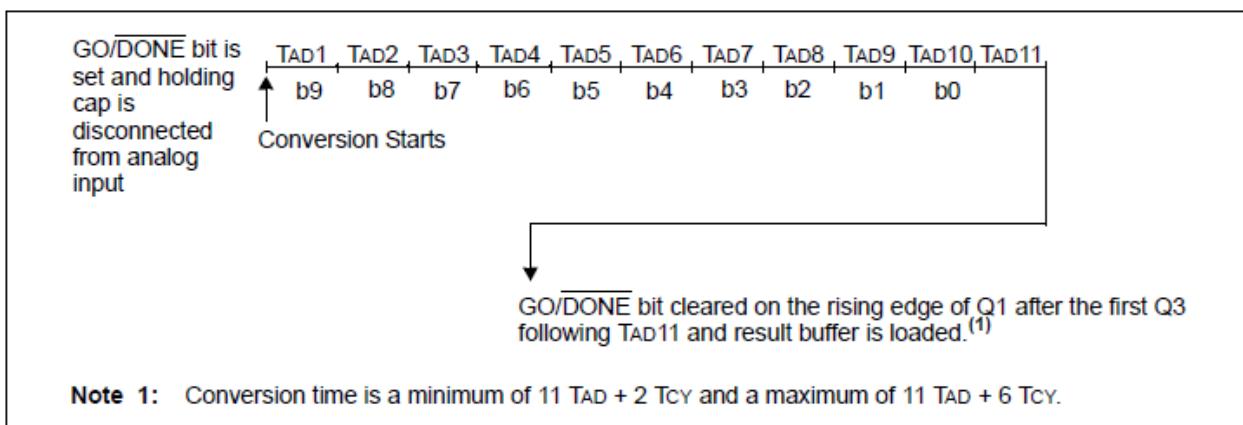
Bảng 2.11 thể hiện tần số tối đa của thiết bị có sử dụng tương ứng với việc chọn  $T_{AD}$ .

AD Clock Source ( $T_{AD}$ )		Maximum Device Frequency	
Operation	ADCS<2:0>	PIC18FXX31	PIC18LFXX31 <sup>(4)</sup>
2 Tosc	000	4.8 MHz	666 kHz
4 Tosc	100	9.6 MHz	1.33 MHz
8 Tosc	001	19.2 MHz	2.66 MHz
16 Tosc	101	38.4 MHz	5.33 MHz
32 Tosc	010	40.0 MHz	10.65 MHz
64 Tosc	110	40.0 MHz	21.33 MHz
$RC/4^{(3)}$	011	1.00 MHz <sup>(1)</sup>	1.00 MHz <sup>(2)</sup>
$RC^{(3)}$	111	4.0 MHz <sup>(2)</sup>	4.0 MHz <sup>(2)</sup>

Bảng 2.11. Lựa chọn  $T_{AD}$  phù hợp với tần số thiết bị

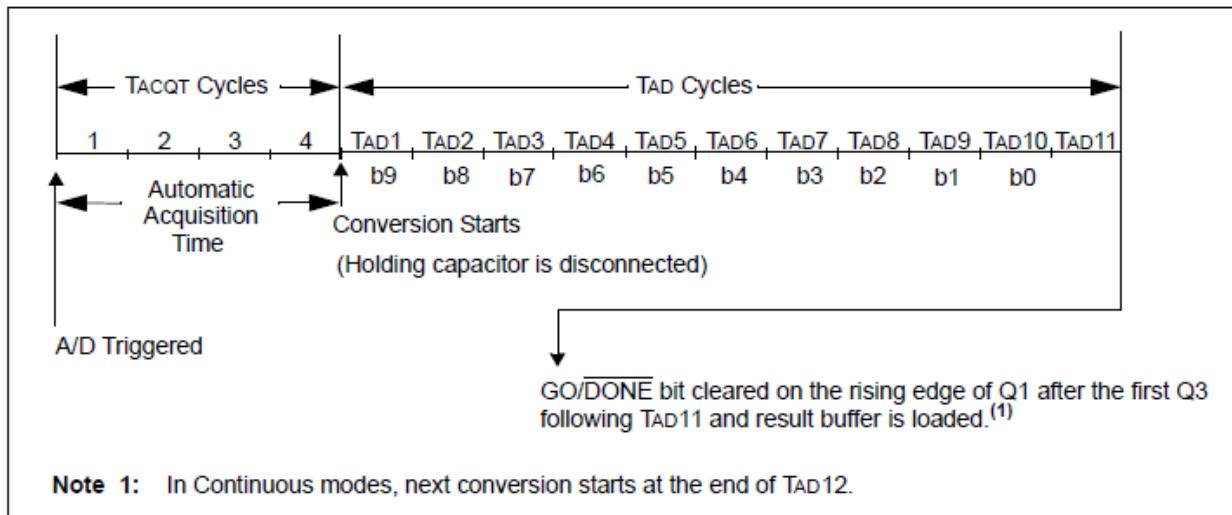
#### ➤ Quá trình chuyển đổi A/D

Hình 2.17 biểu diễn quá trình chuyển đổi tương tự - số của bộ biến đổi A/D theo thời gian với  $ACQT<2:0> = "000"$  (  $T_{ACQ}$  bằng  $0T_{AD}$ ), để bắt đầu chuyển đổi người lập trình cần thiết đặt bit  $GO/\overline{DONE}$  = “1”.



Hình 2.17. Chuyển đổi A/D với  $T_{ACQ} = 0 T_{AD}$

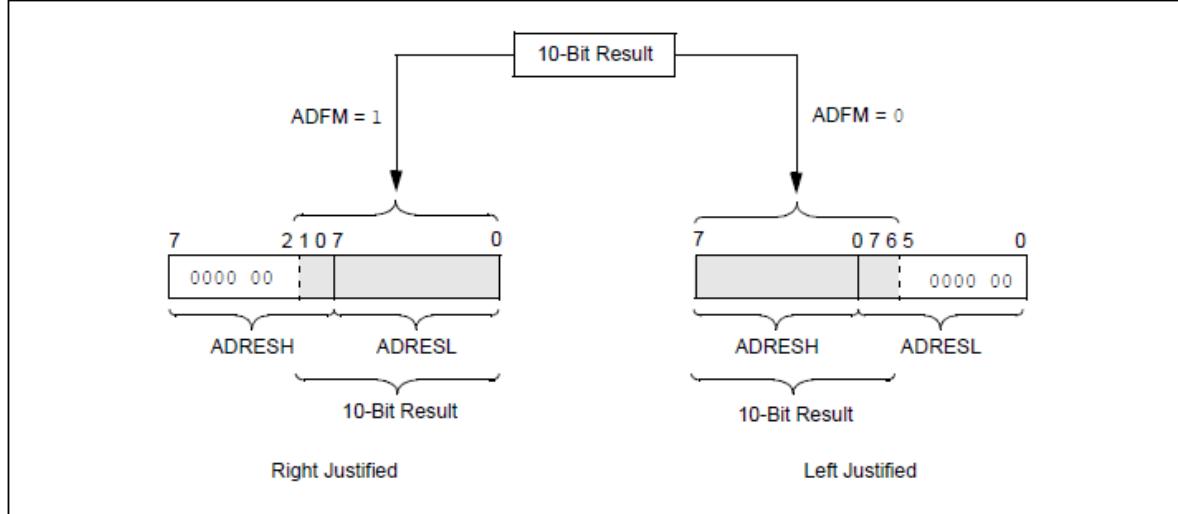
Hình 2.18 biểu diễn quá trình chuyển đổi tương tự - số của bộ biến đổi A/D theo thời gian với  $ACQT<3:0> = "0010"$  ( $T_{ACQ} = 4 T_{AD}$  ).



Hình 2.18. Chuyển đổi A/D với  $T_{ACQ} = 4 T_{AD}$

#### ➤ Lựa chọn cách ghi kết quả vào thanh ghi chúa

Bit ADFM (ADCON2<7>) được sử dụng để lựa chọn cách lưu giá trị số sau khi chuyển đổi vào cặp thanh ghi ADRESH và ADRESL. Cặp thanh ghi có độ dài 16 bit. Thông thường chúng ta chọn ADFM = “1” để có thể sử dụng ngay giá trị sau khi biến đổi, không cần phải dịch bit.



Hình 2.19. Lựa chọn cách ghi kết quả vào thanh ghi chúa

#### ➤ Các bước lập trình chuyển đổi A/D

Bước 1. Cấu hình module A/D:

- Cấu hình pin I/O số, analog, điện áp tham chiếu.
- Lựa chọn kênh đầu vào A/D.
- Chọn chế độ tự động chuyển đổi A/D (Single-Shot hay Continuous Loop).

d) Chọn xung clock chuyển đổi A/D  $T_{AD}$ .

e) Chọn A/D conversion trigger.

Bước 2. Cấu hình A/D interrupt (nếu có):

a) Set bit GIE.

b) Set bit PEIE bit.

c) Set bit ADIE bit.

d) Xóa bit ADIF bit.

e) Chọn bit cài đặt A/D trigger.

f) Chọn bit ưu tiên ngắn A/D.

Bước 3. Kích hoạt chuyển đổi ADC:

a) Set bit ADON trong thanh ghi ADCON0.

b) Đợi thời gian thiết lập, khoảng 5-10 us.

Bước 4. Bắt đầu quá trình lấy mẫu và chuyển đổi trình tự:

a) Lấy mẫu tối thiểu là  $2T_{AD}$  và bắt đầu chuyển đổi bởi bit GO/ $\overline{DONE}$ .

Bit GO/DONE được set bởi người dùng trong phần mềm hay bởi một module nếu cài đặt trigger.

b) Nếu  $T_{ACQ}$  được chỉ định một giá trị (nhiều  $T_{AD}$ ), cài đặt bit GO/ $\overline{DONE}$  để bắt đầu lấy mẫu với thời gian  $T_{ACQ}$ , sau đó bắt đầu chuyển đổi.

Bước 5. Chờ cho tới khi quá trình chuyển đổi A/D hoàn thành sử dụng một trong những lựa chọn sau:

a) Chờ tới khi bit GO/ $\overline{DONE}$  xóa về 0 nếu là chế độ Single-Shot.

b) Chờ cờ ngắn A/D (ADIF) được set.

c) Chờ tới khi bit BFEMT được xóa, báo ít nhất 1 lần chuyển đổi đã hoàn thành.

Bước 6. Đọc kết quả chuyển đổi A/D, xóa cờ ADIF, cấu hình lại trigger.

Lặp lại từ bước 1 nếu muốn quá trình biến đổi A/D diễn ra liên tục.

**Chú ý:** Thời gian chuyển đổi cho mỗi bit được định nghĩa là  $T_{AD}$ . Cần phải chờ thời gian tối thiểu là  $2 T_{AD}$  trước khi bắt đầu thực hiện việc chuyển đổi tiếp theo.

## 2.8. TRUYỀN THÔNG NỐI TIẾP TRONG VI ĐIỀU KHIỂN

### 2.8.1. Khái niệm về truyền thông nối tiếp

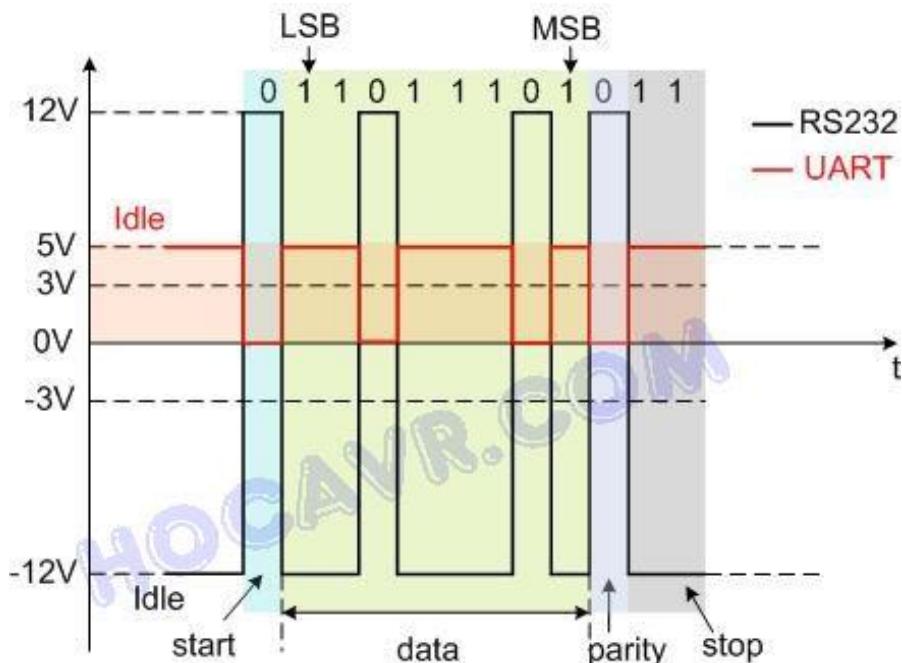
Thuật ngữ USART (*Universal Synchronous & Asynchronous serial Receiver and Transmitter*) nghĩa là bộ truyền nhận nối tiếp đồng bộ và không đồng bộ. Khi nói đến USART ta hiểu là đang nói đến "hardware". Các bộ USART được tích hợp trong

hầu như tất cả các chip vi điều khiển, và được coi như là 1 module ngoại vi - peripheral.

USART thường kết hợp với các *chuẩn giao tiếp* (communication standards) như **RS-232** để truyền nhận dữ liệu nối tiếp giữa chip vi điều khiển với 1 thiết bị khác, chẳng hạn như máy tính các nhân PC. Trên máy tính, cổng dùng để giao tiếp RS-232 là cổng DB9 (male), hay còn được gọi là **cổng COM**.

Khái niệm UART (*Universal Asynchronous serial Receiver and Transmitter*) tức là **bộ truyền nhận dữ liệu nối tiếp không đồng bộ**. Trong vi điều khiển UART là một chức năng ngoại vi dùng để giao tiếp với máy tính hoặc với vi điều khiển khác, hay một thiết bị nào đó có hỗ trợ UART.

USART hay UART cần phải kết hợp với một thiết bị chuyển đổi điện áp để tạo nên một chuẩn giao tiếp nào đó. Ví dụ chuẩn RS232 (COM Port) trên máy tính là sự kết hợp của chip UART và chip chuyển đổi mức điện áp. Trong khi đó tín hiệu theo chuẩn RS232 trên máy tính thường là -12V cho mức cao, và +12V cho mức thấp.



Hình 2.20. So sánh UART và RS232

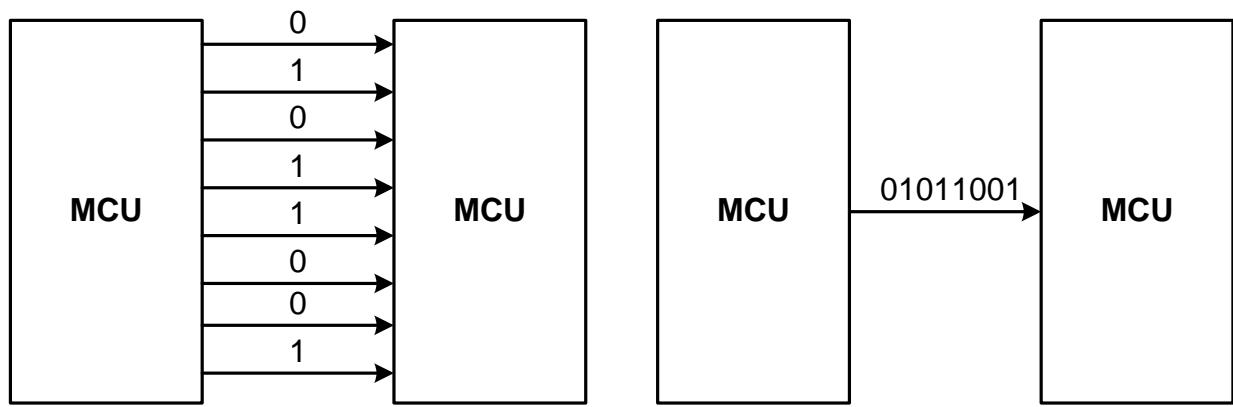
### ➤ Giao thức truyền thông

Máy tính truyền dữ liệu theo hai phương pháp: truyền dữ liệu song song và truyền dữ liệu nối tiếp.

**Truyền song song:** Sử dụng nhiều dây dẫn để truyền dữ liệu giữa các thiết bị có khoảng cách gần nhau (khoảng vài mét). Phương pháp này cho phép truyền dữ liệu với tốc độ cao nhờ sử dụng nhiều dây dẫn để truyền dữ liệu đồng thời nên tại một thời điểm có thể truyền được nhiều bit thông tin nhưng khoảng cách truyền thì có hạn chế.

**Truyền nối tiếp:** Sử dụng một dây dẫn để truyền dữ liệu (một dây phát đi và một dây thu về) giữa các thiết bị có khoảng cách xa nhau (khoảng vài trăm mét trở lên). Phương pháp này sẽ truyền dữ liệu với tốc độ chậm hơn (so với phương pháp truyền song song) vì chỉ sử dụng một dây dẫn để truyền dữ liệu nên tại mỗi thời điểm chỉ có thể truyền được một bit thông tin nhưng khoảng cách truyền thì không bị hạn chế như ở phương pháp song song.

Giả sử muốn truyền một dữ liệu 8 bit giữa hai vi điều khiển với nhau, chúng ta có thể nghĩ ngay đến cách đơn giản nhất là kết nối một PORT (8 bit) của mỗi vi điều khiển với nhau, mỗi đường trên PORT sẽ đảm nhiệm việc truyền/nhận một bit dữ liệu. Đây gọi là cách giao tiếp song song, cách này đơn giản và truyền nhận dữ liệu cũng không qua bất cứ một giải thuật truyền nhận nào, và tốc độ truyền nhận cũng rất nhanh. Tuy nhiên, nhược điểm của cách truyền này là số đường truyền nhiều, dữ liệu truyền càng lớn thì số đường truyền càng nhiều, do đó hệ thống truyền nhận song song thường rất cồng kềnh và kém hiệu quả. Ngược lại trong truyền thông nối tiếp, dữ liệu được truyền theo từng bit trên một đường truyền, chính vì vậy dữ liệu dù có lớn thì cũng chỉ cần một đường truyền duy nhất.



Hình 2.21. Truyền nhận 8 bit theo giao thức song song và nối tiếp

#### ➤ Chế độ truyền

Có 2 chế độ truyền: chế độ đồng bộ và chế độ không đồng bộ :

**Chế độ đồng bộ:** Trong quá trình truyền dữ liệu sử dụng 2 đường truyền: 1 đường truyền dữ liệu, một đường truyền xung nhịp. Khái niệm “đồng bộ” để chỉ sự “báo trước” trong quá trình truyền. Giao tiếp giữa máy tính và các bàn phím (trừ bàn phím kết nối theo chuẩn USB) là một ví dụ của cách truyền thông nối tiếp đồng bộ. Ưu điểm: truyền nhận dễ dàng với ít rủi ro trong quá trình truyền. Tuy nhiên cần đến 2 đường truyền cho một quá trình.

**Chế độ không đồng bộ:** Chỉ dùng một đường truyền cho một quá trình. “Khung dữ liệu” đã được chuẩn hóa bởi các thiết bị nên không cần đường xung nhịp báo trước dữ liệu đến. Thời gian truyền/nhận được thỏa thuận trước. Truyền thông nối tiếp không đồng bộ vì thế hiệu quả hơn truyền thông đồng bộ (không cần nhiều đường truyền).

**Baud rate (tốc độ baud):** để truyền nhận nối tiếp theo chế độ không đồng bộ thành công thì các thiết bị phải thống nhất với nhau về khoảng thời gian giành cho một bit truyền, hay nói cách khác là tốc độ truyền phải được cài đặt giống nhau, tốc độ này gọi là tốc độ baud. Ví dụ nếu tốc độ baud được đặt là 19200 bps thì thời gian dành cho 1 bit truyền là  $1/19200 \sim 52.083\text{us}$ .

### ➤ Frame (khung truyền)

Truyền thông nối tiếp không đồng bộ rất dễ mất hoặc sai lệch dữ liệu, quá trình truyền thông theo kiểu này phải tuân theo một số quy cách nhất định.

Khung truyền bao gồm các quy định về số bit trong mỗi lần truyền, các bit “báo” như bit Start và bit Stop, các bit kiểm tra như Parity, ngoài ra số lượng các bit trong một data cũng được quy định bởi khung truyền.

- **Start bit:** là bit đầu tiên được truyền trong một frame, bit này có nhiệm vụ báo cho thiết bị nhận biết rằng có một gói dữ liệu sắp được truyền tới. Start bit là bit bắt buộc phải có trong một khung truyền.

- **Data:** là số dữ liệu chúng ta cần phải truyền nhận, data có thể là gói 8 bit hay 9 bit tùy theo yêu cầu truyền nhận mà ta quy định. Trong truyền thông nối tiếp UART, bit có trọng số nhỏ nhất LSB sẽ được truyền trước, sau đó bit có trọng số lớn nhất sẽ được truyền sau cùng MSB.

- **Parity bit:** là bit dùng để kiểm tra dữ liệu truyền có đúng không, có hai loại parity là parity chẵn và parity lẻ.

- **Stop bit:** Là một hay các bit báo cho thiết bị biết rằng các bit đã được gửi xong. Sau khi nhận được stop bit, thiết bị nhận sẽ tiến hành kiểm tra khung truyền để đảm bảo tính chính xác của dữ liệu. Stop bit là bit bắt buộc phải có trong một khung truyền.

Ví dụ của một khung truyền, khung truyền này được bắt đầu bằng một start bit, tiếp theo là 8 bit data, sau đó là 1 bit parity dùng kiểm tra dữ liệu và cuối cùng là 1 bits stop.

Start	Data								Parity	Stop
0	1	1	0	1	1	1	0	1	0	1

UART hoạt động theo chuẩn NRZ (None-Return-to-Zero), nghĩa là các bit truyền đi bao gồm: 1 bit START (0) + 8 hay 9 bit dữ liệu (bit LSB sẽ được truyền đi trước) + một bit STOP. Khung truyền phổ biến nhất là : start bit + 8 bit data+1 stop bit



### 2.8.2. Module EUSART trên PIC18F4431

Module truyền nhận đồng bộ/không đồng bộ nối tiếp (Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART)) là một bộ vào/ra dữ liệu nối tiếp theo giao thức truyền tin nối tiếp USART. Module EUSART có thể được cấu hình thành một hệ thống truyền/nhận nối tiếp không đồng bộ song công và có thể truyền thông với các ngoại vi khác như các thiết bị đầu cuối, máy tính cá nhân (Personal Computers). EUSART cũng có thể được cấu hình thành bộ truyền/nhận nối tiếp bán song công (half-duplex), hệ thống đồng bộ có thể giao tiếp được với các thiết bị ngoại vi như là bộ biến đổi A/D hoặc D/A tích hợp, EEPROMs nối tiếp, v.v... EUSART trên PIC còn thực hiện một số tính năng nâng cao như: Tự động phát hiện tốc độ truyền và hiệu chuẩn, tự động đánh thức để tiếp nhận tín hiệu đồng bộ thoát (Sync Break) và truyền 12 ký tự thoát (Break character). Những cải tiến trên giúp cho vi điều khiển PIC18F4431 có thể phù hợp cho việc sử dụng trong hệ thống bus liên kết mạng cục bộ (LIN bus).

Module EUSART có thể hoạt động ở các chế độ:

- ✓ Không đồng bộ (song công) với:
  - Tự động đánh thức bởi ký tự tiếp nhận (Auto-wake-up on character reception).
  - Tự động hiệu chỉnh tốc độ baud.
  - Truyền 12 bit ký tự thoát.
- ✓ Đồng bộ chủ (bán song công), cho phép lựa chọn sự phân cực xung clock.
- ✓ Đồng bộ tớ (bán song công), cho phép lựa chọn sự phân cực xung clock.

Các chân của Module EUSART là các chân đa chức năng trên PORTC. Để lựa chọn cấu hình các chân RC6/TX/CK và RC7/RX/DT là chân của Module EUSART cần phải thiết lập các bit sau:

- Bit SPEN (RCSTA<7>) phải được thiết lập (=1).
- Bit TRISC<7> phải được thiết lập (=1).
- Bit TRISC<6> phải được thiết lập (=1).

Hoạt động của Module EUSART được điều khiển thông qua ba thanh ghi:

- Thanh ghi trạng thái truyền và điều khiển (TXSTA).
- Thanh ghi trạng thái nhận và điều khiển (RCSTA).
- Thanh ghi điều khiển tốc độ baud (BAUDCON).

➤ **Các thanh ghi liên quan**

*Thanh ghi trạng thái truyền và điều khiển (TXSTA):*

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN <sup>(1)</sup>	SYNC	SENDDB	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC**: bit lựa chọn nguồn xung

Chế độ không đồng bộ:

Không hỗ trợ.

Chế độ đồng bộ:

1 = Chế độ chủ (xung được tạo trên chip từ BRG)

0 = Chế độ tớ (nguồn xung từ bên ngoài)

bit 6 **TX9**: Bit cho phép truyền 9 bit

1 = Lựa chọn chế độ truyền 9 bit

0 = Lựa chọn chế độ truyền 8 bit

bit 5 **TXEN**: Bit cho phép truyền

1 = Cho phép truyền

0 = Không cho phép truyền

bit 4 **SYNC**: Bit lựa chọn chế độ EUSART

1 = Chế độ đồng bộ

0 = Chế độ không đồng bộ

bit 3 **SENDDB**: Bit gửi ký tự kết thúc (Break Character)

Chế độ không đồng bộ:

1 = Gửi đồng bộ kết thúc (được xóa bằng phần cứng lúc hoàn thành)

0 = Hoàn thành truyền đồng bộ kết thúc

Chế độ đồng bộ:

Không hỗ trợ.

bit 2 **BRGH**: Bit lựa chọn baud tốc độ cao

Chế độ không đồng bộ:

1 = Tốc độ cao

0 = Tốc độ thấp

Chế độ đồng bộ:

Không được sử dụng.

bit 1 **TRMT:** Bit báo trạng thái thanh ghi dịch truyền dữ liệu

1 = TSR rỗng

0 = TSR đầy

bit 0 **TX9D:** Bit truyền dữ liệu thứ 9

Có thể sử dụng chứa địa chỉ/dữ liệu hoặc bit chẵn lẻ.

*Thanh ghi điều khiển và trạng thái nhận (RCSTA)*

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:** Bit cho phép PORT nối tiếp

1 = Cho phép PORT nối tiếp (các chân RX/DT và TX/CK sẽ được cấu hình là các chân của PORT nối tiếp)

0 = Không cho phép

bit 6 **RX9:** Bit cho phép PORT nối tiếp nhận 9 bit

1 = Lựa chọn nhận 9 bit

0 = Lựa chọn nhận 8 bit

bit 5 **SREN:** Bit cho phép nhận đơn

Chế độ không đồng bộ:

Không sử dụng.

Chế độ đồng bộ – Chủ:

1 = Cho phép nhận đơn

0 = Không cho phép nhận

Bit này sẽ được xóa sau khi quá trình nhận hoàn thành.

Chế độ đồng bộ – Tớ:

Không sử dụng.

bit 4 **CREN:** Bit cho phép nhận liên tục

Chế độ không đồng bộ:

1 = Cho phép nhận liên tục

0 = Không cho phép nhận liên tục

Chế độ đồng bộ:

	1 = Cho phép nhận liên tục, CREN sẽ bị xóa khi SREN(bit cho phép nhận đơn) được thiết lập
	0 = Không cho phép nhận liên tục
bit 3	<b>ADDEN:</b> Bit cho phép phát hiện địa chỉ <u>Chế độ đồng bộ 9-Bit (RX9 = 1):</u> 1 = Cho phép phát hiện địa chỉ, cho phép ngắn và tải dữ liệu từ bộ đệm nhận khi RSR<8> được thiết lập (=1). 0 = Không cho phép phát hiện địa chỉ, tất cả các byte được nhận và bit thứ 9 có thể được sử dụng như là bit kiểm tra chẵn lẻ. <u>Chế độ không đồng bộ 8-Bit (RX9 = 0):</u> Không được sử dụng.
bit 2	<b>FERR:</b> Bit báo lỗi khung truyền/nhận 1 = Khung bị lỗi (có thể được xóa khi đọc thanh ghi RCREG và nhận byte hợp lệ kế tiếp). 0 = Không xảy ra lỗi khung
bit 1	<b>OERR:</b> Bit lỗi do tràn 1 = Lỗi tràn (có thể được xóa bằng khi xóa bit CREN) 0 = Không xảy ra lỗi tràn
bit 0	<b>RX9D:</b> Bit nhận dữ liệu thứ 9 Có thể chứa bit địa chỉ/dữ liệu hoặc bit chẵn lẻ và được tính toán và xử lý theo chương trình của người sử dụng.

#### *Thanh ghi điều khiển tốc độ baud (BAUDCON)*

U-0	R-1	U-0	R/W-1	R/W-0	U-0	R/W-0	R/W-0
—	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
bit 7							bit 0

bit 7	<b>Không sử dụng:</b> Đọc là “0”
bit 6	<b>RCIDL:</b> Bit trạng thái nghỉ (Idle) của hoạt động nhận 1 = Hoạt động nhận ở trạng thái nghỉ (Idle) 0 = Hoạt động nhận ở trạng thái hoạt động (Active)
bit 5	<b>Không sử dụng:</b> Đọc là “0”
bit 4	<b>SCKP:</b> Bit lựa chọn phân cực xung clock (Synchronous Clock Polarity) <u>Chế độ không đồng bộ:</u> Không sử dụng <u>Chế độ đồng bộ:</u>

1 = Trạng thái nghỉ (Idle) của hoạt động phát xung clock (CK) được thiết lập ở mức cao.

0 = Trạng thái nghỉ (Idle) của hoạt động phát xung clock (CK) được thiết lập ở mức thấp.

bit 3 **BRG16:** Bit cho phép thanh ghi tốc độ baud 16-Bit

1 = Bộ phát tốc độ baud 16-bit, gồm hai thanh ghi SPBRGH và SPBRG

0 = Bộ phát tốc độ baud 8-bit, chỉ sử dụng thanh ghi SPBRG, bỏ qua thanh ghi SPBRGH.

bit 2 **Không được sử dụng:** Đọc sẽ được ‘0’

bit 1 **WUE:** Bit cho phép đánh thức (Wake-up)

Chế độ không đồng bộ:

1 = EUSART tiếp tục lấy mẫu trên chân RX – ngắt sẽ phát sinh ở sườn âm; bit này sẽ được xóa bằng phần cứng sau khi có sườn dương.

0 = Chân RX không được giám sát hoặc phát hiện sườn

Chế độ đồng bộ:

Không sử dụng ở chế độ này.

bit 0 **ABDEN:** Bit cho phép phát hiện tốc độ baud tự động

Chế độ không đồng bộ:

1 = Cho phép đo tốc độ baud ở ký tự tiếp theo. Bit này được xóa bằng phần

cứng lúc hoàn thành.

0 = Không cho phép hoạt động đo tốc độ baud hoặc hoàn thành.

Chế độ đồng bộ:

Không sử dụng ở chế độ này.

## ➤ **Tốc độ baud**

Tốc độ baud hay còn được gọi là tốc độ truyền thông, là số bit được truyền đi trên một giây (bps – bit per second). Bộ tạo tốc độ baud BRG (Baud Rate Generator) có thể hoạt động ở chế độ 8 bit hoặc 16 bit, hỗ trợ cả chế độ đồng bộ và không đồng bộ của EUSART. Ở chế độ mặc định, BGR hoạt động ở chế độ 8 bit. Chế độ BGR 16 bit được lựa chọn khi bit BAUDCON<3> được thiết lập.

Hai thanh ghi SPBRGH:SPBRG được sử dụng để điều khiển chu kỳ xung tốc độ baud. Trong chế độ không đồng bộ, cả hai bit BRGH (TXSTA<2>) và BRG16(BAUDCON<3>) đều được sử dụng để điều khiển tốc độ baud. Trong chế độ đồng bộ, bit BRGH không được sử dụng.

- Lựa chọn chế độ và công thức tính tốc độ baud:

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-Bit/Asynchronous	Fosc/[64 (n + 1)]
0	0	1	8-Bit/Asynchronous	Fosc/[16 (n + 1)]
0	1	0	16-Bit/Asynchronous	
0	1	1	16-Bit/Asynchronous	Fosc/[4 (n + 1)]
1	0	x	8-Bit/Synchronous	
1	1	x	16-Bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGH:SPBRG register pair

- Tính sai số tốc độ baud

Truyền thông nối tiếp USART thường sử dụng các tốc độ baud chẵn như: 300, 600, 1200, 2400, 4800, 9600, 19200 v.v. Trên thực tế vi điều khiển PIC18F4520 thường sử dụng các loại thạch anh 40MHz, 20MHz, 16MHz, 10Mhz, 8MHz, 4MHz v.v. Vì vậy, chúng ta rất khó có thể chọn được giá trị nạp vào thanh ghi điều chỉnh tốc độ baud SPBRG để tạo được tốc độ baud mong muốn.

Ví dụ: Với tần số thạch anh FOSC = 16MHz, tốc độ truyền thông mong muốn là 9600, chế độ không đồng bộ, 8 bit BRG:

$$\text{Tốc độ mong muốn} = \text{FOSC}/(64([\text{SPBRGH}:\text{SPBRG}]+1))$$

Giá trị cần nạp vào cặp thanh ghi SPBRGH:SPBRG:

$$\begin{aligned} X &= ((\text{FOSC}/\text{tốc độ mong muốn})/64) - 1 \\ &= ((16000000/9600)/64) - 1 = 25.042 \end{aligned}$$

Giá trị nạp vào cặp thanh ghi SPBRGH:SPBRG phải là số nguyên nên làm tròn là 25.

$$\text{Tốc độ tính toán thực tế} = 16000000/(64 (25 + 1)) = 9615$$

$$\text{Sai số} = (9615 - 9600)/9600 = 0.16\%$$

- Các thanh ghi liên quan đến bộ điều chỉnh tốc độ baud (BRG):

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
BAUDCON	—	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
SPBRGH	EUSART Baud Rate Generator Register High Byte							
SPBRG	EUSART Baud Rate Generator Register Low Byte							

### ➤ Các chế độ không đồng bộ

Chế độ không đồng bộ được lựa chọn khi bit SYNC (TXSTA<4>) bị xóa (=0). Trong chế độ này, bộ EUSART sử dụng tiêu chuẩn Non-Return-to-Zero (NRZ). Tiêu chuẩn NRZ biểu diễn bit 1 bằng mức điện áp cao, còn bit 0 bằng mức điện áp thấp và

không sử dụng mức điện áp trung tính. Định dạng khung truyền NRZ trong chế độ này bao gồm một bit Start (bit khởi đầu khung truyền), kế tiếp là 8 hoặc 9 bit dữ liệu và kết thúc khung truyền là 1 bit Stop (bit kết thúc khung truyền). Định dạng dữ liệu phổ biến nhất là 8 bit.

Bộ EUSART truyền/nhận bit dữ liệu có trọng số thấp (LSB) trước. Hoạt động truyền và nhận của EUSART là độc lập nhau nhưng sử dụng chung định dạng khung truyền và tốc độ baud. Kiểm tra chẵn lẻ không được hỗ trợ bằng phần cứng nhưng có thể được thực hiện bằng phần mềm và lưu trữ trong bit thứ 9.

Khi hoạt động ở chế độ không đồng bộ, EUSART thực hiện một số tính năng quan trọng sau:

- Tạo tốc độ baud (BRG).
- Mạch lấy mẫu.
- Truyền không đồng bộ.
- Nhận không đồng bộ.
- Tự động đánh thức(Auto-Wake-up) bằng ký tự đồng bộ khung (Sync Break Character).
- Truyền 12 bit ký tự ngắt khung (12-bit Break Character).

#### ✓ **Hoạt động truyền của EUSART ở chế độ không đồng bộ**

Thanh ghi TSR (Transmit (Serial) Shift Register) được sử dụng để dịch lần lượt các bit dữ liệu nối tiếp từ bit trọng số thấp nhấp LSb đến bit có trọng số cao MSB ra chân TX. Thanh ghi TSR không cho phép đọc/ghi bằng phần mềm. Thanh ghi TXREG được sử dụng để đếm dữ liệu cho thanh ghi TSR. Dữ liệu cần truyền được nạp vào thanh ghi TXREG, sau đó dữ liệu sẽ được nạp tự động từ TXREG sang TSR. Thanh ghi TSR chưa được nạp dữ liệu khi bit Dừng (Stop) trước đó chưa được truyền đi.

Ngay sau khi bit Dừng được truyền đi thì dữ liệu sẽ được nạp vào TRS (nếu có dữ liệu trong TXREG). Ngay sau khi dữ liệu được nạp từ TXREG sang TSR (trong một chu kỳ máy), thanh ghi TXREG sẽ rỗng và cờ ngắt truyền TXIF (PIR1<4>) sẽ được thiết lập (=1).

Bit TXIF được sử dụng để biết trạng thái của thanh ghi TXREG, còn bit TRMT (TXSTA<1>) được sử dụng để biết trạng thái của thanh ghi TSR. Bit TRMT chỉ được phép đọc, nó thiết lập khi TSR rỗng. Hoạt động ngắt không được gắn liền với bit này, nó chỉ sử dụng để báo trạng thái rỗng của thanh ghi TSR.

*Các bước để truyền dữ liệu ở chế độ không đồng bộ:*

Bước 1. Khởi tạo giá trị cho cùp thanh ghi SPBRGH:SPBRG, thiết lập hoặc xóa bit BRGH và BRG16 để đạt được tốc độ truyền mong muốn (theo bảng chế độ và công thức tính tốc độ baud).

Bước 2. Xóa bit SYNC (TXSTA<4>) để cho phép chế độ không đồng bộ và thiết lập bit SPEN (RCSTA<7>) để cho phép PORT nối tiếp.

Bước 3. Nếu muốn sử dụng ngắt thì cần phải thiết lập bit TXIE.

Bước 4. Để thiết lập khung truyền là 9-bit cần thiết lập bit TX9 (TXSTA<6>). Khi đó bit-9 sẽ có thể được sử dụng để chứa địa chỉ/dữ liệu hoặc bit kiểm tra chẵn lẻ.

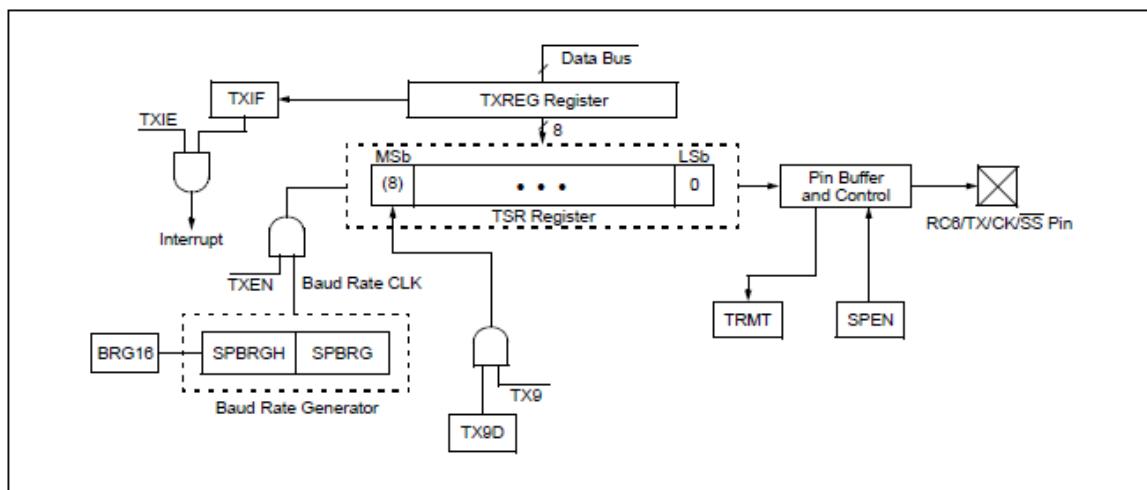
Bước 5. Cho phép truyền dữ liệu bằng bit TXEN.

Bước 6. Nếu khung truyền 9 bit được lựa chọn, bit thứ 9 cần được nạp vào TX9D.

Bước 7. Nạp dữ liệu cần truyền vào thanh ghi TXREG (quá trình truyền dữ liệu sẽ được bắt đầu).

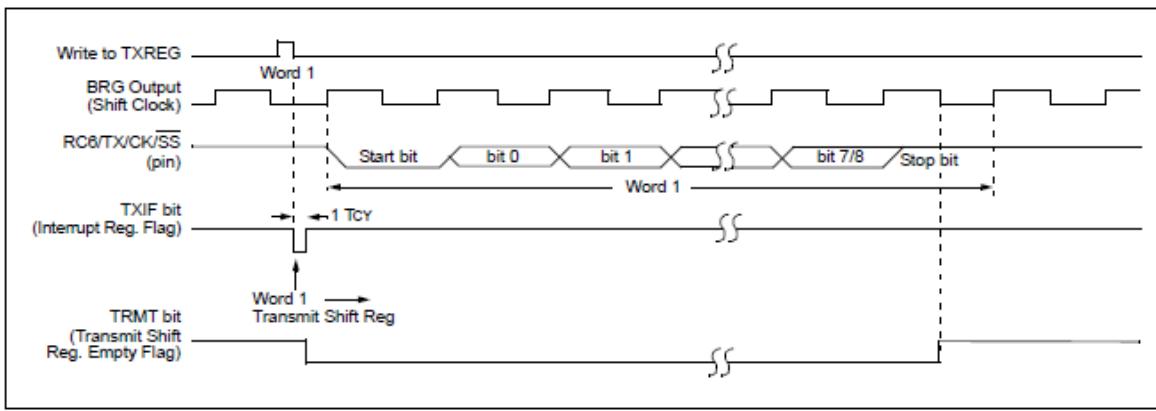
Bước 8. Nếu sử dụng ngắt, cần chắc chắn rằng bit GIE và PEIE của thanh ghi INTCON (INTCON<7:6>) đã được thiết lập.

Sơ đồ khái niệm hoạt động truyền của EUSART ở chế độ không đồng bộ:



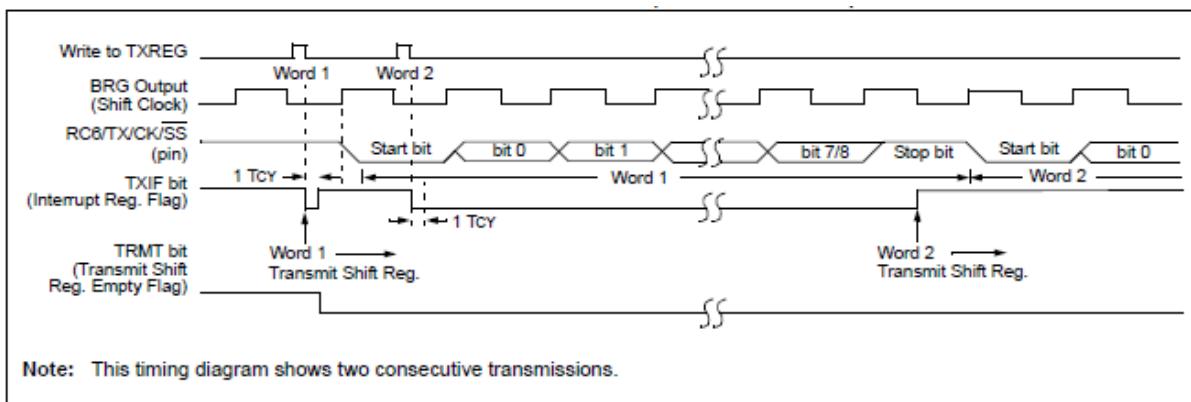
Hình 2.22. Sơ đồ khái niệm hoạt động truyền dữ liệu của EUSART ở chế độ không đồng bộ

Giản đồ thời gian của hoạt động truyền không đồng bộ:



Hình 2.23. Sơ đồ khái hoạt động truyền không đồng bộ

Giản đồ thời gian của hoạt động truyền không đồng bộ (truyền liên tiếp các byte):



Hình 2.24. Sơ đồ khái hoạt động truyền không đồng bộ (truyền liên tiếp các byte)

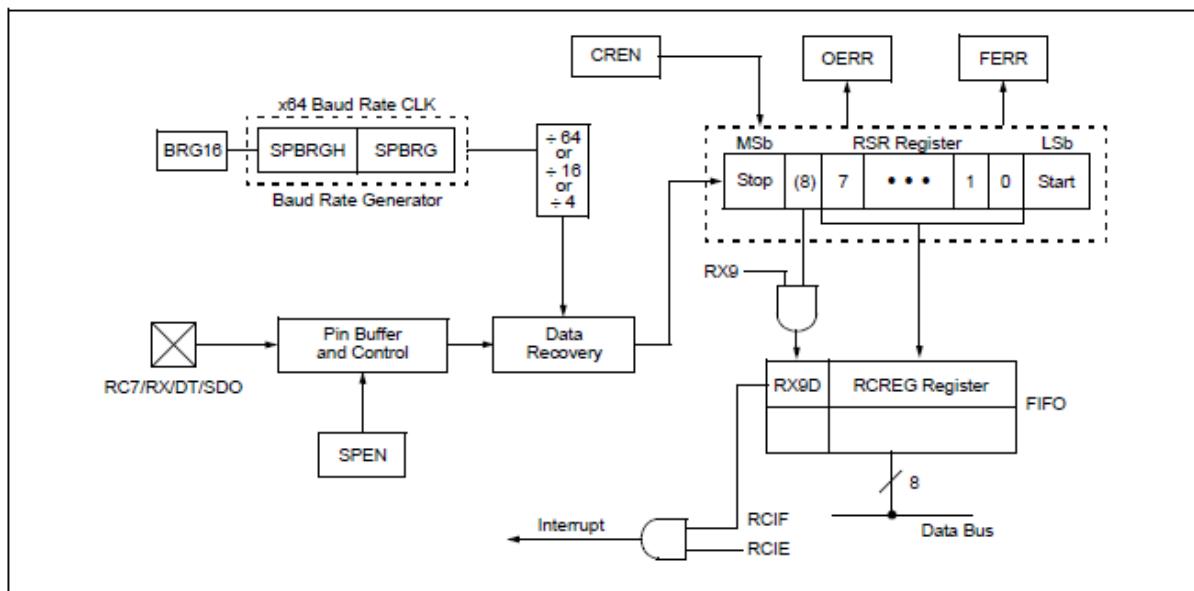
Các thanh ghi liên quan đến hoạt động truyền không đồng bộ:

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	—	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
TXREG	EUSART Transmit Register							
TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
BAUDCON	—	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
SPBRGH	EUSART Baud Rate Generator Register High Byte							
SPBRG	EUSART Baud Rate Generator Register Low Byte							

Legend: — = unimplemented, read as '0'. Shaded cells are not used for asynchronous transmission.

#### - Hoạt động nhận của EUSART ở chế độ không đồng bộ

Sơ đồ khái của hoạt động nhận ở chế độ bát động bộ được thể hiện ở hình 2.14 dưới. Dữ liệu được nhận về qua chân RX và khôi phục hồi dữ liệu. Chế độ này thường được sử trong hệ thống truyền thông RS-232.



Hình 2.25. Sơ đồ khái niệm dữ liệu của EUSART ở chế độ không đồng bộ

Các bước để truyền dữ liệu ở chế độ không đồng bộ:

Bước 1. Khởi tạo giá trị cho cặp thanh ghi SPBRGH:SPBRG, thiết lập hoặc xóa bit BRGH và BRG16 để đạt được tốc độ truyền mong muốn (theo bảng chế độ và công thức tính tốc độ baud).

Bước 2. Xóa bit SYNC (TXSTA<4>) để cho phép chế độ không đồng bộ và thiết lập bit SPEN (RCSTA<7>) để cho phép PORT nối tiếp.

Bước 3. Nếu sử dụng ngắt thì cần phải thiết lập bit RCIE.

Bước 4. Để cho phép nhận bit thứ 9 cần phải thiết lập bit RX9 (RCSTA<6>).

Bước 5. Thiết lập bit CREN để cho phép hoạt động nhận.

Bước 6. Bit cờ ngắt RCIF sẽ được thiết lập khi hoạt động nhận hoàn thành, ngắt sẽ xảy ra khi bit cho phép ngắt RCIE đã được thiết lập trước đó.

Bước 7. Đọc bit RX9D (RCSTA<0>) để có được bit thứ 9 (nếu khung truyền 9 bit được cho phép), căn cứ vào bit thứ 9 để phát hiện lỗi khung truyền.

Bước 8. Đọc 8 bit dữ liệu nhận về trong thanh ghi RCREG.

Bước 9. Nếu phát hiện dữ liệu nhận bị lỗi, xóa lỗi bằng cách xóa bit cho phép nhận CREN.

Bước 10. Nếu sử dụng ngắt, cần chắc chắn rằng bit GIE và PEIE của thanh ghi INTCON (INTCON<7:6>) đã được thiết lập.

*Hoạt động dò tìm địa chỉ ở chế độ nhận 9 bit:* Chế độ này thường được sử dụng trong hệ thống truyền thông RS-485.

Các bước để nhận không đồng bộ và cho phép tự động phát hiện địa chỉ:

Bước 1. Khởi tạo giá trị cho cặp thanh ghi SPBRGH:SPBRG, thiết lập hoặc xóa bit BRGH và BRG16 để đạt được tốc độ truyền mong muốn (theo bảng chế độ và công thức tính tốc độ baud).

Bước 2. Xóa bit SYNC (TXSTA<4>) để cho phép chế độ không đồng bộ và thiết lập bit SPEN (RCSTA<7>) để cho phép PORT nối tiếp.

Bước 3. Nếu sử dụng ngắt thì cần phải thiết lập bit RCIE và lựa chọn mức ưu tiên ngắt bằng bit RCIP.

Bước 4. Thiết lập bit RX9 để cho phép hoạt động nhận 9 bit.

Bước 5. Thiết lập bit ADDEN để cho phép phát hiện địa chỉ.

Bước 6. Cho phép hoạt động nhận bằng bit CREN.

Bước 7. Bit cờ ngắt RCIF sẽ được thiết lập khi hoạt động nhận hoàn thành. Ngắt sẽ xảy ra khi bit RCIE và GIE đã được thiết lập trước đó.

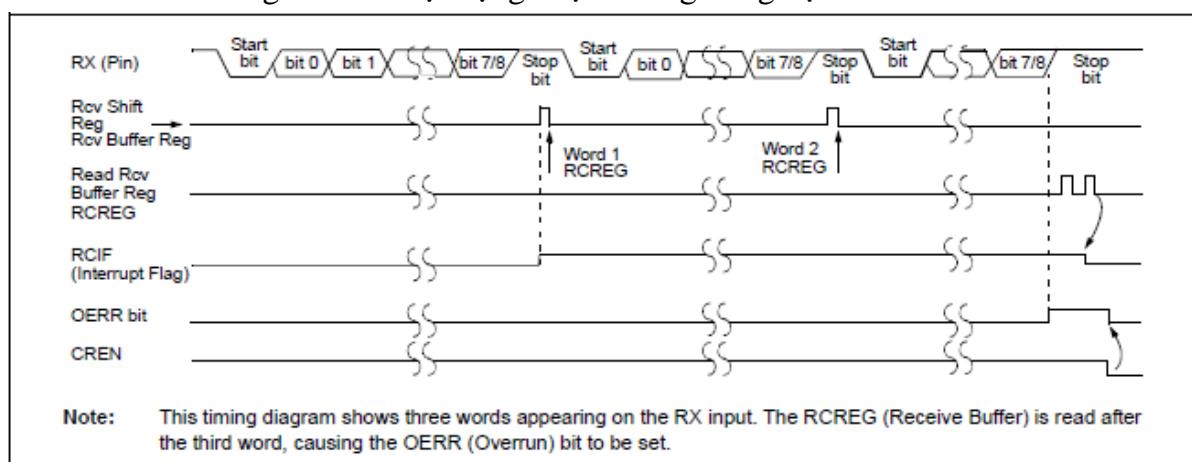
Bước 8. Đọc thanh ghi RCSTA để xác định lỗi có xảy ra ở hoạt động nhận, cũng như đọc bit dữ liệu thứ 9 (nếu có).

Bước 9. Đọc thanh ghi RCREG để phát hiện thiết bị có địa chỉ phù hợp.

Bước 10. Nếu phát hiện lỗi xảy ra thì cần phải xóa bit CREN.

Bước 11. Nếu thiết bị phù hợp địa chỉ được phát hiện, cần xóa bit ADDEN để nhận dữ liệu vào bộ đếm nhận và ngắt CPU.

Giản đồ thời gian của hoạt động nhận không đồng bộ:



Hình 2.26. Giản đồ thời gian của hoạt động nhận không đồng bộ

Các thanh ghi liên quan đến chế độ nhận không đồng bộ:

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	—	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
RCREG	EUSART Receive Register							
TXSTA	CSRC	TX9	TXEN	SYNC	SENDDB	BRGH	TRMT	TX9D
BAUDCON	—	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
SPBRGH	EUSART Baud Rate Generator Register High Byte							
SPBRG	EUSART Baud Rate Generator Register Low Byte							

Legend: — = unimplemented, read as '0'. Shaded cells are not used for asynchronous reception.

## ➤ Ngắt USART

Ngắt từ bộ USART xảy ra khi kết thúc quá trình truyền một ký tự (ngắt truyền) hoặc kết thúc quá trình nhận một ký tự (ngắt nhận).

Để đặt ngắt từ bộ USART, ngoài việc đặt/xóa các bit toàn cục (xem chương 4) cần tác động đến các bit sau:

- Bit cho phép ngắt nhận RCIE (thanh ghi PIE1); bit ưu tiên ngắt nhận RCIP (thanh ghi IPR1).
- Bit cho phép ngắt truyền TXIE (thanh ghi PIE1); bit ưu tiên ngắt nhận TXIP (thanh ghi IPR1).

Khi xảy ra ngắt, cờ ngắt tương ứng sẽ được thiết lập: (Với ngắt truyền là bit TXIF (thanh ghi PIR1); Với ngắt nhận là bit RCIF (thanh ghi PIR1)). Các cờ ngắt này cần xóa bằng phần mềm.

## CÂU HỎI HƯỚNG DẪN ÔN TẬP, THẢO LUẬN

Câu 1: Trình bày khái niệm ngắt trong vi điều khiển? Cho ví dụ.

Câu 2: Trình bày các chế độ hoạt động của bộ định thời TIMER0. Cho ví dụ.

Câu 3: Trình bày các chế độ hoạt động của bộ định thời TIMER1. Cho ví dụ.

Câu 4: Trình bày các chế độ hoạt động của bộ định thời TIMER2. Cho ví dụ.

Câu 5: Trình bày nguyên lý và các bước thiết lập khối CCP ở chế độ PWM. Cho ví dụ.

Câu 6: Trình bày nguyên lý làm việc và các bước cấu hình bộ biến đổi ADC. Cho ví dụ.

Câu 7: Trình bày nguyên lý làm việc và các bước cấu hình bộ truyền thông nối tiếp trong vi điều khiển PIC. Cho ví dụ.

## BÀI TẬP ÚNG DỤNG

**Câu 1:** Viết chương trình vi điều khiển PIC thực hiện chức năng tạo xung 10KHz trên chân RB0 sử dụng TIMER0 ở chế độ 8 bit, tần số thạch anh sử dụng là 20MHz.

**Câu 2:** Viết chương trình vi điều khiển PIC thực hiện chức năng tạo xung 1KHz trên chân RB0 sử dụng TIMER0 ở chế độ 16 bit, tần số thạch anh sử dụng là 20MHz.

**Câu 3:** Viết chương trình vi điều khiển PIC thực hiện chức năng đếm sự kiện trên RC3/T0CKI (tích cực mức 0) sử dụng TIMER0 ở chế độ 8 bit, tần số thạch anh sử dụng là 20MHz.

**Câu 4:** Viết chương trình vi điều khiển PIC thực hiện đếm sự kiện trên RC3/T0CKI (tích cực mức 0) sử dụng TIMER0 ở chế độ 16 bit, tần số thạch anh sử dụng là 20MHz.

**Câu 5:** Viết chương trình vi điều khiển PIC thực hiện chức năng tạo xung 10KHz trên chân RB0 sử dụng ngắt TIMER0, tần số thạch anh sử dụng là 20MHz.

**Câu 6:** Viết chương trình vi điều khiển PIC thực hiện chức năng tạo xung 1KHz trên chân RB0 sử dụng ngắt TIMER0, tần số thạch anh sử dụng là 20MHz.

**Câu 7:** Viết chương trình vi điều khiển PIC thực hiện chức năng tạo xung PWM tần số 10KHz trên RC2/CCP1 với độ rộng xung 50% ?

**Câu 8:** Viết chương trình vi điều khiển PIC thực hiện chức năng tạo xung PWM tần số 10KHz trên RC2/CCP1 với độ rộng xung 70% ?

**Câu 9:** Viết chương trình vi điều khiển PIC thực hiện chức năng tạo xung PWM tần số 20KHz trên RC2/CCP1 với độ rộng xung 50% ?

**Câu 10:** Viết chương trình vi điều khiển PIC thực hiện chức năng đọc giá trị ADC trên AN0, AN1. Trong đó AN0 và AN1 sử dụng 2 biến trỏ (0-5V).

### CHƯƠNG 3

## LẬP TRÌNH CÁC ỨNG DỤNG VỚI VI ĐIỀU KHIỂN PIC 18F

### MỤC TIÊU CỦA CHƯƠNG

Nội dung chương 3 cung cấp cho sinh viên kiến thức về lập trình các ứng dụng với vi điều khiển PIC 18F sử dụng ngôn ngữ lập trình C: lập trình giao tiếp với nút ấn, cảm biến logic, Led 7 thanh, LCD 16x2, một số ứng dụng trong đo lường, bộ điều khiển PID.

#### 3.1. GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH C

C là một ngôn ngữ khá mạnh và rất nhiều người dùng .Nếu nói số lệnh cơ bản của C thì không nhiều. Nhưng đối với lập trình cho vxl , chúng ta chỉ cần biết số những lệnh cơ bản sau.

##### a) Cấu trúc cơ bản của 1 chương trình C viết cho vi điều khiển

//Các chỉ thị tiền định

```
#include <lcd.h> //Gọi thư viện có sẵn cách viết khác "*.h"
```

```
#define led1 PORTA.0 //dùng định nghĩa các biến
```

```
char bien1,bien2; //cac bien can dung
```

```
int a,b;
```

```
void chuongtrinhcon(unsigned int b) // chương trình con
```

```
{
```

```
...
```

```
}
```

```
int ham(void) // chương trình con đang ham
```

```
{
```

```
....
```

```
Return(a);
```

```
}
```

```
void main(void) //chương trình chính
```

```
{
```

```
int a; // khai báo biến đang số nguyên
```

```
chuongtrinhcon();
```

```
a = ham();
```

```
}
```

Chương trình con là nơi các viết các chương trình nhỏ, rất tiện cho các đoạn lệnh gắp lại nhiều lần . Chương trình con có thể có thể gọi ở trong chương trình chính bất kì đâu .Hàm là chương trình con trả về cho mình một giá trị.Cách sử dụng hàm và chương trình con các bạn nên tham khảo thêm quyển kĩ thuật lập trình C để hiểu rõ hơn .

### b) Các lệnh cơ bản của C

#### +Lệnh điều kiện if - else:

Lệnh điều kiện if – else được dùng để rẽ nhánh chương trình, phát biểu if – else có dạng như sau:

```
If (điều kiện)
{
    Các lệnh trong chương trình
}
else
{
    Các lệnh trong chương trình
}
```

Cặp từ khóa {} được dùng nếu các lệnh trong chương trình gồm nhiều lệnh. Trong trường hợp chỉ

có một lệnh thì không cần dùng cặp từ khoá {}.

#### + Vòng lặp WHILE:

Vòng lặp while được dùng để lặp chương trình. Cấu trúc của vòng lặp while như sau:

```
while (biểu thức điều kiện)
{
    Các lệnh trong chương trình
}
```

Hoạt động của vòng lặp while là sẽ thực hiện các lệnh trong cặp từ khoá {} khi mà biểu thức điều kiện là đúng.

Ví dụ:

```
while(get_rtcc()!=0)
{
    putc('n');
    getc();
```

...

}

+ **Vòng lặp do – while:**

Vòng lặp do – while được sử dụng tương tự như vòng lặp while tuy nhiên, vòng lặp while kiểm

tra điều kiện trước khi thực hiện các lệnh còn vòng lặp do – while sẽ kiểm tra điều kiện sau khi thực hiện các lệnh. Cấu trúc của vòng lặp do – while như sau:

```
do
{
    Các lệnh trong chương trình
}
while (biểu thức điều kiện);
```

Ví dụ:

```
do
{
    putc(getc());
    get_rtcc();
    ...
}
while(c!=0);
```

+ **Vòng lặp for:**

Vòng lặp for được dùng để lặp lại chương trình theo một biến đếm. Cấu trúc của vòng lặp for như sau:

```
For (biểu thức 1; biểu thức 2; biểu thức 3)
{
    Các lệnh trong chương trình
}
```

Trong đó biểu thức 1 là giá trị khởi đầu của biến đếm, biểu thức 2 là giá trị cuối của biến đếm, biểu thức 3 là biểu thức đếm.

Ví dụ:

```
For (I=1;I<=100; I++)
{
    b = I +100;
```

```
    printf("%u\r\n", I);
}
```

+ **Phát biểu SWITCH – CASE:**

Phát biểu switch – case được dùng để rẽ nhánh chương trình. Cấu trúc của phát biểu này như sau:

```
Switch (biểu thức điều kiện)
{
    case điều kiện 1:
        Các lệnh trong chương trình;
        Break;
    Case điều kiện 2:
        Các lệnh trong chương trình;
        Break;
    ...
    default:
        Các lệnh trong chương trình;
        Break;
}
```

Ví dụ:

```
Switch (cmd)
{
    case 0:
    {
        printf("cmd 0");
        break;
    }
}
```

+ **Phát biểu return:**

Phát biểu return được dùng để trả về trị của hàm. Phát biểu return như sau:

```
Return (giá trị);
```

+ **Phát biểu goto:**

Phát biểu goto được dùng để nhảy tới một nhãn trong chương trình. Phát biểu goto có dạng như sau:

goto nhän;

+ **Phát biểu break:**

Phát biểu break dùng để kết thúc một nhóm lệnh trong cặp từ khóa {}. Phát biểu break như sau:

break;

+ **Phát biểu continue:**

Phát biểu continue dùng để tiếp tục thực hiện một nhóm lệnh trong cặp từ khóa {}. Phát biểu continue như sau:

continue;

c) **Các phép toán trong C**

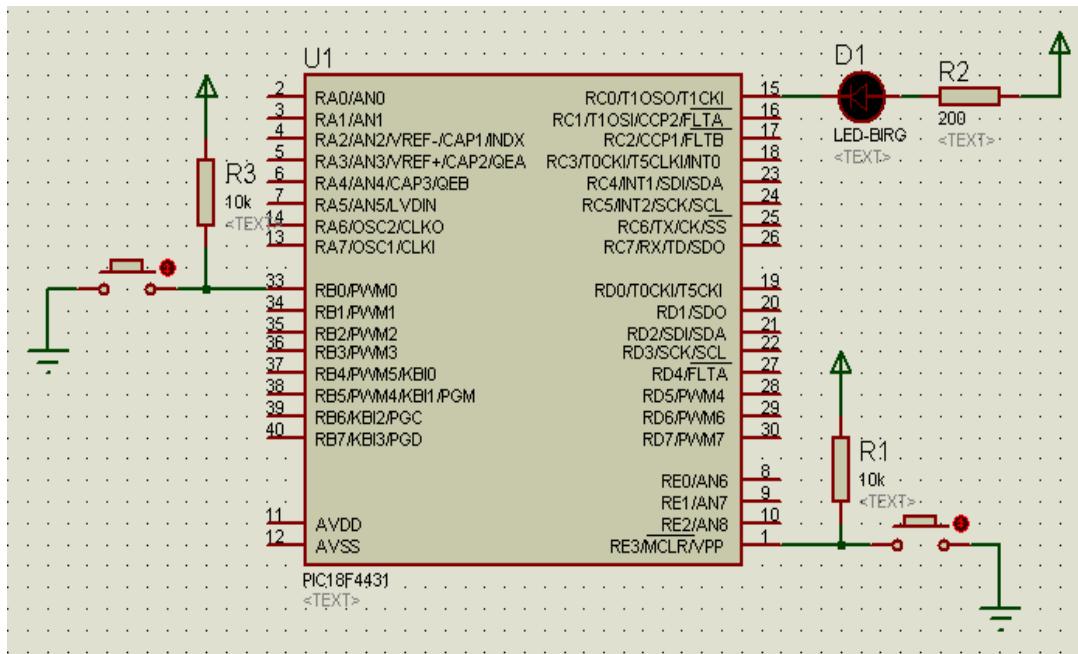
Trong PIC C compiler sử dụng các ký hiệu như bảng sau để đặc trưng cho các phép toán.

Ký hiệu	Phép toán
Các phép toán số học	
+	Thực hiện phép tính cộng
+=	Thực hiện phép cộng dồn. Ví dụ: x+=y tương tự như x = x + y
-	Thực hiện phép tính trừ
-=	Thực hiện phép trừ dồn. Ví dụ: x-=y tương tự như x = x - y
*	Thực hiện phép tính nhân
*=	Thực hiện phép nhân dồn. Ví dụ: x*=y tương tự như x = x * y
/	Thực hiện phép tính chia
/=	Thực hiện phép chia dồn. Ví dụ: x/=y tương tự như x = x / y
++	Thực hiện việc tăng một giá trị. Ví dụ: i++ tương tự như i = i + 1
--	Thực hiện việc giảm một giá trị. Ví dụ: i-- tương tự như i = i - 1
=	Phép gán. Ví dụ: x = 1
%	Lấy trị tuyệt đối
Các phép toán so sánh	
==	Phép so sánh bằng
!=	Phép so sánh khác
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
Các phép toán nhị phân	
&&	Phép toán AND
	Phép toán OR
>>	Dịch phải
<<	Dịch trái
!	Đảo bit
~	Lấy bù 1
&	AND từng bit
	OR từng bit
sizeof	Lấy kích thước của byte

### 3.2. LẬP TRÌNH VÀ GIAO TIẾP VỚI NÚT NHẤN, CẢM BIẾN LOGIC

Yêu cầu: Nhấn nút nhấn tại RD0, thì LED tại RC0 sáng. Nhả nút nhấn, LED tắt.

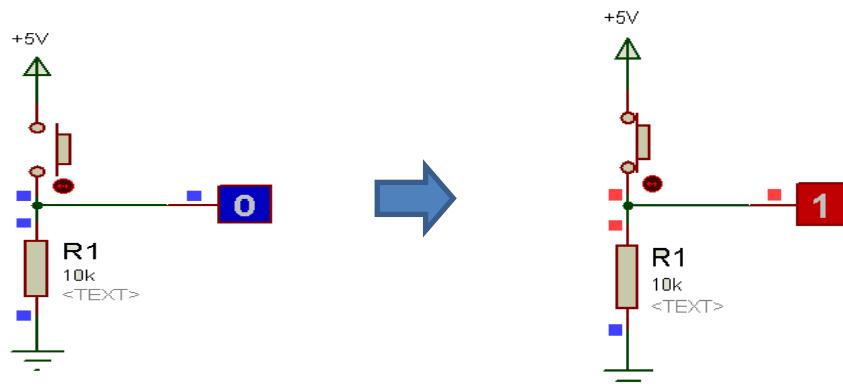
Sơ đồ ghép nối vi điều khiển PIC với nút bấm như hình 3.1.



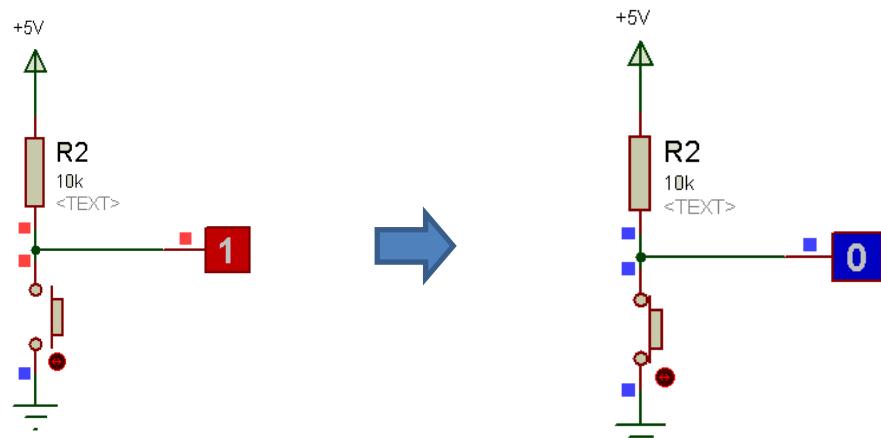
Hình 3.1. Ghép nối vi điều khiển PIC với nút bấm

➤ Các kiểu mắc nút nhấn với vi điều khiển.

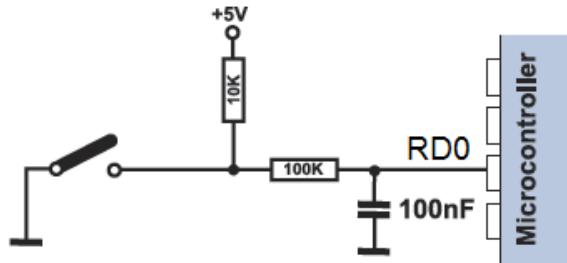
Kiểu 1: Tích cực mức 1



Kiểu 2: Tích cực mức 0

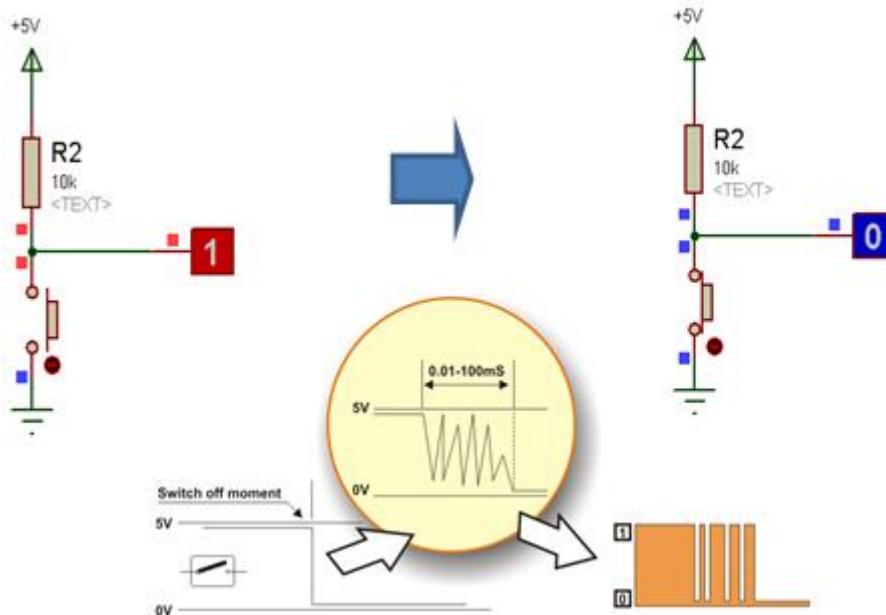


- **Kiểm tra nút nhấn** có được nhấn hay không bằng cách kiểm tra trạng thái trên chân của vi điều khiển:

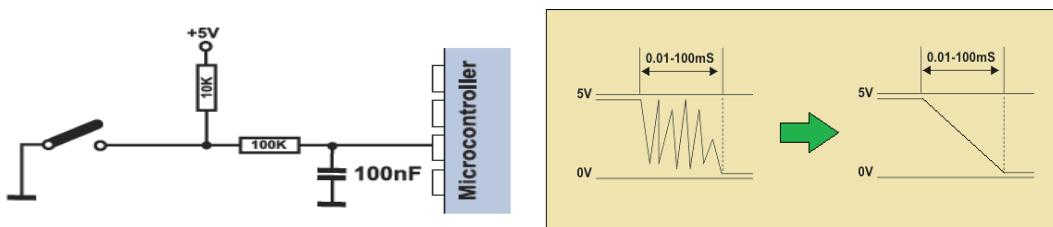


- RD0=1: phím chưa được nhấn
- RD0=0: phím được nhấn

- **Hiện tượng dội phím (rung phím):**

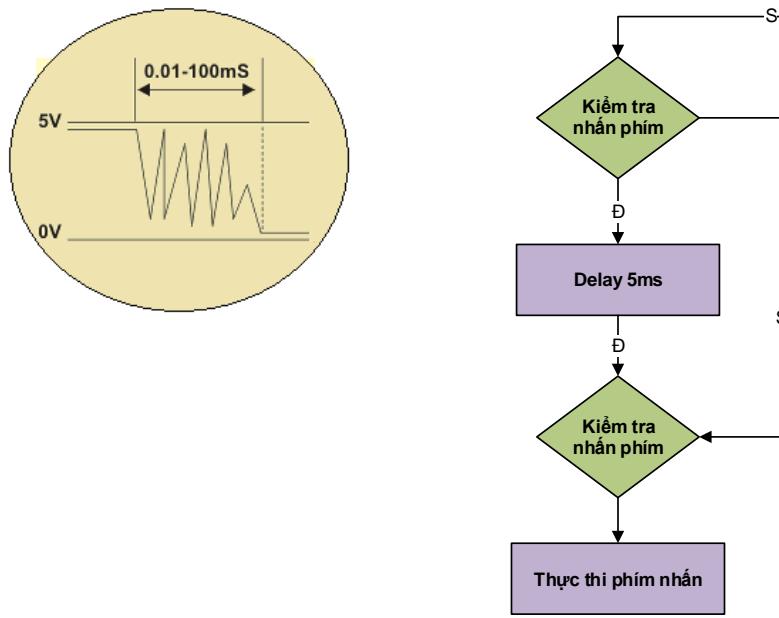


- ✓ Chống dội bằng phần cứng:



- ✓ Chống dội bằng phần mềm:

- + Hành động xảy ra khi nhấn phím:



### Chương trình điều khiển

```

#include <xc.h>
#include "newxc8_header.h"
#define _XTAL_FREQ 20000000
void main(void)
{
    TRISDbits.RD0=1;
    TRISBbits.RB0=0;
    LATBbits.LATB0=1;
    while(1)
    {
        if(PORTDbits.RD0==0)
        {
            __delay_ms(5);
            if(PORTDbits.RD0==0)
            {
                LATBbits.LATB0=0;
            }
        }
    }
}

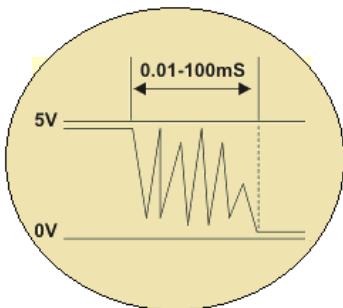
```

```

{
    LATBbits.LATB0=1;
}
}

+
+ Hành động xảy ra khi nhả phím

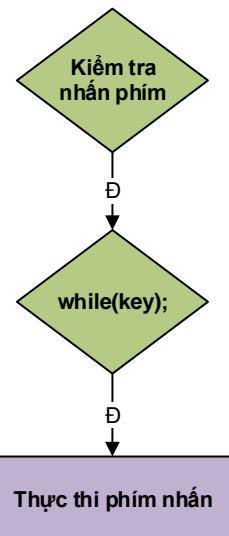
```



```

if(PORTDbits.RD0==0)
{
    //Nhảy tại chỗ nếu vẫn nhấn phím
    while(PORTDbits.RD0==0);
    PORTB=~PORTB;
}

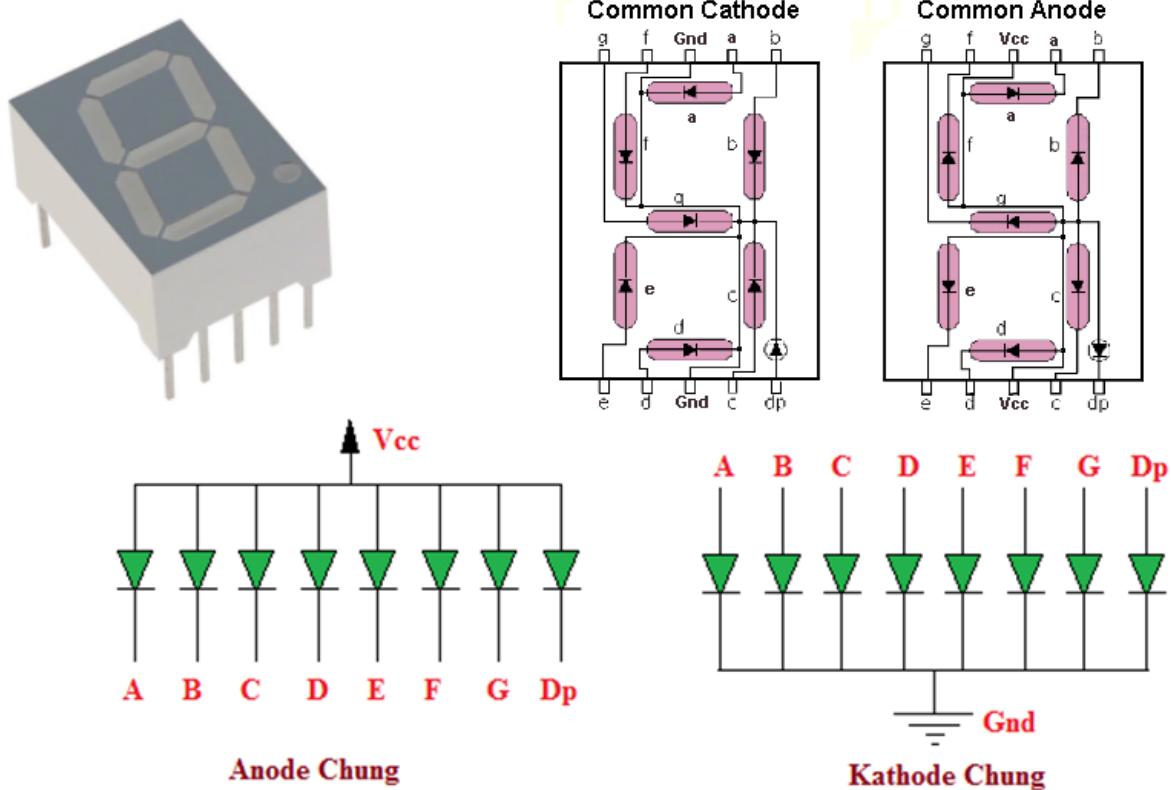
```



### 3.3. LẬP TRÌNH VÀ GIAO TIẾP VỚI LED 7 THANH VÀ LCD 16x2

#### 3.3.1. Lập trình và giao tiếp với LED 7 thanh

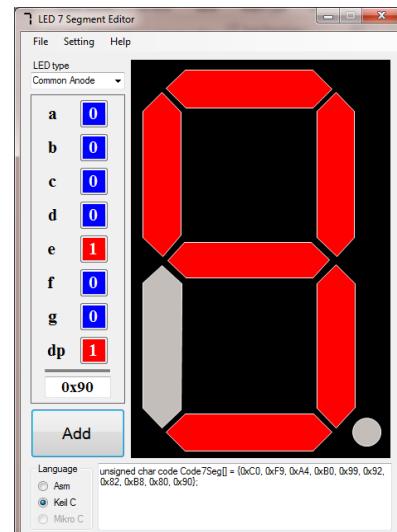
Cấu tạo Led 7 thanh minh họa như hình 3.2.



Hình 3.2. Cấu tạo Led 7 thanh

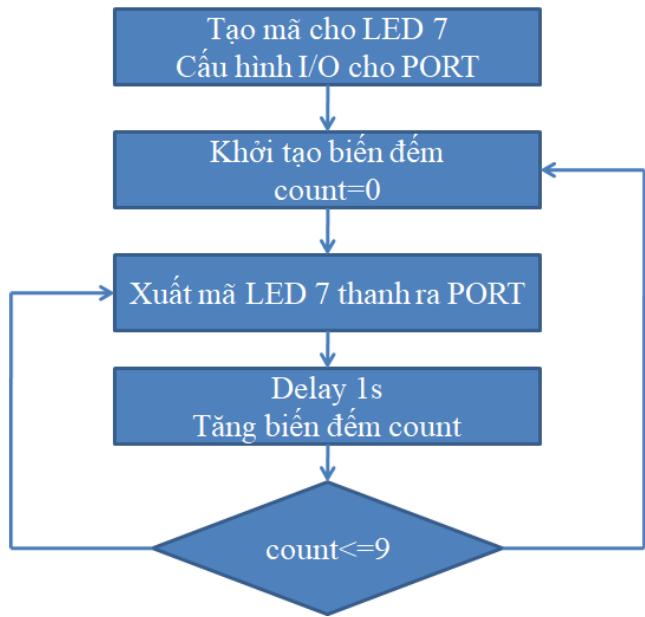
Mã Led 7 thanh Anot chung cho như bảng dưới, hoặc sử dụng phần mềm Sử dụng phần mềm LED 7 Segment Editor.

Số thập phân	Mã Led 7 thanh Anot chung								Mã Hexa
	db	g	f	e	d	c	b	a	
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	0	0	0	0	90

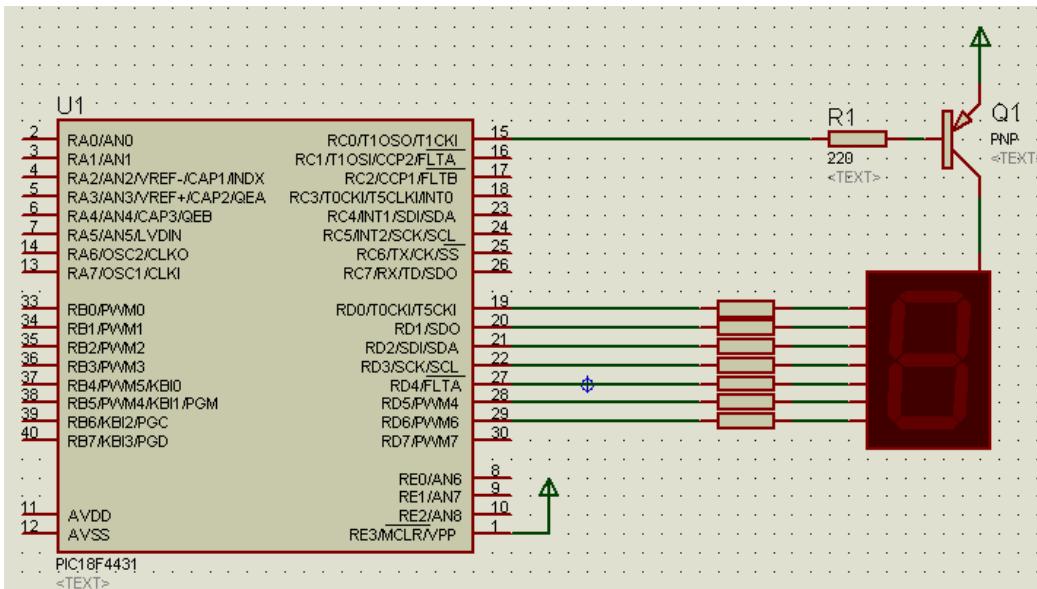


### ➤ Ghép nối một Led 7 thanh với Vi điều khiển

Ví dụ: Viết chương trình hiển thị từ số 0 đến 9 trên một LED 7 thanh



✓ **Sơ đồ kết nối mô phỏng**



✓ **Chương trình lập trình:**

```

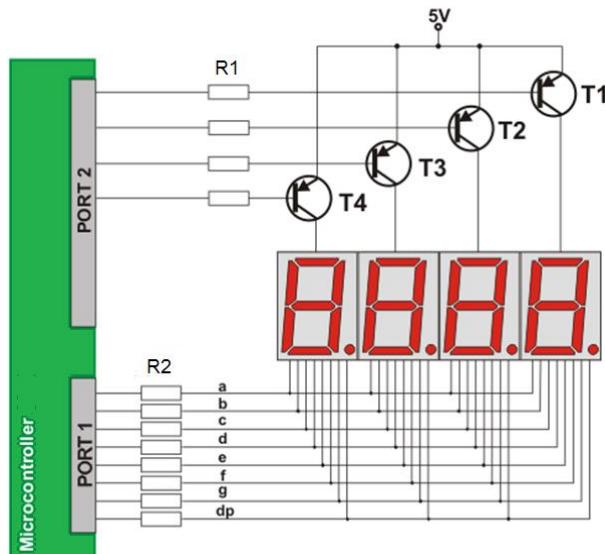
#include <xc.h>
#include "File_Config.h"
#define _XTAL_FREQ 20000000
//Khai bao ma LED 7 doan Anot chung
unsigned char maled[10] = {0xC0, 0xF9, 0xA4,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90};
void delay_ms(int i)
{
    for (int x = 0;x<=i;x++)
    {
        __delay_ms(1);
    }
}
✓
}

void main(void)
{
    TRISD=0x00; //ngo ra
    TRISCbits.RC0=0; //ngo ra
    LATD=0xFF; //Gia tri ban dau
    LATCbits.LATC0=0; //Cap nguon Led 7
    while(1)
    {
        for(char dem=0;dem<10;dem++)
        {
            LATD=maled[dem]; //ma LED7
            delay_ms(1000); //Tre 1s
        }
    }
}

```

### ➤ Ghép nối nhiều LED 7 thanh với vi điều khiển

Sơ đồ ghép nối nhiều LED 7 thanh với vi điều khiển minh họa như hình 3.3. Ngoài việc xuất dữ liệu LED, ta còn phải thực hiện quét LED 7 thanh. Vấn đề là giữ thời gian 1 LED sáng trong bao lâu.



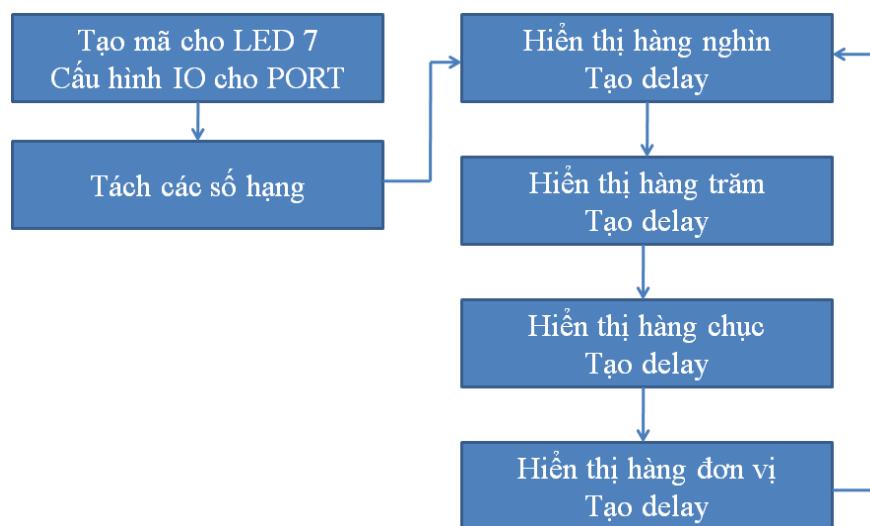
*Hình 3.3. Sơ đồ ghép nối nhiều LED 7 thanh với vi điều khiển*  
Để mắt người nhìn thấy ảnh, thì ảnh phải được hiển thị (quét) 24 lần/giây

$$\text{Thời gian delay} = \frac{1 \text{ giây}}{\text{Số LED} \times \text{số lần quét}}$$

Ví dụ: Mạch có 4 LED 7 thanh, số lần quét 30 lần/giây

$$\text{Thời gian delay} = \frac{1 \text{ giây}}{4 \times 30} = 0,08 \text{ giây} = 8 \text{ ms}$$

✓ **Ví dụ viết chương trình hiển thị một số có 4 chữ số trên 4 LED 7 số 1234**



Chương trình lập trình:

```

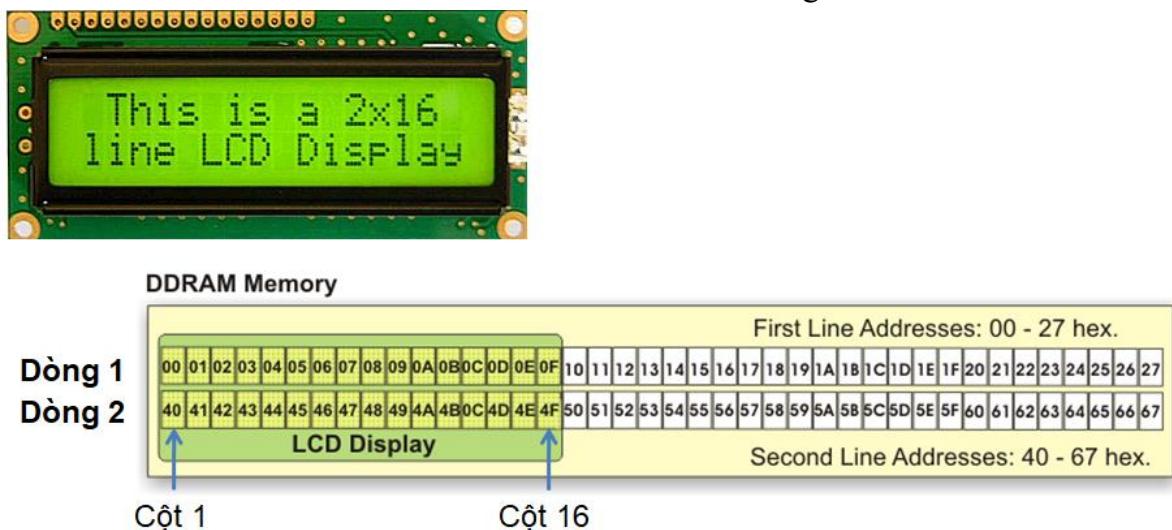
//Tach cac so hang cu
dvi=number%10; //Dvi =4
number=number/10; //number=123
chuc=number%10; //Chuc = 3
number=number/10; //number=12
tram=number%10; //tram = 2
number=number/10; //number=1
nghin=number;

while(1){
    //Hien thi so hang dvi
    LATCbits.LC3=0;
    LATD=Code7Seg[dvi];
    delay_ms(8);
    LATC=0xff; //Chong lem

    //Hien thi so hang chuc
    LATCbits.LC2=0;
    LATD=Code7Seg[chuc];
    delay_ms(8);
    LATC=0xFF; //Chong lem
}
  
```

### 3.3.2. Lập trình và giao tiếp với LCD 16x2

Cấu tạo của LCD 16x2 minh họa như hình 3.4. Trong đó:



Hình 3.4. Hình ảnh và cấu tạo LCD 16x2

- ✓ DDRAM: chứa các ký tự hiển thị lên màn hình
  - ✓ CGRAM: chứa các ký tự tự do người dùng tự định nghĩa. Có 64 ô nhớ tương ứng 8 ký tự.
  - ✓ CGROM: chứa các ký tự định nghĩa sẵn

## Sơ đồ chân của LCD 16x2:



<b>1. GND, 2. VCC</b>	Nguồn	<b>6. E</b>	Enable
<b>3. VEE</b>	Tương phản	<b>7 -&gt; 14</b> <b>D0 -&gt; D7</b>	Chân dữ liệu
<b>4. RS</b>	Register Select	<b>15. K, 16. A</b>	LED nền
<b>5. R/W</b>	Đọc/Ghi		

## Thư viện LCD 16x2:

- **Hàm void LCD\_Init()**
    - Chức năng: Khởi tạo các chế độ hoạt động của LCD

- **Hàm LCD\_Write\_String(String);**
  - Chức năng: Gửi một ký tự lên LCD
  - Ví dụ: LCD\_Write\_String("HELLO")
- **Hàm LCD\_Gotoxy(x,y);** Trong đó x=1...2 , y=0...15
  - Chức năng: Di chuyển tới vị trí hàng/cột.

Ví dụ: LCD\_Gotoxy(1,0); // con trỏ hàng 1, cột 0.

- **Hàm void LCD\_Clear();**
  - Chức năng: Xóa màn hình LCD
- **Hàm chuyển đổi sang chuỗi ký tự**
- **Sprintf**

Ví dụ: unsigned int number=12;

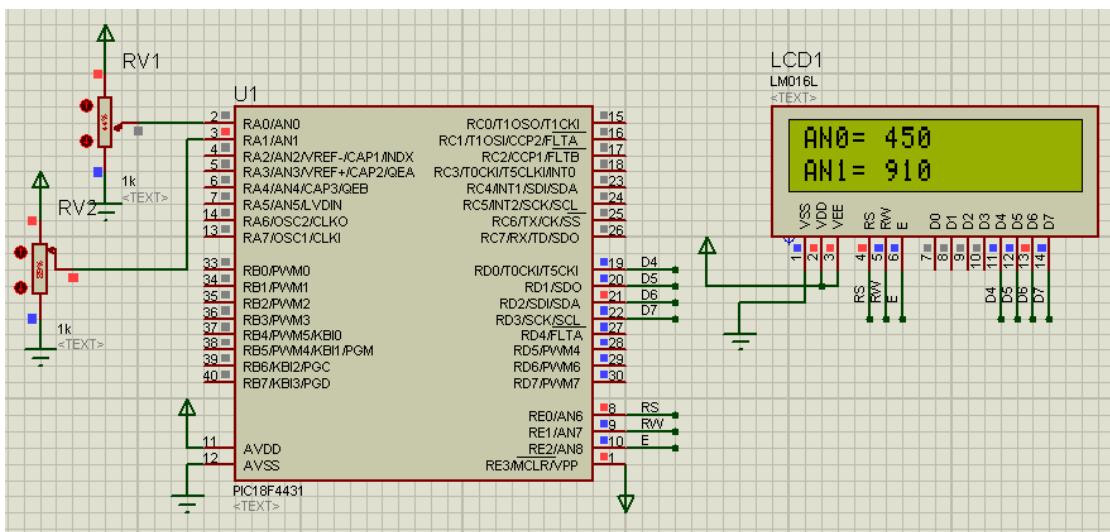
char text[3];

sprintf(text,"%2d",number); // Chuyển đổi số nguyên sang ký tự

- Khai báo thư viện : stdio.h khi sử dụng hàm

### 3.4. LẬP TRÌNH VÀ GIAO TIẾP CÁC TÍN HIỆU TƯƠNG TỰ

Ta xét ví dụ minh họa sau: Viết chương trình biến đổi điện áp tương tự trên chân AN0, AN1 và hiển thị trên LCD.



#### Cài đặt các thanh ghi điều khiển

- ADCON0=0x33=0b00110011
- ADCON1=0x10=0b00010000
- ADCON2=0x88=0b10001000

#### Viết chương trình

```
#include <xc.h>
```

```

#include "18F_LCD.h" //Khai bao thu vien LCD
#include <stdio.h>

#pragma config OSC = HS      // Dao dong tan so cao
#pragma config WDTEN = OFF   // Tat che do Watchdog Timer
#pragma config MCLRE = ON    // Chon reset ngoai boi Pin MCLR
#define _XTAL_FREQ 20000000
char text[4];

//Cau hinh ADC (AN0 va AN1)
void Config_ADC( )
{
    ANSEL0=0x03; //AN0 va AN1
    ANSEL1=0x00; //AN8 off
    ADCON0=0x33; //GO/DONE: on e Conversor A/D: on
    ADCON1=0x10; //FIFO is enabled
    ADCON2=0x88; //Right justified A;
    ADCHS=0x00; //AN0, AN1, AN2 va AN3
}

unsigned int Temp1;
void Read_ADC_AN0()
{
    Temp1=ADRESH;
    Temp1=Temp1<<8;
    Temp1+=ADRESL;
}

unsigned int Temp2;
void Read_ADC_AN1()
{
    Temp2=ADRESH;
    Temp2=Temp2<<8;
    Temp2+=ADRESL;
}

```

```

void main(void)
{
    TRISAbits.RA0=1;
    TRISAbits.RA1=1;
    LCD_Init();
    Config_ADC();
    LCD_Clear();
    LCD_Gotoxy(1,0);
    LCD_Write_String("AN0=");
    LCD_Gotoxy(2,0);
    LCD_Write_String("AN1=");
    while(1)
    {
        Read_ADC_AN0();
        Read_ADC_AN1();
        sprintf(text,"%4d",Temp1);
        LCD_Gotoxy(1,4);
        LCD_Write_String(text);
        sprintf(text,"%4d",Temp2);
        LCD_Gotoxy(2,4);
        LCD_Write_String(text);
    }
}

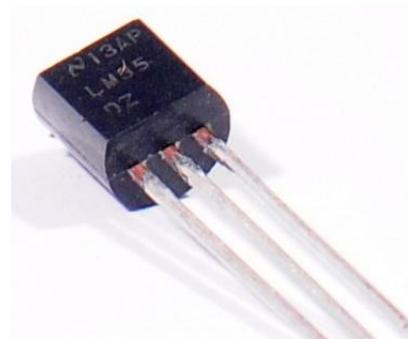
```

### **3.4.1. Lập trình giao tiếp với cảm biến nhiệt độ**

Hiện nay trên thị trường có rất nhiều loại cảm biến nhiệt độ, tùy theo nhu cầu sử dụng và môi trường làm việc mà ta lựa chọn loại cảm biến nhiệt độ phù hợp. Nếu phân loại theo tín hiệu trả về của cảm biến, ta chia thành những loại sau: áp, điện trở. (xem hình 3.6). Với các loại cảm biến có tín hiệu trả về dạng áp, điện trở ta sử dụng các kênh ADC của vi điều khiển để xử lý và tính toán. Ngoài ra có một số cảm biến nhiệt độ trả về dưới dạng dữ liệu truyền thông I2C, SPI những kiểu truyền thông này hầu hết các vi điều khiển đều hỗ trợ.



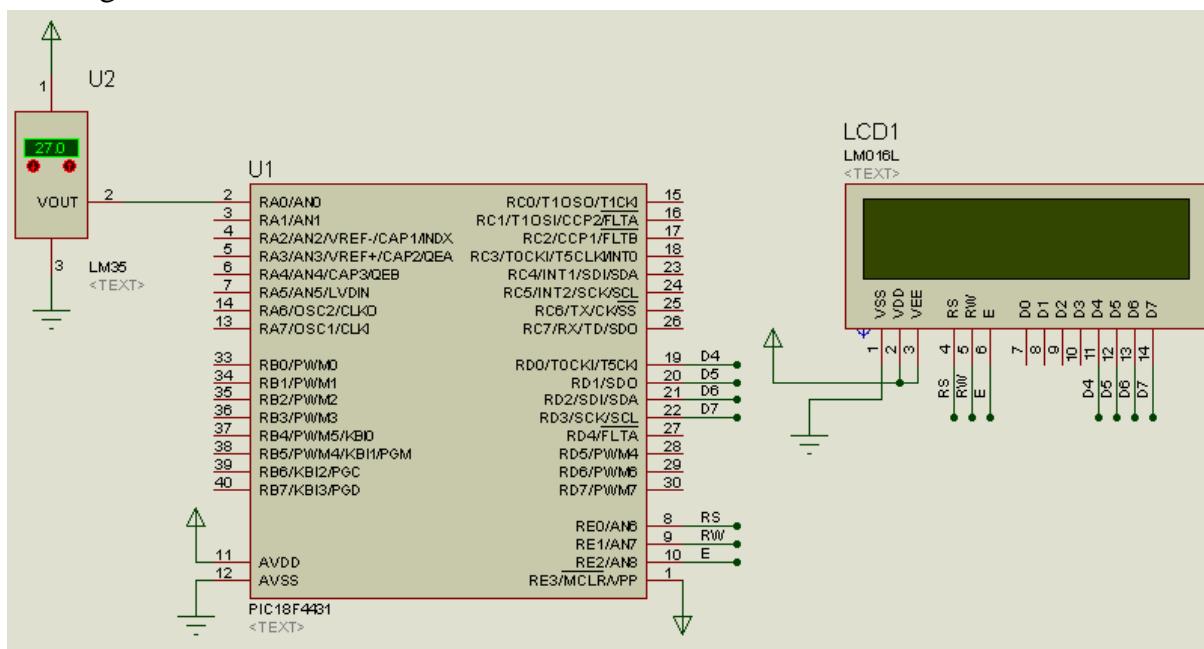
a) Cảm biến NTC 10K



b) Cảm biến LM35

Hình 3.5. Các loại cảm biến nhiệt độ thông dụng

Sơ đồ nguyên lý mô phỏng sử dụng vi điều khiển 18F4431 để đo lường nhiệt độ sử dụng LM25 minh họa như hình 3.6



Hình 3.6. Nguyên lý mô phỏng đo lường nhiệt độ

Chương trình điều khiển đọc giá trị nhiệt độ và hiện thị LCD như sau:

```
#include <xc.h>
#include "18F_LCD.h" //Khai bao thu vien LCD
#include <stdio.h>

#pragma config OSC = HS          // Dao dong tan so cao
#pragma config WDTEN = OFF        // Tat Watchdog Timer
#pragma config MCLRE = ON         // Chon reset ngoai

#define _XTAL_FREQ 20000000

char text[4];
```

```

//Cau hinh ADC (AN0 va AN1)
void Config_ADC()
{
    ANSEL0=0x01; //AN0 va AN1 1 kenh: 0x01
    ANSEL1=0x00; //AN8 off
    ADCON0=0x33; //GO/DONE
    ADCON1=0x10; //FIFO is enabled
    ADCON2=0x88; //Right justified ADCON2=0x80;
    ADCHS=0x00; //AN0, AN1, AN2 va AN3
}

//Doc gia tri nhiet do AN0. Tin hieu tra ve 0-5V
unsigned int Temp1;
float Temp2;
unsigned int Temp3;
float Temp4;
void Read_ADC_AN0()
{
    Temp1=ADRESH;
    Temp1=Temp1<<8;
    Temp1+=ADRESL;
    Temp2=4.88e-3*Temp1*0.097;
}
}

void main(void)
{
    TRISAbits.RA0=1; //Khai bao dau vao
    LCD_Init();
    Config_ADC();
    LCD_Clear();
    LCD_Gotoxy(1,0);
    LCD_Write_String("Nhiet do=");

    while(1)
    {
        Read_ADC_AN0();
        Read_ADC_AN1();
        sprintf(text,"%4d",Temp1);
        LCD_Gotoxy(1,10);
        LCD_Write_String(text);
    }
}

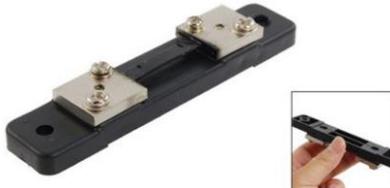
```

}

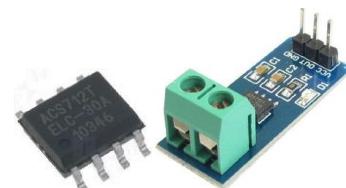
}

### 3.4.2. Lập trình giao tiếp với cảm biến dòng điện, điện áp

Để đo giá trị dòng điện của một mạch điện, hoặc giá trị dòng điện qua tải chúng ta có thể sử dụng các thiết bị như điện trở Shunt, cảm biến ACS712 -xxA, cảm biến Hall, ... (xem hình 3.7). Các thiết bị trên sẽ trả về giá trị điện áp tương tự, sau đó sử dụng mạch chuẩn hóa tín hiệu về dạng 0-5V để đưa vào kênh ADC của vi điều khiển để xử lý. Thông thường giá trị điện áp trả về của thiết bị đo dòng điện có giá trị nhỏ cỡ mV ta sử dụng mạch khuếch đại vi sai minh họa như hình 7, với hệ số khuếch đại (-R<sub>2</sub>/R<sub>1</sub>)



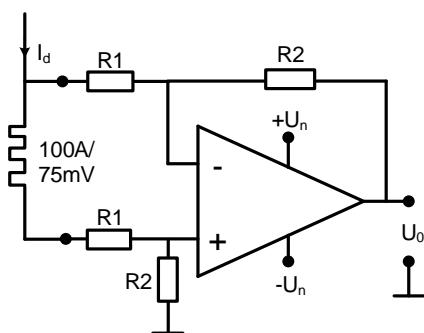
a) Điện trở Shunt 30A/75mV



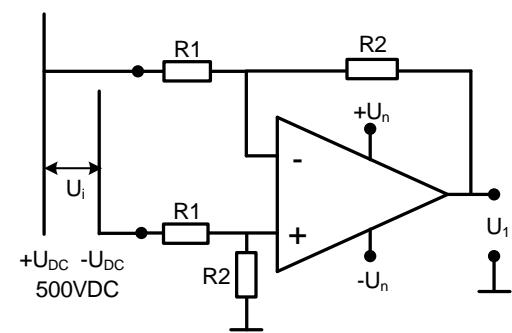
b) ACS 712 -30A (180 - 190 mV/A)

Hình 3.7. Thiết bị đo dòng điện

Muốn giám sát giá trị điện áp thông thường là đo giá trị điện áp cao tại các mạch lực của bộ biến đổi công suất để phát hiện sự cố khi cao áp, thấp áp; ta có thể sử dụng mạch khuếch đại vi sai dạng suy giảm điện áp với hệ số suy giảm (-R<sub>2</sub>/R<sub>1</sub>) như hình 3.8.

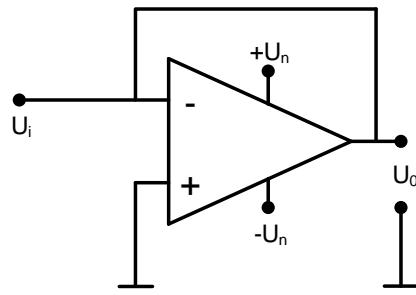


Hình 3.8. Mạch khuếch đại vi sai sử dụng OA



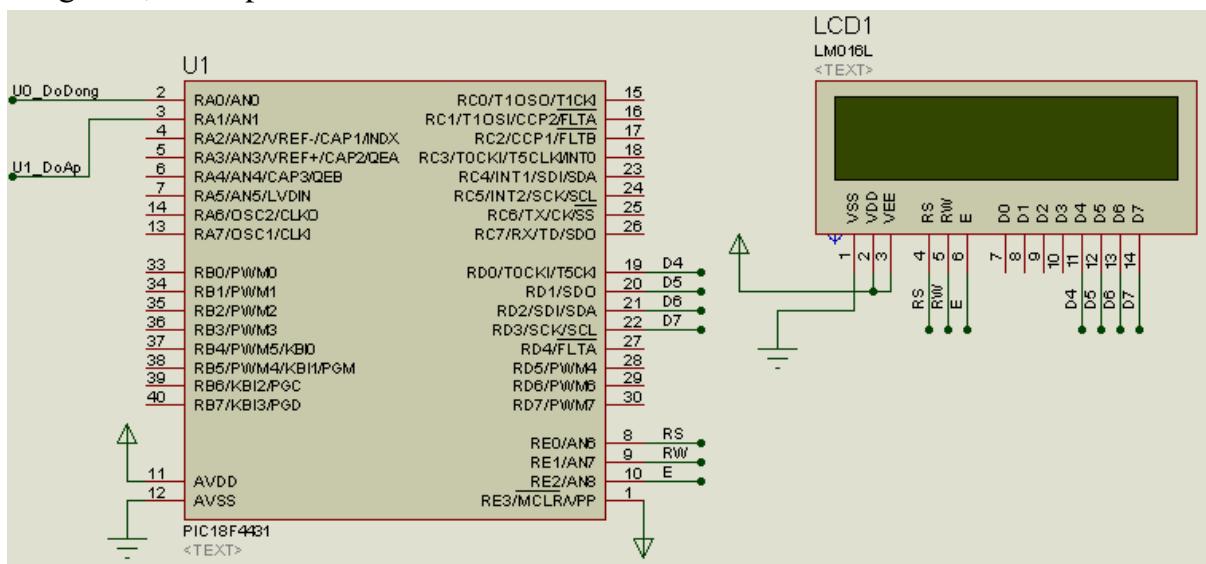
Hình 3.9. Mạch đo lường điện áp sử dụng OA

Để có được dạng điện áp phù hợp, ta nối tiếp mạch hình 3.8, 3.9 với mạch khuếch đại đảo để có được tín hiệu khuếch đại dương (hình 3.10)



Hình 3.10. Mạch khuếch đại đảo (hệ số khuếch đại -1)

Sơ đồ nguyên lý mô phỏng sử dụng vi điều khiển 18F4431 để đo lường tín hiệu dòng điện, điện áp minh họa như hình 3.11.



Hình 3.11. Nguyên lý mô phỏng đo lường tín hiệu dòng điện, điện áp

Chương trình điều khiển đọc giá trị dòng điện, điện áp và hiện thị LCD như sau:

```
#include <xc.h>
#include "18F_LCD.h" //Khai bao thu vien LCD
#include <stdio.h>
#pragma config OSC = HS          // Dao dong tan so cao
#pragma config WDTEN = OFF        // Tat Watchdog Timer
#pragma config MCLRE = ON         // Chon reset ngoai

#define _XTAL_FREQ 20000000

char text[4];
//Cau hinh ADC (AN0 va AN1)
void Config_ADC()
{
    ANSEL0=0x03; //AN0 va AN1 1 kenh: 0x01
    ANSEL1=0x00; //AN8 off
```

```

    ADCON0=0x33; //GO/DONE
    ADCON1=0x10; //FIFO is enabled
    ADCON2=0x88; //Right justified ADCON2=0x80;
    ADCHS=0x00; //AN0, AN1, AN2 va AN3
}

//Doc tin hieu dong dien AN0. Tin hieu tra ve 0-5V
unsigned int Temp1;
float Temp2;
unsigned int Temp3;
float Temp4;
void Read_ADC_AN0()
{
    Temp1=ADRESH;
    Temp1=Temp1<<8;
    Temp1+=ADRESL;
    Temp2=4.88e-3*Temp1/0.097; %He so khuech dai 100
}
//Doc tin hieu dien ap AN1. Tin hieu tra ve 0-5V
void Read_ADC_AN1()
{
    Temp3=ADRESH;
    Temp3=Temp3<<8;
    Temp3+=ADRESL;
    Temp4=4.88e-3*Temp3*0.097; %5V tuong ung 100A
}
void main(void)
{
    TRISAbits.RA0=1; //Khai bao dau vao
    LCD_Init();
    Config_ADC();
    LCD_Clear();
    LCD_Gotoxy(1,0);
    LCD_Write_String("Dong Dien=");
    LCD_Gotoxy(1,0);
    LCD_Write_String("Dien ap=");
    while(1)
    {
        Read_ADC_AN0();
        Read_ADC_AN1();
    }
}

```

```

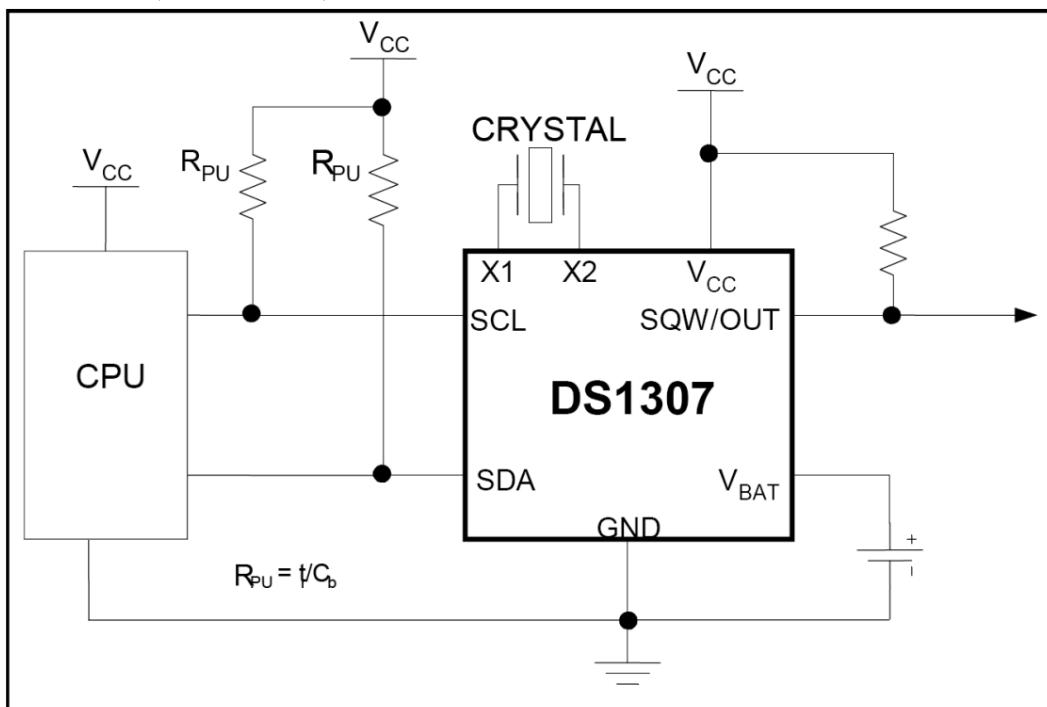
        sprintf(text,"%4d",Temp1);
        LCD_Gotoxy(1,11);
        LCD_Write_String(text);
        sprintf(text,"%4d",Temp3);
        LCD_Gotoxy(2,9);
        LCD_Write_String(text);
    }
}

```

### 3.5. LẬP TRÌNH ỨNG DỤNG SỬ DỤNG GIAO TIẾP I2C

#### 3.5.1. Giao tiếp DS1307

DS1307 là một bộ đồng hồ thời gian thực của hãng Maxim có tích hợp giao thức I2C. Vi mạch này cung cấp thông tin về năm, tháng, ngày, giờ, phút, giây dưới dạng mã BCD. Thông tin về lịch sẽ được vi mạch tự cập nhật sau khi được cài đặt. Thời gian cập nhật lên tới năm 2100. Hình 3.12 mô tả cách kết nối vi mạch này với bộ vi điều khiển. Để có thể hoạt động, vi mạch này cần một nguồn PIN (3V) và một bộ dao động thạch anh (32.768 KHz).



Hình 3.12. Mạch điện giao tiếp DS1307 với vi điều khiển

Người dùng có thể cài đặt thời gian thực cho vi mạch bằng cách ghi thông tin thời gian vào các thanh ghi tương ứng của DS1307 (hình 3.13). Sau khi được cài đặt, DS1307 sẽ trở thành một đồng hồ thời gian thực (real time clock). Thông tin về thời gian được đọc bằng cách truy xuất các thanh ghi tương ứng.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE		
00h	CH		10 Seconds		Seconds				Seconds	00–59		
01h	0		10 Minutes		Minutes				Minutes	00–59		
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23		
		24	PM/ AM									
03h	0	0	0	0	0	DAY		Day	01–07			
04h	0	0	10 Date		Date				Date	01–31		
05h	0	0	0	10 Month	Month				Month	01–12		
06h	10 Year				Year				Year	00–99		
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—		
08h–3Fh									RAM 56 x 8	00h–FFh		

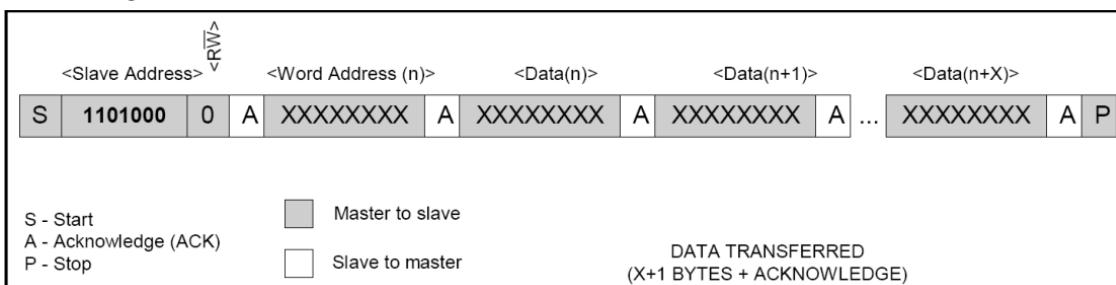
Hình 3.13. Các thanh ghi của DS1307.

Bit 7 của thanh ghi seconds (địa chỉ 00h) là bit clock halt (CH). CH=1, mạch tao dao động (oscillator) trên DS1307 sẽ bị cấm, thông tin về thời gian trên vi mạch sẽ không được cập nhật. Chức năng này được sử dụng cho mục đích tiết kiệm năng lượng trên PIN khi cần thiết (DS1307 chỉ sử dụng dòng điện 10nA ở chế độ oscillator off thay vì 480 nA ở chế độ oscillator on). Khi xuất xưởng, nha sản xuất mặc định CH=1, ngay/tháng/năm-giờ/phút:giây = 01/01/00-00:00:00.

Bit 6 trong thanh ghi hours (địa chỉ 02h) dùng để cài đặt chế độ 12h (=1) hoặc chế độ 24h (=0). Ở chế độ 12h, đọc bit 5 trên thanh ghi này sẽ cho thông tin về thời gian: buổi sáng (=0), buổi chiều (=1). Ở chế độ 24h, bit 5 bằng 1 khi thời gian hiện tại là 20h đến trước 24h.

Thanh ghi control (địa chỉ 07h) sử dụng điều khiển xung vuông phát ra từ chân SQW của DS1307.

Trình tự ghi trên DS1307:



Hình 3.14.. Ghi các thanh ghi/ô nhớ trên DS1307

Tù vi điều khiển, tạo “start”:

```
StartI2C();  
IdleI2C(); //chờ bus rảnh (idle)
```

Ghi byte chứa địa chỉ thiết bị (slave address=1101000) và bit R/W(=0):

```
WriteI2C(0xd0);  
IdleI2C();
```

Ghi byte chứa địa chỉ của thanh ghi đầu tiên cần ghi (word address (n)). Đây thực chất là một con trỏ (pointer) cho phép từ thiết bị chủ có thể lựa chọn bắt đầu quá

trình ghi từ địa chỉ nào trong không gian từ 0x00 đến 0xff trên DS1307. Ví dụ: câu lệnh dưới đây sẽ ghi vào word address giá trị 0x01:

```
WriteI2C(0x01);
IdleI2C();
```

Điều này đồng nghĩa với các lệnh ghi tiếp theo sẽ ghi vào các thanh ghi/ô nhớ có địa chỉ 0x01 (thanh ghi phút), 0x02....

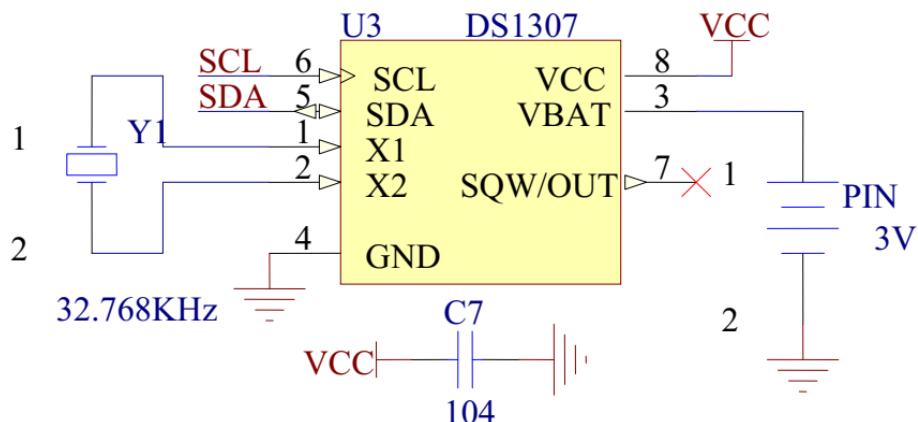
Ghi vào các thanh ghi/ô nhớ bắt đầu từ địa chỉ word address:

```
WriteI2C(0x06);           //ghi vào thanh ghi địa chỉ 0x01,
                           //giá trị 0x06
IdleI2C();
WriteI2C(0x07);           //ghi vào thanh ghi địa chỉ 0x02,
                           //giá trị 0x07
IdleI2C();
```

Tạo tín hiệu stop kết thúc quá trình ghi:

```
StopI2C();
```

Ví dụ 1. Cho sơ mạch điện như hình 1.17. Viết chương trình đọc các thanh ghi giây (địa chỉ 0x00), phút, giờ và hiển thị trên LCD. Ứng dụng trong các bài toán điều khiển theo thời gian thực.



Hình 3.15. Giao tiếp PIC18F4520 với DS1307

Dưới đây là chương trình tham khảo:

```
#include<p18f4520.h>
#include<delays.h>
#include<stdio.h>
#include<i2c.h>

#define lcd_data PORTD
#define RS PORTAbits.RA0
#define RW PORTAbits.RA1
```

```

#define E PORTAbits.RA2
Unsigned int s,m,h;
char M[32];
void lcd_int(void);
void lcd_cmd(unsigned char cmd);
void lcd_write(unsigned char data);
void lcd_str(unsigned char *str);
char bcd_int(int x);
int int_bcd(int x);
//*****cac ham cua LCD*****
Void lcd_int(void)
{
lcd_cmd(0x01);
lcd_cmd(0x38);
lcd_cmd(0x0c);
lcd_cmd(0x06);
lcd_cmd(0x01);
}
Void lcd_cmd (unsigned char cmd)
{
RW=0;
RS=0;
E=1;
lcd_data=cmd;
E=0;
Delay1KTCYx(10);
}

Void lcd_write(unsigned char data)
{
if(data=='\n')
{
lcd_cmd(0xc0);
Delay1KTCYx(10);
return;
}
RW=0;
}

```

```

RS=1;
E=1;
lcd_data=data;
E=0;
Delay1KTCYx(10);
}

Void lcd_str(unsigned char*str)
{
while(*str)
{
lcd_write(*str);
str++;
}

/****************************************/
//Cac ham chuyen doi so nguyen, BCD
Char bcd_int(int x)
{
return (((x>>4)&0x0f)*10)+(x&0x0f);
}

Int int_bcd(int x)
{
char
N[10]={0X00,0X01,0X02,0X3,0X4,0X05,0X06,0X07,0X08,0X09};
int a,b;
a=x/10;
b=x%10;
return ((N[a]<<4)&0xf0)+N[b];
}

/****************************************/
void main(void)
{
unsigned int a;
TRISA=0X00;
TRISB=0X0F;
TRISC=0X00;
TRISD=0X00;
}

```

```

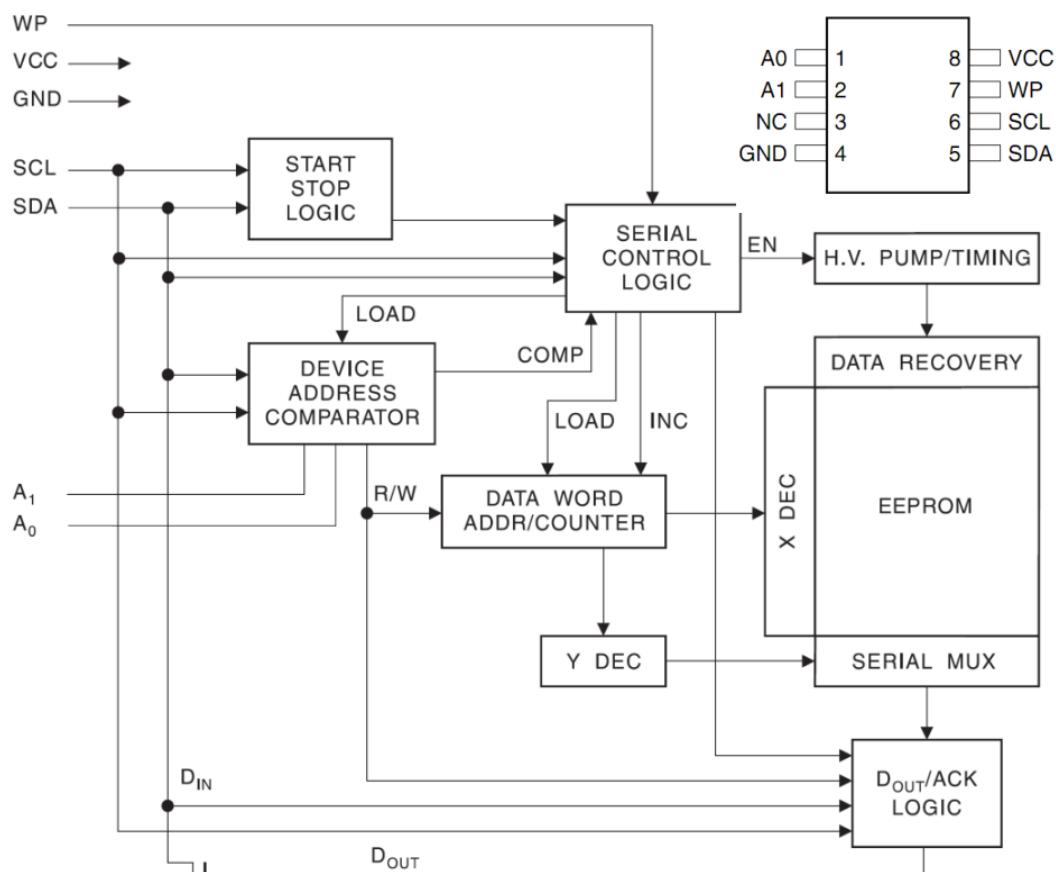
TRISE=0X00;
ADCON1=0X0F;
OpenI2C(MASTER,SLEW_OFF); //master mode, clock=100Khz
lcd_int();
while(1)
{
StartI2C();
IdleI2C();
WriteI2C(0xd0);
IdleI2C();
WriteI2C(0x00);
IdleI2C();
RestartI2C();
IdleI2C();
WriteI2C(0xd1);
IdleI2C();
s=ReadI2C();
IdleI2C();
AckI2C();
IdleI2C();
m=ReadI2C();
IdleI2C();
AckI2C();
IdleI2C();
h=ReadI2C();
IdleI2C();
NotAckI2C();
IdleI2C();
StopI2C();
s=bcd_int(s);
m=bcd_int(m);
h=bcd_int(h);
lcd_cmd(0x80);
sprintf(&M[0],"TIME:
%d%d:%d%d:%d%d",h/10,h%10,m/10,m%10,s/10,s%10);
lcd_str(&M[0]);
}

```

### 3.5.2. Giao tiếp EEPROM 24C256

24C256 là vi mạch nhớ EEPROM của hãng Atmel được tích hợp giao thức I2C. Vì mạch này có một số tính năng chính sau:

- Dung lượng 32768 x 8 bit (32 Kbyte)
- Có thể hoạt động ở dải điện áp rộng (2,7÷5,5V hoặc 1,8÷3,6V)
- Có thể hoạt động với xung clock ở 03 tần số: 1Mhz, 400Khz và 100Khz.
- Có chức năng bảo vệ dữ liệu bằng cả phần cứng và phần mềm. Dữ liệu được ghi có thể lưu được trong 40 năm.



Hình 3.16. Sơ đồ khái niệm và sơ đồ chân của vi mạch 24C256

Các khối chính trên 24C256:

- EEPROM: Gồm 32Kbyte x 8bit được chia thành 512 page (mỗi page gồm 64 byte). Từ vi mạch chủ có thể ghi/đọc từng byte hoặc ghi/đọc cả 64 byte trong cùng một page.
- Khối so sánh địa chỉ (address comparator). Vi mạch 24C256 có 2 chân địa chỉ (A0, A1) quy định địa chỉ của từng vi mạch khi chúng được ghép trên cùng bus. Mức logic trên 2 chân này sẽ được so sánh với 2 bit trong byte địa chỉ mà 24C256 nhận được từ vi mạch chủ. Nếu trùng khớp nghĩa là vi mạch chủ muốn đọc/ghi trên bộ nhớ

EEPROM tương ứng. Cách đặt địa chỉ bằng phần cứng này cho phép có tối đa 4 vi mạch 24C256 cùng kết nối trên bus.

- Khối Start Stop Logic và khối Serial Control Logic: Tao thành module I2C.

Ví dụ 1. Ghi vào ô nhớ có địa chỉ 0x0006 của vi mạch 24C256 byte dữ liệu 0x41 sau đó, đọc từ ô nhớ này ra và hiển thị trên LCD (vi mạch EEPROM được ghép nối với PIC18F4520 như hình 3.15).

Dưới đây là chương trình tham khảo:

```
Void main(void)
{
    unsigned char a;

    TRISA=0X00;
    TRISB=0X0F;
    TRISC=0X00;
    TRISD=0X00;
    TRISE=0X00;
    ADCON1=0X0F;
    OpenI2C(MASTER,SLEW_OFF);
    lcd_int();
    //ghi vào ROM (byte write)
    //(1)
    StartI2C();
    IdleI2C();
    //(2)
    WriteI2C(0b10100000); //A0A1=00, R/W=0
    IdleI2C();
    WriteI2C(0x06);
    IdleI2C();
    //(4)
    WriteI2C(0x41); //ma ky tu "A"
    IdleI2C();
    //(5)
    StopI2C();
    while(1)
    {
        //doc tu ROM
        //(1)
```

```

StartI2C();
IdleI2C();
// (2)
WriteI2C(0b10100000); //A0A1=00, R/W=0
IdleI2C();
// (3)
WriteI2C(0x00);
IdleI2C();
WriteI2C(0x06);
IdleI2C();
// (4)
RestartI2C();
IdleI2C();
// (5)
WriteI2C(0b10100001); //A0A1=00, R/W=1
IdleI2C();
// (6) Lưu ý, biên a là biên kiểu char
a=ReadI2C();
IdleI2C();
// (7)
NotAckI2C();
IdleI2C();
StopI2C();
//hienthi
lcd_cmd(0x80);
lcd_write(a); } }

```

### **3.6. LẬP TRÌNH ÚNG DỤNG SỬ DỤNG GIAO THỨC SPI**

SPI (Serial Perippheral Interface - Giao tiếp ngoại vi nối tiếp) do hãng Motorola phát minh. Ngoài ra SPI còn được biết đến với tên gọi khác là Microwire (hãng National Semiconductor phát triển).

SPI cung cấp một giao thức nối tiếp giữa vi điều khiển và các thiết bị ngoại vi. SPI ngày càng được sử dụng rộng rãi trong lĩnh vực điện tử, đặc biệt là trong giao tiếp trao đổi dữ liệu với các ngoại vi. SPI là một giao thức nối tiếp đồng bộ. Dữ liệu truyền nhận giữa các thiết bị được giữ nhịp bởi một xung đồng bộ duy nhất phát ra từ một thiết bị chủ (master) trong hệ thống.

Bus của SPI gồm 04 đường dây (MISO, MOSI, SCK, SS) nên đôi khi SPI còn được gọi là giao thức giao tiếp 4 dây.

- MISO (Master Input Slave Output).

Khi một module SPI (có thể là vi diệu khiển hoặc thiết bị ngoại vi) được cấu hình là thiết bị chủ (master) chân MISO được dùng để nhận dữ liệu. Ngược lại, khi module SPI được cấu hình là thiết bị tớ (slave) chân MISO được dùng để truyền dữ liệu.

- MOSI (Master Output Slave Input).

Khi một module SPI được cấu hình là thiết bị chủ chân MOSI được dùng để truyền dữ liệu. Ngược lại, khi module SPI được cấu hình là thiết bị tớ chân MOSI được dùng để nhận dữ liệu.

- SCK (Serial Clock).

Chân SCK cung cấp xung đồng bộ từ thiết bị chủ để truyền nhận dữ liệu với một thiết bị tớ nào đó được chọn bởi SS.

- SS (Slave Select).

SS là chân tín hiệu chọn chip (tích cực ở mức thấp) khi một module SPI được cấu hình là thiết bị tớ. Đối với thiết bị chủ, chân SS sẽ cung cấp một tín hiệu chọn chip đến một thiết bị tớ có kết nối với nó.

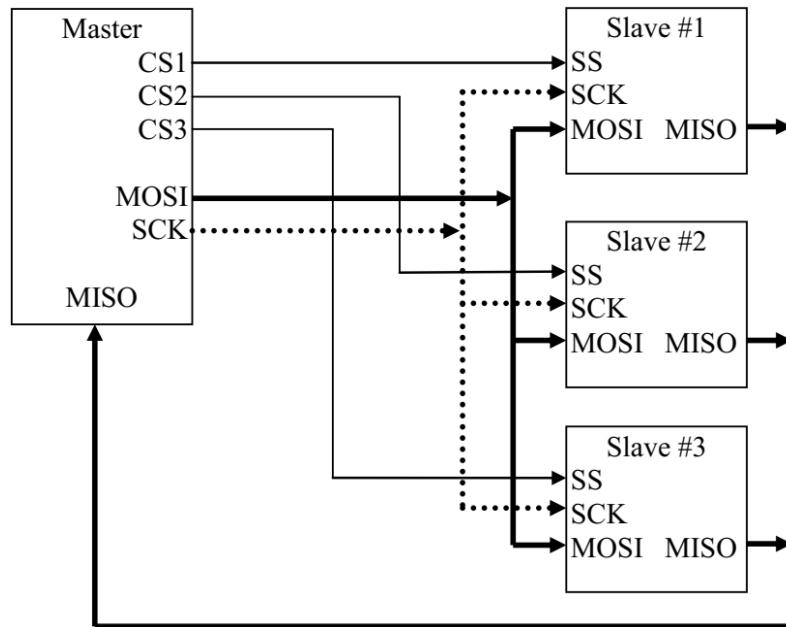
Trong một hệ thống gồm nhiều thiết bị giao tiếp với nhau, có 02 cách để kết nối thiết bị chủ với các thiết bị tớ.

Cách 1: Thiết bị chủ giao tiếp với các thiết bị tớ độc lập (hình 3.17)

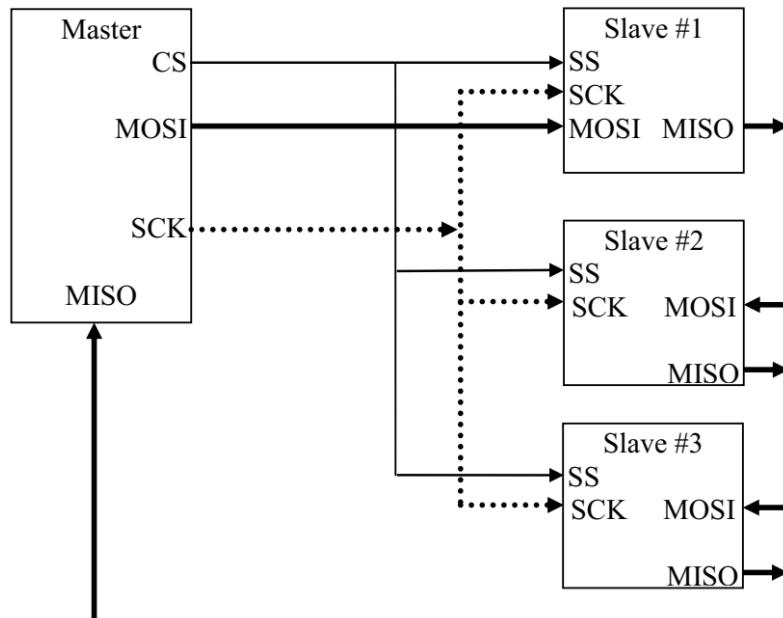
Trong cách kết nối này, tín hiệu SCK và MOSI từ Master được cung cấp đến từng Slave. Đường tín hiệu MISO của các Slave nối chung lại với nhau và truyền về Master. Lúc này, Master sẽ lựa chọn các chip Slave riêng lẻ thông qua các chân CS để trao đổi dữ liệu.

Cách 2: Thiết bị chủ giao tiếp với các thiết bị tớ theo chuỗi (Daisy-Chained)

Hình 3.18 mô tả cách giao tiếp giữa một Master và 03 Slave. Các Slave luôn hoạt động đồng thời với nhau. Một byte dữ liệu truyền từ Master sẽ tới Slave #1 trước tiên, sau đó sẽ được dịch sang Slave #2, Slave #3 và cuối cùng quay trở lại Master.



Hình 3.17. Thiết bị chủ kết nối với các thiết bị tách độc lập



Hình 3.18. Thiết bị chủ kết nối với các thiết bị thành chuỗi.

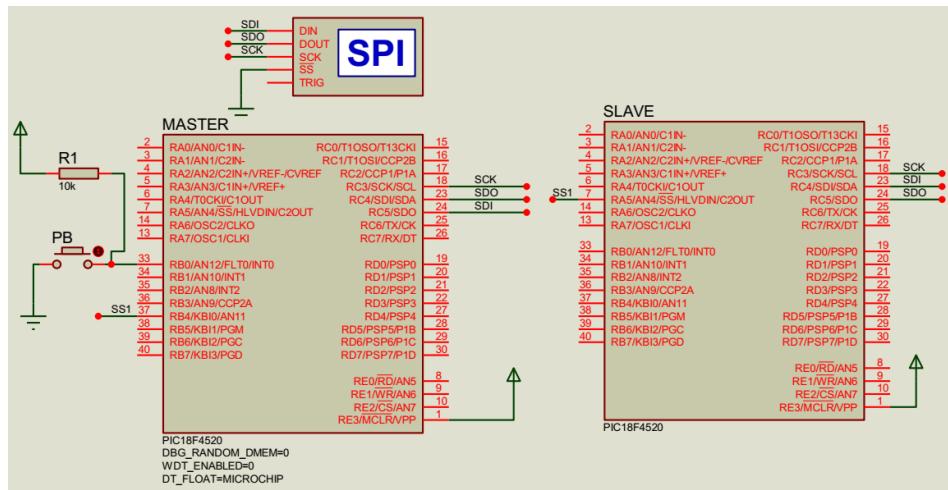
Ví dụ 1. Cho sơ đồ mô phỏng giao tiếp SPI như hình 3.19. Ứng dụng trong bài toán đo lường, truyền thông.

Viết chương trình trên master thực hiện các yêu cầu:

- Khởi tạo module SPI
- Đếm số lần nhấn PB
- Truyền đi chữ số hàng đơn vị của số lần nhấn (giả thiết số lần nhấn không vượt quá 65535).
- Đọc bus SPI và hiển thị giá trị trên PORTD

Viết chương trình trên slave thực hiện các yêu cầu:

- Khởi tạo module SPI
- Đọc bus SPI và hiển thị giá trị trên PORTD



Hình 3.19. Mô phỏng giao tiếp SPI giữa 2 vi điều khiển

Dưới đây là chương trình tham khảo:

Chương trình trên master:

```
#include<p18f4520.h>
#include <spi.h>

#pragma config OSC = HS
#pragma config MCLRE = ON
#pragma config WDT = OFF

void main(void)
{
    unsigned int i1;
    TRISD=0x00;
    TRISB=0x01;
    ADCON1=0X0f;
    TRISC=0X10;      //SDI=vao;      SDO=SCK=ra
    SSPCON1bits.SSPEN=1; //cho phep module SPI
    //che do master, clock=Fosc/4
    SSPCON1bits.SSPM3=0;
    SSPCON1bits.SSPM2=0;
    SSPCON1bits.SSPM1=0;
    SSPCON1bits.SSPM0=0;
    //bit nhan duoc lay mau o giua cua bit truyen
    SSPSTATbits.SMP=0;
```

```

//bus SPI che do 0,0
SSPSTATbits.CKE=0;
SSPCON1bits.CKP=0;
PORTBbits.RB4=0; //Slave1=on

while(1)
{
    while(PORTBbits.RB0); //cho nhan PB
    while(!PORTBbits.RB0); //cho nha PB
    while(SSPCON1bits.WCOL); //cho den khi het xung dot
    (WCOL=0)

    SSPBUF=i1%10; //truyen di
    while(!SSPSTATbits.BF); //cho bo dem day
    SSPSTATbits.BF=0; //xoa co bao tran
    PORTD=SSPBUF; //doc byte du lieu tu bo dem
    va hien thi tren PORTD

    ++i1;
}
}

```

### Chương trình trên slave:

```

#include<p18f4520.h>
#include <spi.h>
#pragma config OSC = HS
#pragma config MCLRE = ON
#pragma config WDT = OFF
void main(void)
{
    unsigned int i;
    TRISD=0x00;
    ADCON1=0X0F;
    TRISC=0X18;

    //OpenSPI(SLV_SSON, MODE_01, SMPMID);
    SSPCON1bits.SSPEN=1; //cho phep module SPI
    //che do slave, su dung chuc nang cua chan SS
    SSPCON1bits.SSPM3=0;
    SSPCON1bits.SSPM2=1;
}

```

```

SSPCON1bits.SSPM1=0;
SSPCON1bits.SSPM0=0;
//bit nhan duoc lay mau o giua cua bit truyen
SSPSTATbits.SMP=0;
//bus SPI che do 0,0
SSPSTATbits.CKE=0;
SSPCON1bits.CKP=0;
while(1)
{
while(!SSPSTATbits.BF); //cho bo dem day
SSPSTATbits.BF=0; //xoa co bao tran
PORTD=SSPBUF; //doc byte du lieu va hien thi tren
PORTD} }

```

### **3.7. LẬP TRÌNH BỘ ĐIỀU KHIỂN PID**

Bộ điều khiển PID được sử dụng rộng rãi trong các ứng dụng thực tiễn bởi tính linh hoạt, dễ triển khai và giá thành rẻ. Năm bắt được vai trò và xu hướng sử dụng vi điều khiển trong tương lai, nhiều công ty đã cho ra đời nhiều mẫu vi điều khiển có cấu hình mạnh, tốc độ cao, bộ nhớ lớn, tích hợp nhiều chức năng trên phiến. Đi kèm với đó là các công cụ, trình biên dịch ngày càng đơn giản thuận tiện cho người sử dụng.

Một hệ thống điều khiển số bao gồm hai phần chính: phần cứng và phần mềm. Phần cứng với trung tâm xử lý là một MCU kết hợp cùng các mạch điện tử ngoại vi phục vụ cho việc đo lường và điều khiển hệ thống.

#### **3.7.1. Một số vấn đề kỹ thuật khi thực hiện hệ điều khiển số với vi điều khiển.**

Đối với hệ thống điều khiển số khi thiết kế cần chú ý một số điểm sau:

- Ảnh hưởng của giá trị chu kỳ lấy mẫu đến đặc tính hệ thống: Nhìn chung chu kỳ lấy mẫu càng nhỏ càng tốt. Khi đó hệ thống điều khiển số càng tiệm cận với hệ thống thực. Tuy nhiên lúc này lại đòi hỏi năng lực, tốc độ tính toán của MCU phải lớn dẫn tới giá thành hệ thống cao.
- Ảnh hưởng của các khâu biến đổi A/D, D/A lên hệ thống bao gồm các vấn đề: lượng tử hóa và sai số do lượng tử hóa, đặc tính giới hạn của các khâu biến đổi.
- Độ chính xác của cảm biến.
- Các phép tính trong luật điều khiển số thường được quy về kiểu tính số nguyên. Đây là cách tính toán phổ biến trong các hệ thống nhúng có tài nguyên hữu hạn. Các phép tính số nguyên sẽ tạo ra sai số tính toán và xuất hiện hiện tượng tràn số liệu trong trường hợp nhất định.

### 3.7.2. Quy trình thực hiện hệ thống điều khiển số

Để thực hiện một hệ thống điều khiển số trên vi điều khiển người thiết kế thường phải thực hiện một số bước quy trình thiết kế như sau:

Bước 1: Dựa vào yêu cầu của bài toán thiết kế điều khiển để xây dựng và phác thảo sơ đồ cấu trúc và các thành phần, đối tượng, thiết bị cấu thành hệ thống.

Bước 2: Xây dựng các mô hình toán học từng phần tử trong hệ thống từ đó xác định được mô hình toán học của toàn hệ thống dưới dạng phương trình trạng thái hoặc hàm truyền đạt.

Bước 3: Lựa chọn chu kỳ lấy mẫu và rời rạc hóa hệ thống(nếu thiết kế theo phương pháp rời rạc)

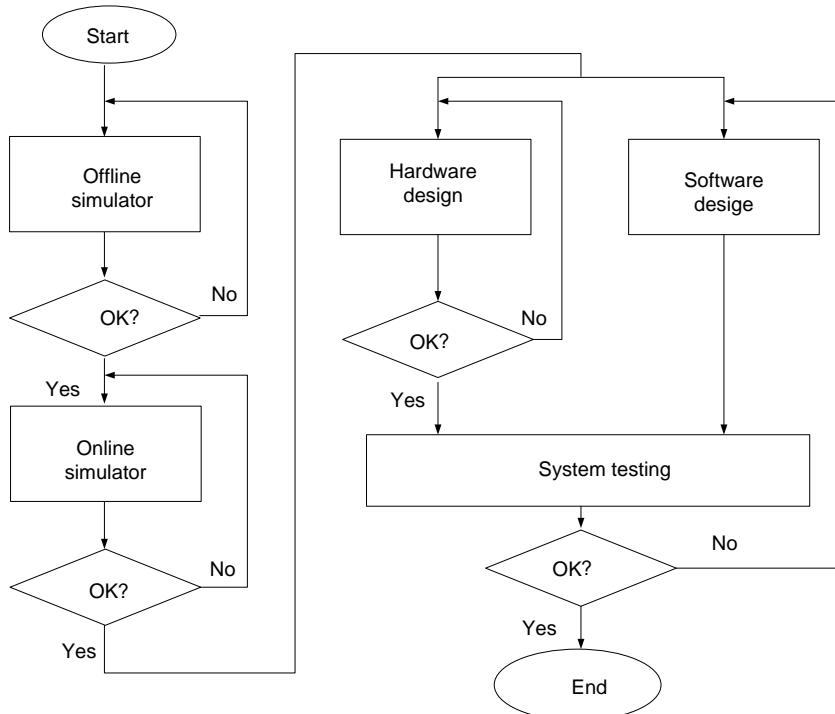
Bước 4: Sử dụng lý thuyết điều khiển để xác định bộ điều khiển một cách phù hợp với yêu cầu điều khiển (có thể sử dụng kỹ thuật điều khiển liên tục hoặc kỹ thuật điều khiển rời rạc)

Bước 5: Thực hiện mô phỏng offline hệ thống trên phần mềm chuyên dụng(Matlab, LabVIEW...) để hiệu chỉnh tham số bộ điều khiển cho phù hợp.

Bước 6: Mô phỏng online hệ thống với phần mềm và phần cứng để hiệu chỉnh tham số bộ điều khiển.

Bước 7: Thiết kế phần cứng, phần mềm hệ thống trên thiết bị tính toán số (vi điều khiển, FPGA, DSP, PLC...)

Bước 8: Chạy thử nghiệm và hiệu chỉnh hệ thống thực



Hình 3.20. Lưu đồ quy trình mô phỏng và thực hiện hệ thống thực

Công đoạn từ bước 1 đến bước 4 là quá trình xây dựng mô hình, phân tích và thiết kế hệ thống. Từ bước 5 đến bước 8 mô tả quá trình xây dựng và hiệu chỉnh hệ thống thực. Với cấu trúc lưu đồ thực hiện trên hình 3.20 có thể thấy trọng tâm của việc hiệu chỉnh hệ thống đặt vào phần mềm. Tuy nhiên có những trường hợp hiệu chỉnh phần mềm không thể đạt được kết quả mong muốn khi đó cần phải tiến hành hiệu chỉnh thiết kế phần cứng một cách phù hợp.

### 3.7.3. Luật điều khiển PID

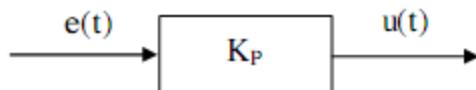
Bộ điều khiển PID (Proportional Integral Derivative controller) là bộ điều khiển sử dụng kỹ thuật điều khiển theo vòng lặp có hồi tiếp được sử dụng rộng rãi trong các hệ thống điều khiển tự động, nó hiệu chỉnh sai lệch giữa tín hiệu ra và vào sau đó đưa ra một tín hiệu điều khiển để điều chỉnh quá trình cho phù hợp.

Bộ điều khiển PID (vi tích phân tỉ lệ) rất hay dùng trong các hệ thống điều khiển. Vì nó tăng chất lượng đáp ứng của hệ thống với các ưu điểm sau: PID là sự kết hợp ưu điểm của hai khâu PD và PI, nó làm giảm thời gian xác lập, tăng tốc độ đáp ứng của hệ thống, giảm sai số xác lập, giảm độ vọt lô,...

Theo loại tín hiệu làm việc mà chia thành ba loại chính là bộ điều chỉnh liên tục, bộ điều chỉnh on-off và bộ điều chỉnh số. Bộ điều chỉnh liên tục có thể thực hiện bằng các cơ cấu cơ khí, thiết bị khí nén, mạch điện RC, mạch khuếch đại thuần.

#### ✓ Các bộ điều chỉnh liên tục gồm bộ P, I, PI, PD, PID

**Bộ điều chỉnh tỉ lệ P** (Proportional): Bộ điều chỉnh tỉ lệ tạo tín hiệu điều khiển  $u_p(t)$  tỉ lệ với tín hiệu sai lệch  $e(t)$



Hình 3.21. Sơ đồ khối bộ điều chỉnh tỉ lệ

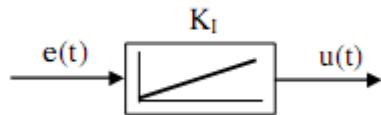
Phương trình vi phân:  $u_p(t) = K_p \cdot e(t)$

Trong đó  $K_p$  gọi là hệ số khuếch đại

Hàm truyền trong miền Laplace:  $G_p(s) = K_p$ .

- **Bộ điều chỉnh tích phân I** (Integration):

Bộ điều chỉnh tích phân tạo tín hiệu điều khiển  $u_I(t)$  tỉ lệ với tích phân của tín hiệu sai lệch  $e(t)$



Hình 3.22. Sơ đồ khối bộ điều chỉnh tích phân

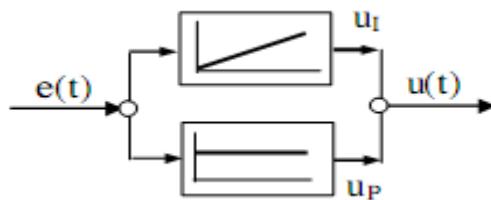
Phương trình vi phân:  $u_I(t) = K_I \cdot \int e(t) dt$

$$\text{Hàm truyền trong miền Laplace: } G_I(s) = \frac{U(s)}{E(s)} = \frac{K_I}{s}$$

Trong đó:  $K_I$  là hằng số tích phân

#### - Bộ điều chỉnh tỉ lệ - tích phân (PI)

Bộ điều chỉnh PI là cấu trúc ghép song song của khâu P và khâu I. Tín hiệu ra của bộ PI là tổng tín hiệu ra của hai khâu thành phần.



Hình 3.23. Sơ đồ khối của bộ điều chỉnh tỉ lệ - tích phân

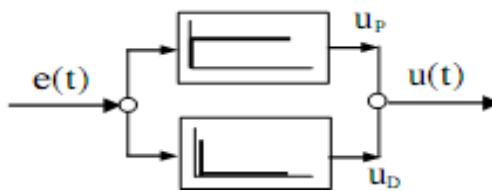
$$\text{Phương trình vi phân: } u(t) = K_p \cdot e(t) + K_I \cdot \int e(t) dt = K_p \left( e(t) + \frac{1}{T_I} \int e(t) dt \right)$$

Đặt  $T_i = \frac{K_p}{K_i}$  : hằng số thời gian tích phân.

$$\text{Hàm truyền trong miền Laplace: } G_{PI}(s) = K_p \left( 1 + \frac{1}{T_I s} \right)$$

#### - Bộ điều chỉnh tỉ lệ - vi phân (bộ PD)

Bộ điều chỉnh PD lý tưởng là cấu trúc ghép song song của khâu P và khâu D. Tín hiệu ra của bộ PD là tổng tín hiệu ra của hai thành phần.



Hình 3.24. Sơ đồ khối bộ điều chỉnh tỉ lệ - vi phân

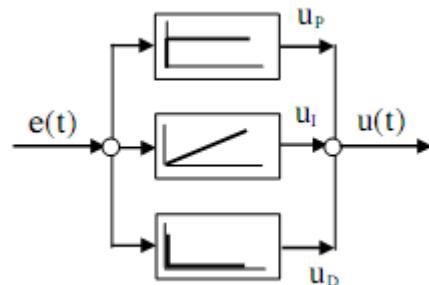
$$\text{Phương trình vi phân: } u(t) = K_p \cdot e(t) + K_D \cdot \frac{de(t)}{dt} = K_p \cdot \left( e(t) + T_d \cdot \frac{de(t)}{dt} \right)$$

Đặt  $T_d = K_D / K_p$  là hằng số thời gian vi phân.

Hàm truyền đạt trong miền Laplace:  $G_{PD}(s) = K_p(1 + T_d s)$

### - Bộ điều chỉnh tỉ lệ - vi tích phân (bộ PID)

Bộ điều chỉnh PID lý tưởng là cấu trúc ghép song song của ba khâu: P, I và D.



Hình 3.25. Sơ đồ khói của bộ điều chỉnh tỉ lệ - vi tích phân

Phương trình vi phân của bộ PID lý tưởng:

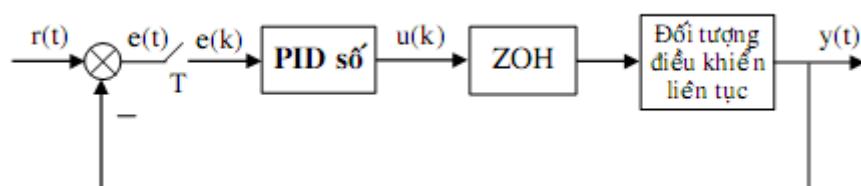
$$u(t) = K_p \cdot e(t) + K_D \cdot \frac{de(t)}{dt} + K_I \cdot \int e(t) dt = K_p \left( e(t) + T_d \cdot \frac{de(t)}{dt} + \frac{1}{T_I} \int e(t) dt \right)$$

Hàm truyền đạt trong miền Laplace:

$$G_{PID}(s) = K_p \left( 1 + T_d s + \frac{1}{T_I s} \right)$$

Trong thực tế có nhiều sơ đồ điều khiển khác nhau có thể áp dụng cho hệ rời rạc, nhưng sơ đồ thường được sử dụng là hiệu chỉnh nối tiếp với bộ điều khiển PID số.

Ta có sơ đồ điều khiển với bộ PID số:



Xuất phát từ mô tả toán học của bộ PID liên tục ở trên ta có:

$$u(t) = u_p(t) + u_D(t) + u_I(t)$$

$$u(t) = K_p \cdot e(t) + K_I \cdot \int e(t) dt + K_D \cdot \frac{de(t)}{dt}$$

- Khi chuyển sang mô hình rời rạc của bộ PID số thì  $u(t)$  thay bằng  $u_k = u(k)$ .

$$u_k = u_k^p + u_k^I + u_k^D$$

- Khâu tỉ lệ  $u_p(t) = K_p \cdot e(t)$  được thay bằng:  $u_k^p = K_p e_k$

Suy ra hàm truyền:  $G_p(z) = \frac{U_p(z)}{E(z)} = K_p$

- Khâu vi phân  $u_D(t) = K_D \cdot \frac{de(t)}{dt}$  được thay bằng sai phân lùi:  $u_k^D = K_D \frac{e_k - e_{k-1}}{T}$

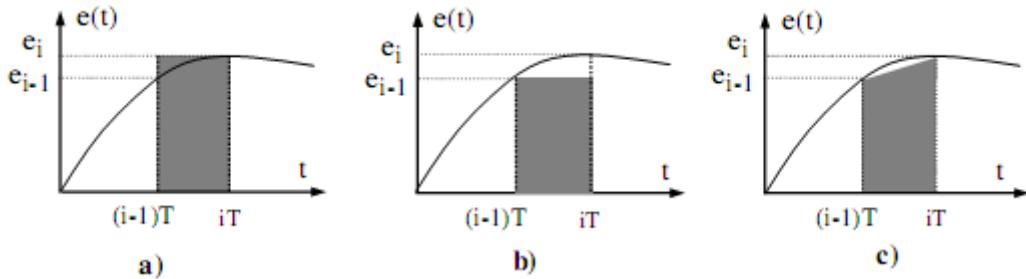
Biến đổi Z hai vế ta được:

$$U_D(z) = K_D \frac{E(z) - z^{-1}E(z)}{T} = \frac{K_D}{T} (1 - z^{-1}) E(z) = \frac{K_D}{T} \frac{z-1}{z} E(z)$$

Hàm truyền:  $G_D(z) = \frac{U_D(z)}{E(z)} = \frac{K_D}{T} \left[ \frac{z-1}{z} \right]$

Khâu tích phân  $u_I(t) = K_I \cdot \int_0^t e(t) dt$  có nhiều cách tính:

- Thứ nhất là tính tích phân chữ nhật lùi:  $u_k^I = K_I \sum_{i=1}^k T e_i = u_{k-1}^I + K_I T e_k$
- Thứ hai là tính tích phân chữ nhật tới:  $u_k^I = K_I \sum_{i=1}^k T e_{i-1} = u_{k-1}^I + K_I T e_{k-1}$
- Thứ ba là tính tích phân hình thang:  $u_k^I = K_I \sum_{i=1}^k T \frac{e_{i-1} + e_i}{2} = u_{k-1}^I + T \frac{e_{k-1} + e_k}{2}$



Hình 3.26. Minh họa ba cách tính tích phân

- Trong ba cách tính tích phân trình bày ở trên, thì cách tính tích phân hình thang cho kết quả chính xác nhất, do đó thực tế người ta thường sử dụng công thức:

$$u_k^I = u_{k-1}^I + T \frac{e_{k-1} + e_k}{2}$$

$$U_I(z) = z^{-1} U(z) + \frac{K_I T}{2} (z^{-1} E(z) + E(z))$$

Hàm truyền:  $G_I(z) = \frac{U_I(z)}{E(z)} = \frac{K_I T}{2} \left( \frac{1+z^{-1}}{1-z^{-1}} \right) = \frac{K_I T}{2} \left( \frac{z+1}{z-1} \right)$

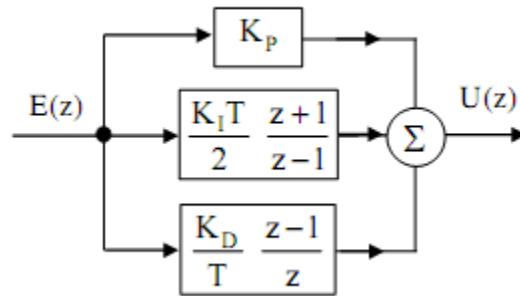
- Từ các hàm truyền cơ bản vừa phân tích ở trên, ta rút ra được hàm truyền của bộ PI, PD, PID số như sau:

$$G_{PI}(z) = K_p + \frac{K_I T}{2} \left( \frac{z+1}{z-1} \right)$$

$$G_{PD}(z) = K_p + \frac{K_D}{T} \left( \frac{z-1}{z} \right)$$

$$G_{PID}(z) = K_p + \frac{K_I T}{2} \left( \frac{z+1}{z-1} \right) + \frac{K_D}{T} \left( \frac{z-1}{z} \right)$$

- Từ đó ta có sơ đồ khối bộ PID số:



Hình 3.27. Sơ đồ khối bộ PID số

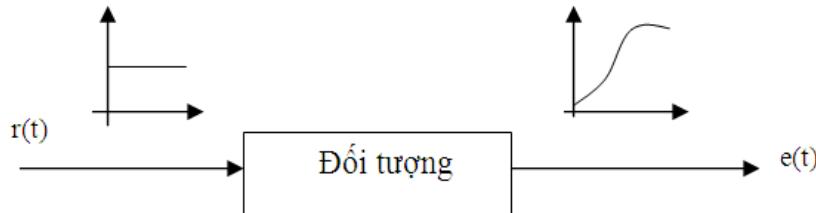
### ✓ Phương pháp hiệu chỉnh PID bằng thực nghiệm

Phương pháp Zeigle – Nichols là phương pháp thực nghiệm để thiết kế bộ điều khiển P, PI hoặc PID bằng cách dựa vào đáp ứng quá độ của đối tượng điều khiển. Bộ điều khiển PID cần thiết kế có hàm truyền là :

$$G_{PID} = K_p + \frac{K_I}{s} + K_D s = K_p \left( 1 + \frac{1}{T_I s} + T_d s \right)$$

Zeigle – Nichols đưa ra 2 phương pháp lựa chọn thông số bộ điều khiển PID tùy đối tượng.

#### • Phương pháp Zeigle – Nichols 1



Phương pháp này sử dụng mô hình xấp xỉ quan tính bậc nhất có trễ của đối tượng điều khiển:  $W_{DT}(s) = \frac{k \cdot e^{-T_1 s}}{1 + T_2 s}$

Phương pháp thực nghiệm này có nhiệm vụ xác định tham số  $k_p$ ,  $T_I$ ,  $T_d$  cho bộ điều khiển PID trên cơ sở xấp xỉ hàm truyền đạt  $S(s)$  của đối tượng thành dạng (1.1), để hệ kín nhanh chóng trở về chế độ xác lập và độ quá điều chỉnh  $\Delta h$  không vượt quá một giới hạn cho phép, khoảng 40% so với  $h_\infty = \lim_{t \rightarrow \infty} h(t)$ , tức là có  $\left| \frac{\Delta h}{h_\infty} \right| \leq 0,4$

Ba tham số  $T_1$  (hằng số thời gian),  $K$  (hệ số khuyếch đại) và  $T_2$  (hằng số thời gian quan tính) của mô hình xấp xỉ quan tính bậc nhất có thể được xác định gần đúng

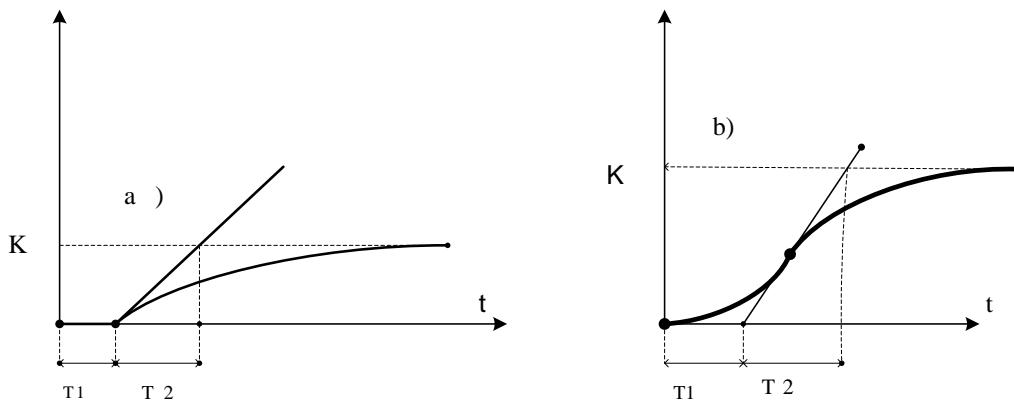
từ đồ thị hàm quá độ  $h(t)$  của đối tượng. Nếu đối tượng có hàm quá độ dạng như hình 5.14 mô tả thì từ đồ thị  $h(t)$  đó ta đọc ra được:

-  $T_1$  là khoảng thời gian tín hiệu ra  $h(t)$  chưa có phản ứng ngay với tín hiệu kích thích  $1(t)$  tại đầu vào.

-  $K$  là giá trị tới hạn  $h_\infty = \lim_{t \rightarrow \infty} h(t)$ .

- Gọi  $A$  là điểm kết thúc khoảng thời gian trễ, tức là điểm trên trực hoành có hoành độ bằng  $T_1$ . Khi đó  $T_2$  là khoảng cần thiết sau  $T_1$  để tiếp tuyến của  $h(t)$  tại đầu vào.

Như vậy điều kiện áp dụng được phương pháp xấp xỉ mô hình bậc nhất có trễ của đối tượng là đối tượng đã phải ổn định, không có dao động và ít nhất hàm quá độ của nó phải có dạng chữ S.



Hình 3.28. Xác định tham số cho mô hình xấp xỉ của đối tượng

Sau khi đã xác định các tham số cho mô hình xấp xỉ của đối tượng, Ziegler-Nichols đã đề nghị sử dụng các tham số  $k_p, T_I, T_D$  cho bộ điều khiển như sau:

$K$  là giá trị giới hạn  $h(\infty)$ .

- Kẻ đường tiếp tuyến của  $h(t)$  tại điểm uốn của nó. Khi đó  $T_1$  sẽ là hoành độ giao điểm của tiếp tuyến với trực hoành và  $T_2$  là khoảng thời gian cần thiết để đường tiếp tuyến đi được từ giá trị 0 tới được giá trị  $K$ .

Như vậy ta thấy điều kiện để áp dụng được phương pháp xấp xỉ mô hình bậc nhất hàm quá độ của nó phải có dạng chữ S. Sau khi đã có tham số cho mô hình xấp xỉ của đối tượng, ta chọn các thông số của bộ điều chỉnh theo bảng:

Bảng 3.1: Bảng chọn thông số PID phương pháp Ziegler-Nichols thứ nhất.

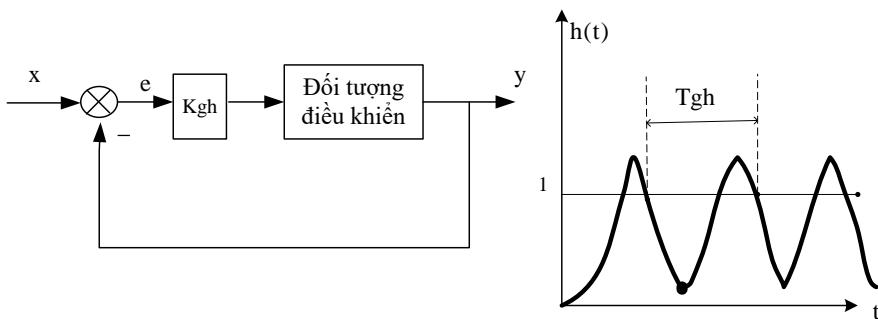
Bộ ĐK/ Tham số	$K_p$	$T_I$	$T_d$
P	$T_2/(T_1.K)$	$\infty$	0

PI	$0.9 T_2/(T_1 \cdot K)$	$10T_1/3$	0
PID	$1,2 T_2/(T_1 \cdot K)$	$2T_1$	$0,5 T_1$

### • Phương pháp Zeigle – Nichols 2

Phương pháp thứ hai này không sử dụng mô hình toán học của đối tượng. Phương pháp thực nghiệm thứ hai này chỉ áp dụng được cho những đối tượng có được chế độ biên giới ổn định khi hiệu chỉnh hằng số khuếch đại trong hệ kín.

Trước tiên, sử dụng bộ P lắp vào hệ kín (hoặc dùng bộ PID và chỉnh các thành phần  $K_I$  và  $K_d$  về giá trị 0). Khởi động quá trình với hệ số khuếch đại  $K_p$  thấp, sau đó tăng dần  $K_p$  tới giá trị tới hạn  $K_{gh}$  để hệ kín ở chế độ giới hạn ổn định, tức là tín hiệu ra  $h(t)$  có dạng dao động điều hòa. Xác định chu kỳ tới hạn  $T_{gh}$  của dao động.



Hình 3.29. Xác định hệ số khuếch đại tới hạn

Thông số bộ điều chỉnh như sau:

Bảng 3.2: Bảng chọn thông số PID phương pháp Ziegler-Nichols thứ hai.

Bộ điều khiển (Thông số)	$K_p$	$K_I$	$T_d$
P	$0.5K_{gh}$	$\infty$	0
PI	$0.45K_{gh}$	$0.93T_{gh}$	0
PID	$0.6K_{gh}$	$0.5T_{gh}$	$0.125T_{gh}$

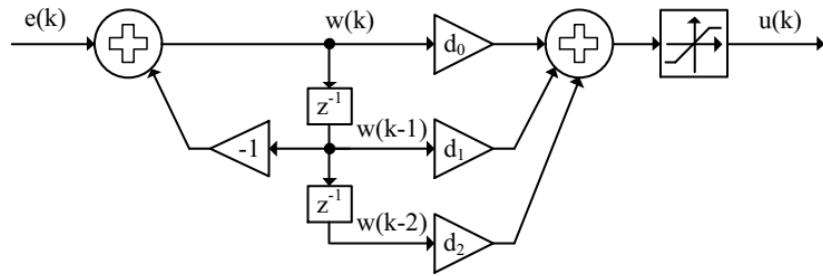
**Ví dụ 3.1:** Viết chương trình cho bộ điều khiển PID số thông thường

Hàm truyền đạt của bộ điều khiển PID thông thường

$$PID(z) = \frac{U(z)}{E(z)} = \frac{d_0 + d_1 z^{-1} + d_2 z^{-2}}{1 - z^{-1}}$$

Phương trình sai phân

$$u(k) = u(k-1) + d_0 e(k) + d_1 e(k-1) + d_2 e(k-2)$$



Hình 3.30. Sơ đồ bộ điều khiển PID số thông thường

```

void PID()
{
    e0=vtdata-vtecd;
    tanso = tanso
    + kp * (e0-e1)
    + ki * (e0 + e1 ) / 2
    + kd * (e0 - (2 * e1) + e2);
    if (tanso>= 70)
    {
        tanso = 70;
    }
    else if (0 <tanso && tanso <= 9)
    {
        tanso = 9;
    }
    else if (tanso < 0)
    {
        tanso= 0;
    }
    e2=e1;
    e1=e0;
}
//-----INTERRUP PROGRAM-----
//-----vector ngat uu tien thap-----

```

## CÂU HỎI ÔN TẬP VÀ THẢO LUẬN

**Câu 1:** Trình bày cấu trúc chương trình khi lập trình vi điều khiển PIC trên MPLAB với trình dịch XC8 hoặc C18?

**Câu 2:** Trình bày các kiểu biến và cách khai báo biến trong ngôn ngữ lập trình C với trình dịch XC8 hoặc C18? Sự giống và khác nhau giữa biến toàn cục và biến cục bộ?

**Câu 3:** Trình bày sự giống và khác nhau giữa chương trình con và chương trình ngắn? Vi điều khiển PIC 18F4431 có bao nhiêu nguồn ngắn và kể tên?

**Câu 4:** Trình bày các cách khai báo hàm với dịch dịch XC8 hoặc C18?

**Câu 5:** Trình bày các cấu trúc lệnh điều kiện và vòng lặp với dịch dịch XC8 hoặc C18?

**Câu 6:** Trình bày những vấn đề kỹ thuật khi thực hiện điều khiển số với VĐK.

**Câu 7:** Trình bày quy trình thực hiện hệ thống điều khiển số

**Câu 8:** Trình bày khái niệm bộ điều khiển PID và phương pháp hiệu chỉnh bộ PID bằng thực nghiệm.

**Câu 9:** Trình bày cấu trúc và phương pháp lập trình bộ điều khiển PID sử dụng vi điều khiển PIC

## BÀI TẬP ÚNG DỤNG

**Câu 1:** Vẽ mạch nguyên lý và viết chương trình điều khiển cho mạch gồm 2 nút nhấn A, B và 8 led đơn ghép theo dạng tích cực âm nối với VĐK PIC18F như sau:

Ban đầu 8 led đơn nhấp nháy.

Ấn A 8 led đơn sáng lần lượt từ trên xuống và lặp đi lặp lại cho đến khi ấn B.

Ấn B 8 led đơn sáng dần từ trên xuống và lặp đi lặp lại cho đến khi ấn A.

**Câu 2:** Vẽ mạch nguyên lý và viết chương trình điều khiển cho mạch gồm 2 nút ấn A, B và 8 led đơn ghép theo dạng tích cực dương nối với VĐK PIC18F như sau:

Ban đầu 8 led đơn nhấp nháy.

Ấn A 8 led đơn sáng lần lượt từ trên xuống và lặp đi lặp lại cho đến khi ấn B.

Ấn B 8 led đơn sáng dần từ trên xuống và lặp đi lặp lại cho đến khi ấn A.

**Câu 3:** Vẽ mạch nguyên lý và viết chương trình điều khiển cho mạch gồm 2 nút ấn A, B và 16 led đơn nối với 8051 như sau:

Ban đầu 16 led đơn nhấp nháy.

Ấn A 16 led đơn sáng lần lượt từ trên xuống và lặp đi lặp lại cho đến khi ấn B.

Ấn B 16 led đơn sáng dần từ trên xuống và lặp đi lặp lại cho đến khi ấn A.

**Câu 4:** Vẽ mạch nguyên lý và viết chương trình điều khiển cho mạch gồm 2 nút ấn A, B ghép tích cực âm và 2 led 7 đoạn loại anot chung ghép nối với VĐK PIC18F như sau:

Ban đầu 2 led 7 đoạn hiển thị 00.

Ấn A 2 led 7 đoạn hiển thị từ 00 đến 99 và lặp đi lặp lại cho đến khi ấn B.

Ấn B 2 led 7 đoạn hiển thị từ 99 về 00 và lặp đi lặp lại cho đến khi ấn A.

**Câu 5:** Vẽ mạch nguyên lý và viết chương trình điều khiển cho mạch gồm 2 nút ấn A, B ghép tích cực dương và 2 led 7 đoạn loại cathode chung ghép nối với VĐK PIC18F như sau:

Ban đầu 2 led 7 đoạn nhấp nháy số 00.

Ấn A 2 led 7 đoạn hiển thị từ 00 đến 60 và lặp đi lặp lại cho đến khi ấn B.

Ấn B 2 led 7 đoạn hiển thị từ 60 về 00 và lặp đi lặp lại cho đến khi ấn A.

**Câu 6:** Vẽ mạch nguyên lý và viết chương trình điều khiển cho mạch gồm 2 nút ấn A, B ghép tích cực âm, 1 led 7 đoạn anot chung và 8 led đơn ghép tích cực âm nối với VĐK PIC18F như sau:

Ban đầu led 7 đoạn hiển thị 0, 8 led đơn nhấp nháy liên tục.

Ấn A led 7 đoạn hiển thị số 1, 8 led đơn sáng lần lượt từ trên xuống rồi quay về như ban đầu.

Ấn B led 7 đoạn hiển thị số 2, 8 led đơn sáng dần từ trên xuống rồi quay về như ban đầu.

Nếu các led đơn đang sáng lần lượt mà ấn B thì các led sẽ chuyển sang sáng dần và ngược lại.

**Câu 7:** Vẽ mạch nguyên lý và viết chương trình cho mạch gồm LCD 16x2 ghép nối vi điều khiển VĐK PIC18F như sau:

Dòng 1 hiển thị tên lớp.

Dòng 2 hiển thị tên sinh viên.

**Câu 8:** Thiết kế chương trình điều khiển cho mạch đo lường dòng điện sử dụng điện trở Shunt 30A/75mV, hiện thị trên LCD và bật đèn cảnh báo nhấp nháy 1Hz khi dòng điện trên 10A.

**Câu 9:** Thiết kế chương trình điều khiển cho mạch đo lường điện áp 300VDC sử dụng mạch khuếch đại vi sai, hiện thị trên LCD và bật đèn cảnh báo nhấp nháy 1Hz khi điện áp quá 250V và dưới 100V.

## CHƯƠNG 4

### THIẾT KẾ MẠCH VÀ MÔ PHỎNG VI ĐIỀU KHIỂN PIC 18F

#### MỤC TIÊU CỦA CHƯƠNG

Cung cấp cho sinh viên các kiến thức về phần mềm thiết mạch nguyên lý cho PIC 18F và phần mềm mô phỏng chương trình ứng dụng.

#### 4.1. CÁC PHẦN MỀM THIẾT KẾ MẠCH

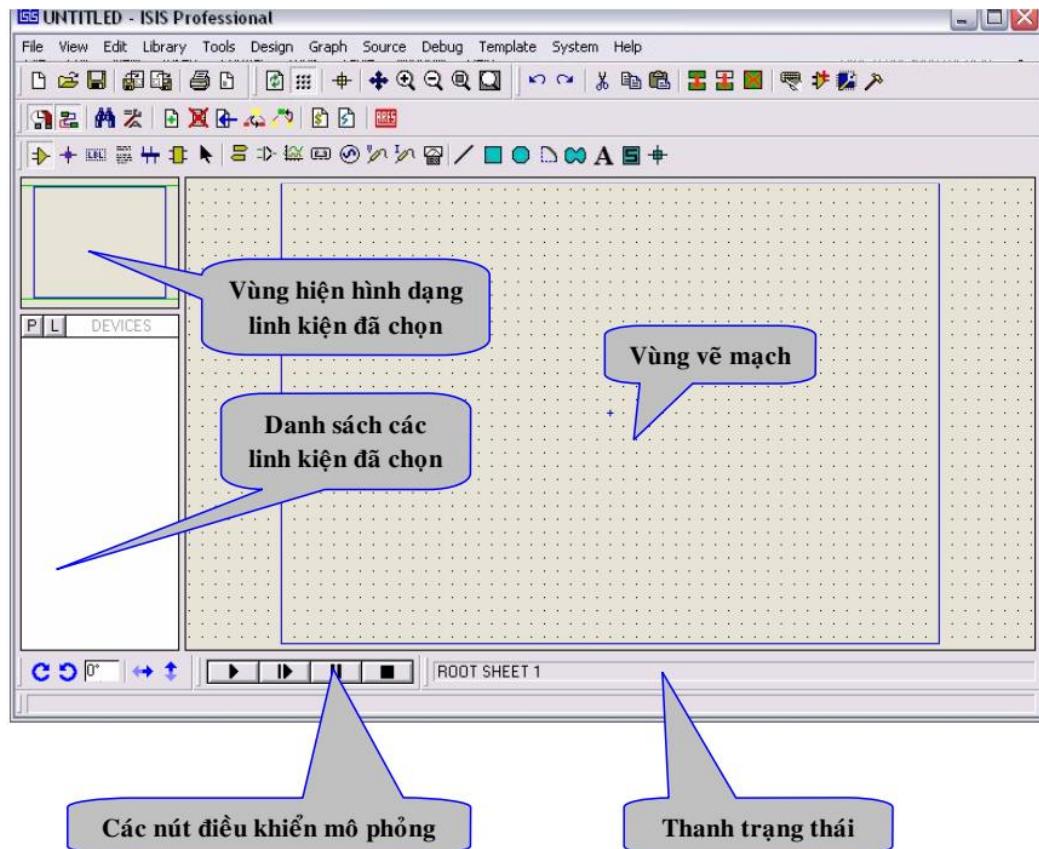
##### 4.1.1. Phần mềm Proteus

Proteus là phần mềm của hãng Labcenter dùng để vẽ sơ đồ nguyên lý, mô phỏng và thiết kế mạch điện. Gói phần mềm gồm có phần mềm chính :

- ISIS dùng để vẽ sơ đồ nguyên lý và mô phỏng
- ARES dùng để thiết kế mạch in.

###### a. Vẽ sơ đồ nguyên lý với ISIS

Chương trình được khởi động và có giao diện như hình 4.1



Hình 4.1. Giao diện vẽ sơ đồ nguyên lý ISIS

Phía trên và phía phải của chương trình là các công cụ để ta có thể thiết kế sơ đồ nguyên lý. Phần giữa có màu xám là nơi để chúng ta vẽ mạch.

Bảng 4.1. Một số ký hiệu cơ bản

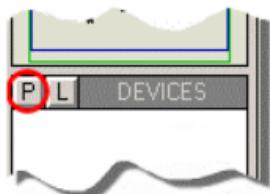
STT	Ký hiệu	Ý nghĩa
-----	---------	---------

1		Section mode: Chức năng này để chọn linh kiện
2		Component mode: Dùng để lấy linh kiện trong thư viện linh kiện
3		Đặt label cho wire
4		Bus
5		Terminal: Chứa Power, Ground
6		Graph: Dùng để vẽ dạng sóng, datasheet, trở kháng
7		Generator Mode: Chứa các nguồn điện, nguồn xung, nguồn dòng
8		Voltage Probe Mode: Dùng để đo điện thế tại 1 điểm trên mạch, đây là 1 dụng cụ chỉ có 1 chân và không có thát trong thức tế
9		Current Probe mode: Dùng để đo chiều và độ lớn của dòng điện tại 1 điểm trên wire
10		Virtual Instrument Mode: Chứa các dụng cụ đo dòng và áp, các dụng cụ này được mô phỏng như trong thực tế

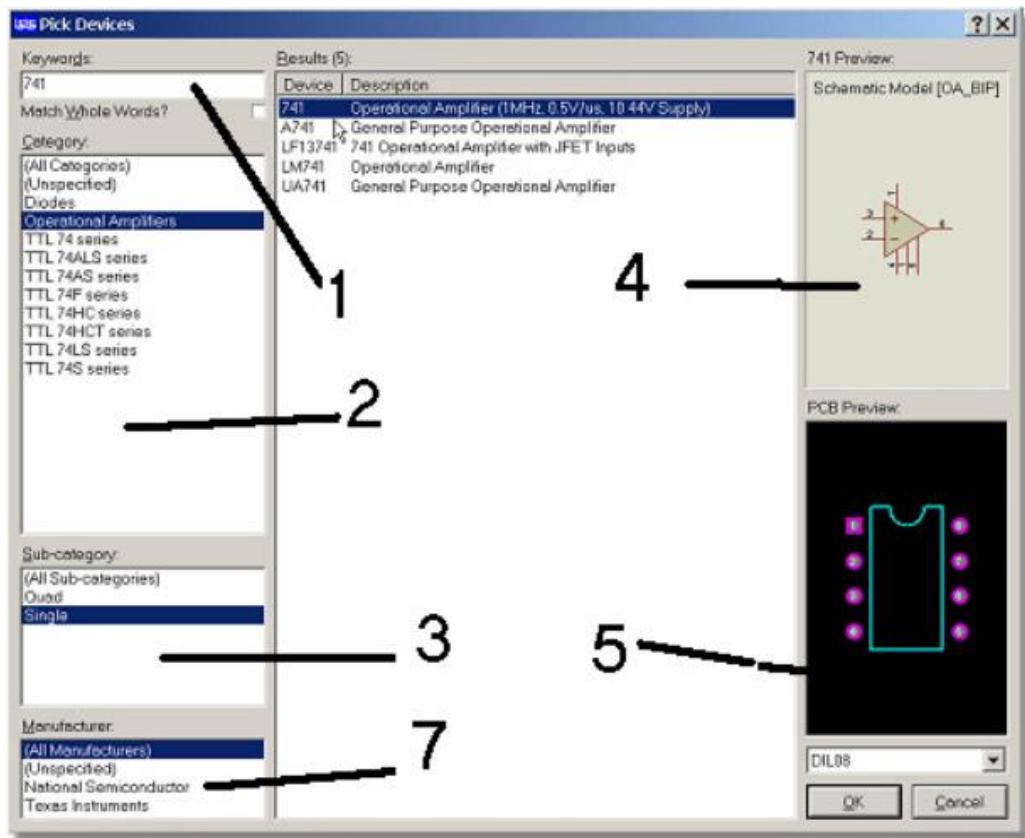
### b. Cách lấy linh kiện

Để lấy linh kiện, nhìn vào phía trái của chương trình và thực hiện như sau:

- Bấm vào biểu tượng Component Mode
- Sau đó bấm vào chữ P hoặc ấn phím tắt P trên bàn phím

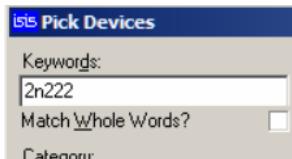


- Khung chương trình Pick Devices hiện ra như hình 4.1 :



Hình 4.2. Khung chương trình Pick Devices

- là ô tìm kiếm linh kiện, chỉ cần gõ từ khóa vào, ví dụ như muốn tìm BJT 2N2222 thì gõ 2N2222 như hình vẽ (không phân biệt chữ hoa và chữ thường).



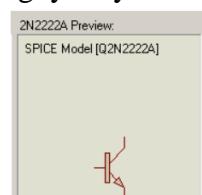
- Là các nhóm linh kiện liên quan đến từ khóa cần tìm.



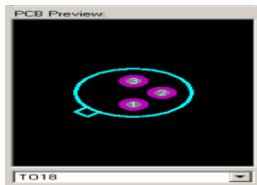
- Là nhóm con của linh kiện, ví dụ như transistor thì có BJT, FET



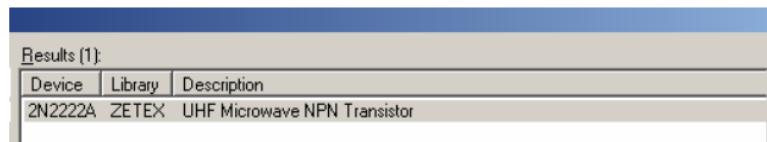
- Là ký hiệu (Schematic) trên sơ đồ nguyên lý



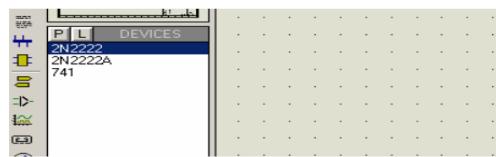
5. Là hình dáng trên sơ đồ mạch in (**PCB**), ví dụ như BJT có nhiều kiểu đóng gói như TO18, TO220, vv ...



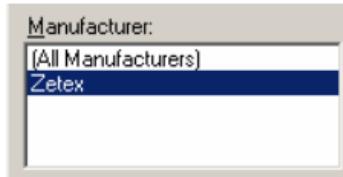
6. Là kết quả của việc tìm kiếm linh kiện.



Double Click vào linh kiện cần lấy, lập tức linh kiện sẽ được bổ sung vào “bàn làm việc” là vùng màu trắng phí bên trái . Xem hình dưới



7. Là tên nhà sản xuất

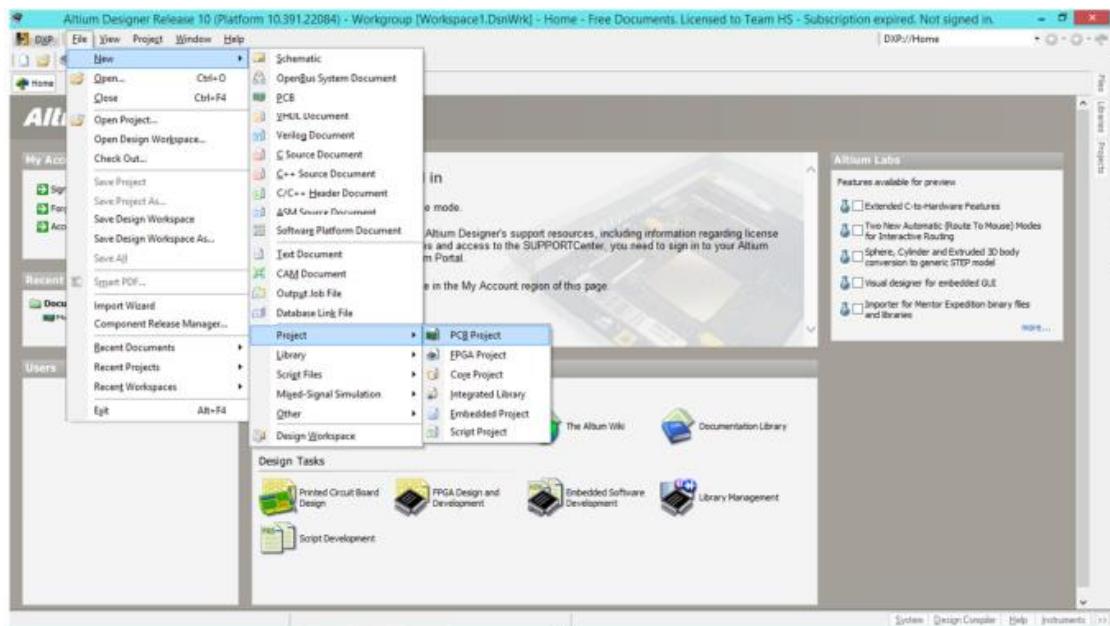


#### 4.1.2. Phần mềm Altium

Hiện nay, để thiết kế một mạch điện tử cơ bản thì có rất nhiều các phần mềm đáp ứng được yêu cầu đó: Altium Designer, OrCAD, Proteus, Eagle, Sprint. Đặc điểm chung của các phần mềm này là tích hợp sẵn các thư viện mẫu có sẵn về các linh kiện cơ bản mà người dùng có thể lấy ra và thiết kế 1 cách đơn giản.

Phần mềm Altium Designed là một trong những phần mềm đó, nó có thể giúp người dùng thiết kế một mạch in một cách nhanh chóng nhất. Phần mềm Altium Designer với giao diện đơn giản với người dùng, hỗ trợ vẽ mạch nguyên lý, mạch in (PCB), hỗ trợ xuất file in, xuất các file cho máy CNC. Ngoài ra phần mềm còn hỗ trợ người dùng tạo linh kiện một cách nhanh nhất bằng cách lựa chọn các kiểu chân có sẵn.

Để bắt đầu thiết kế mạch điện tử, đầu tiên phải tạo một project : File > New > Project > PCB Project



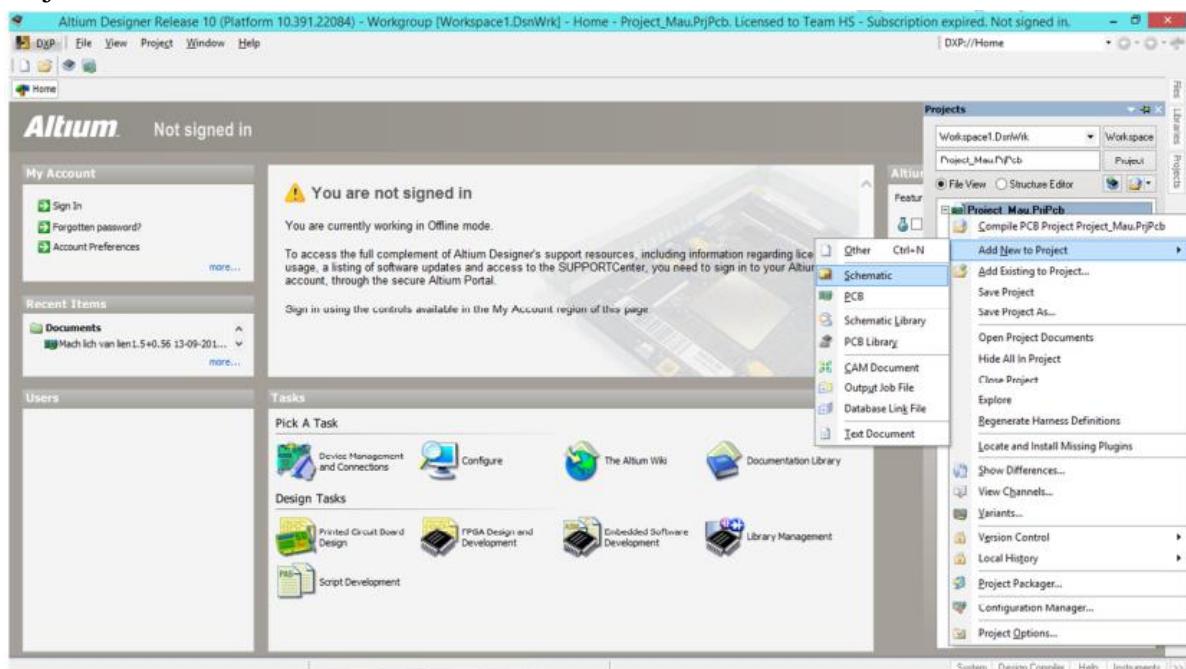
Sau khi đã xuất hiện như hình 3. Tiến hành lưu Project đó lại bằng cách:

Nhấn chuột phải vào Project > Save Project.

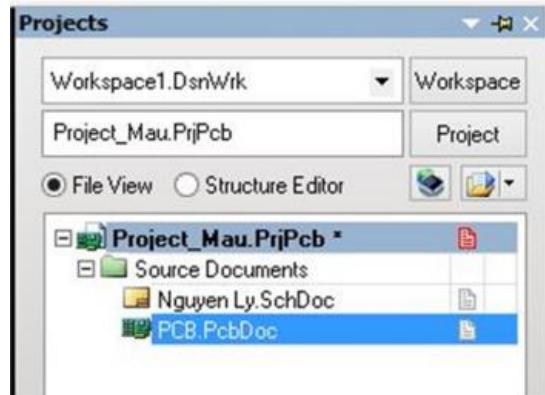
Bước tiếp theo là tạo file nguyên lý và tạo file PCB:

Nhấn chuột phải vào Project vừa tạo > Add New to Project > Schematic và PCB.

Sau khi đã tạo xong file nguyên lý và mạch in thì lưu lại giống như lưu Project. Chú ý file schematic và PCB phải được làm trong cùng thư mục với file Project.



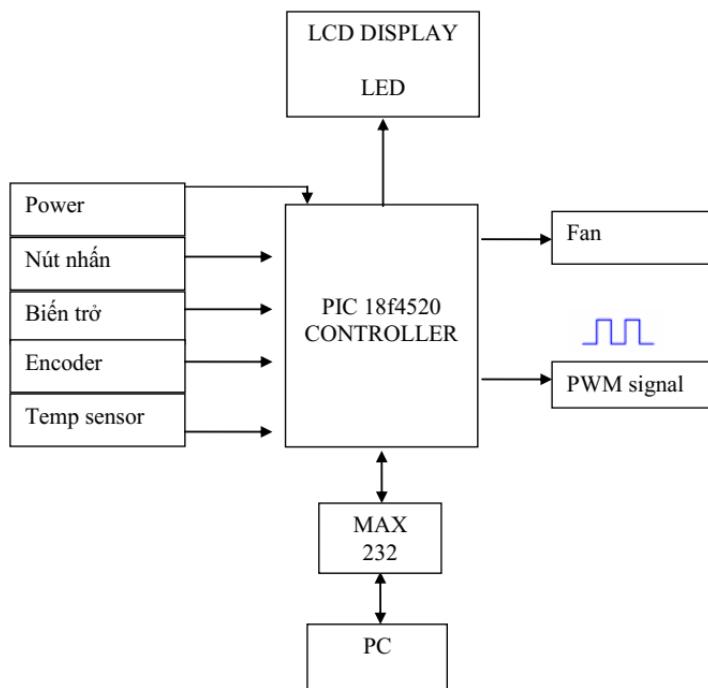
Toàn bộ Project sau khi tạo sẽ được thể hiện ở mục trên hình 4.3



Hình 4.3. Toàn bộ Project

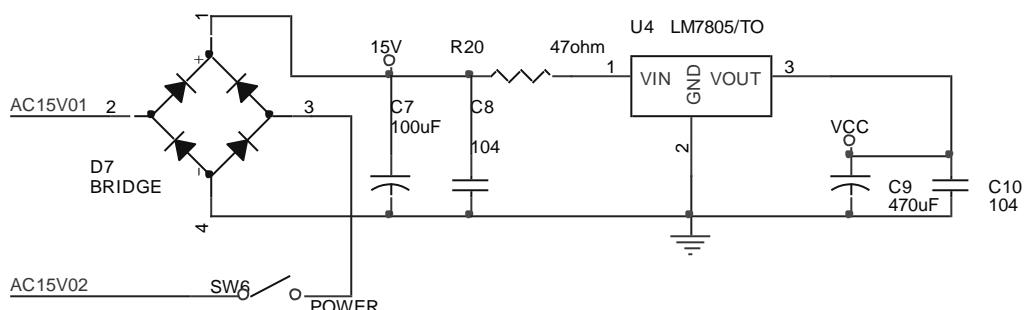
#### 4.2. THIẾT KẾ MẠCH CHO VI ĐIỀU KHIỂN PIC 18F

Sơ đồ khái niệm của khối điều khiển:



Hình 4.4. Sơ đồ khái niệm mạch điều khiển, hiển thị.

- **Khối nguồn:**



Điện áp AC 15V được chỉnh lưu theo phương pháp chỉnh lưu toàn kì không điều khiển dùng cầu diode, điện áp DC sau đó được lọc phẳng bằng tụ và đưa vào vi mạch ổn áp 7805 để tạo ra điện áp 5V ổn định cấp cho mạch điều khiển.

Do yêu cầu về an toàn và chống nhiễu, điện áp DC cung cấp cho mạch điều khiển, mạch công suất, mạch lái đều được dùng các nguồn khác nhau.

#### - Nút nhấn và biến trở.

Mạch điều khiển được thiết kế với 4 nút nhấn và 4 biến trở.

Nút nhấn dùng để chọn chế độ điều khiển, chọn chiều quay động cơ được dùng chủ yếu trong mode dùng PID trên PIC.

Tất cả các nút nhấn đều được thiết kế tích cực mức thấp.

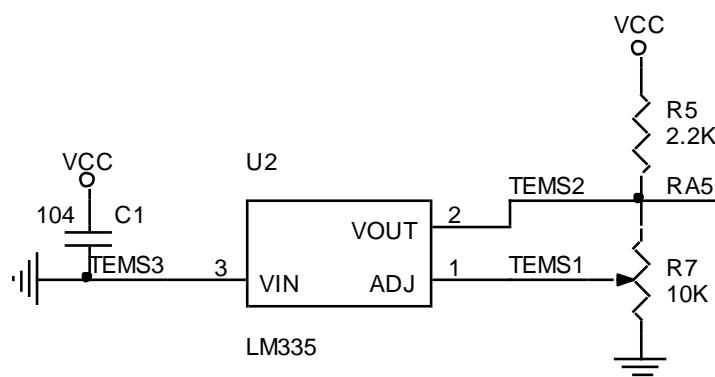
Biến trở dùng để cài đặt tốc độ cũng như cài đặt các giá trị của các thông số PID.

#### - Encoder.

Sử dụng Encoder quang có độ phân giải 100 xung/vòng, dùng để đo và hồi tiếp tốc độ quay của động cơ. Các xung được vi xử lý đếm vào khi có tác động cạnh xuống trên chân vi điều khiển.

#### - Cảm biến nhiệt độ:

Cảm biến dùng để đo nhiệt độ, giúp lấy mẫu nhiệt độ của các khóa công suất. Cảm biến nhiệt có rất nhiều loại, ở đây sử dụng IC cảm biến nhiệt LM335 với tầm đo trong khoảng  $-40^{\circ}C - 100^{\circ}C$ .



#### - LCD hiển thị.

Sử dụng màn hình tinh thể lỏng LCD 20x4, gồm có 4 hàng và mỗi hàng đều hiển thị được 20 ký tự. LCD được giao tiếp với vi xử lý ở chế độ 4 bit giúp tiết kiệm chân port cho vi xử lý.

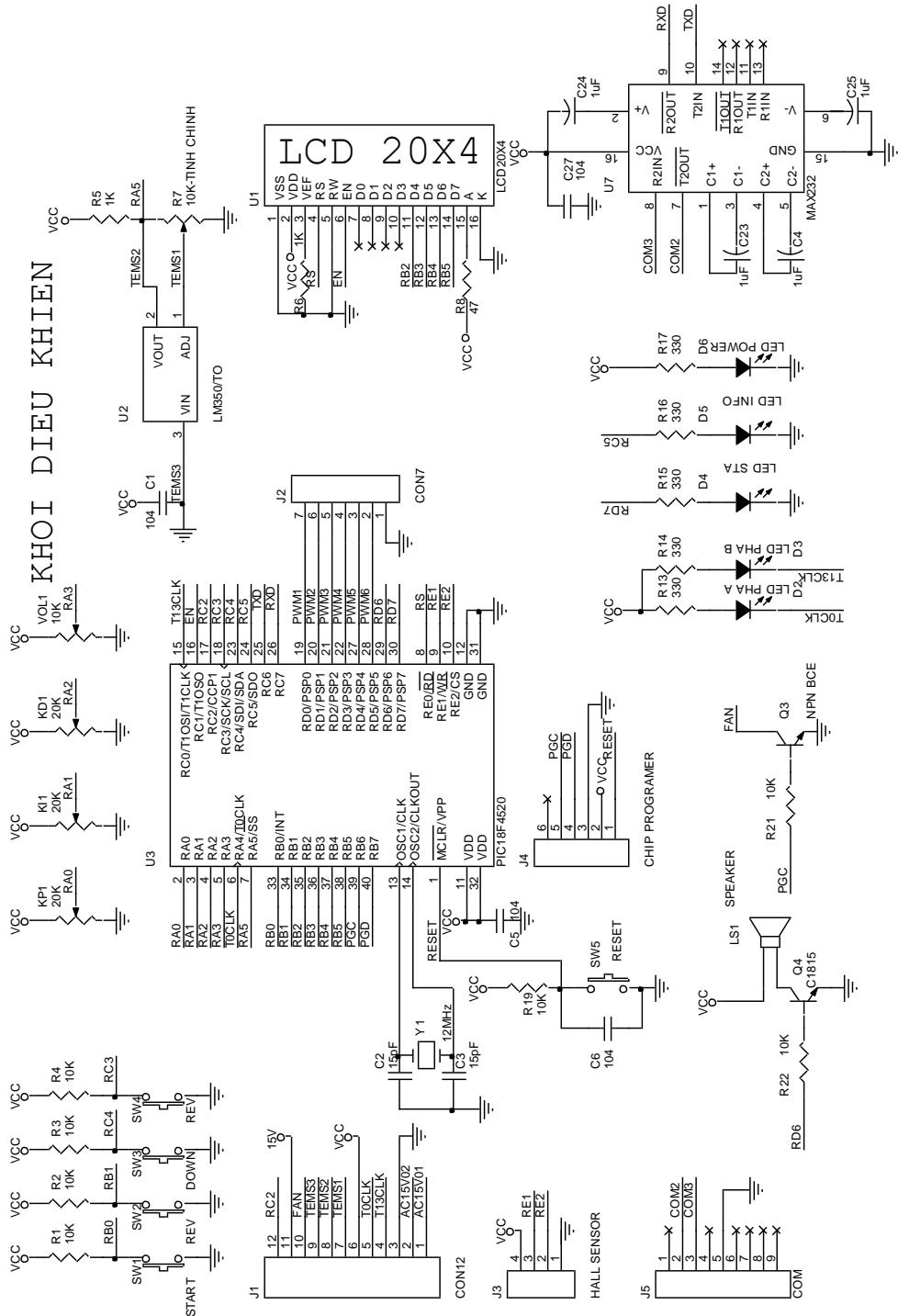
Việc dùng LCD để hiển thị có nhiều ưu điểm là:

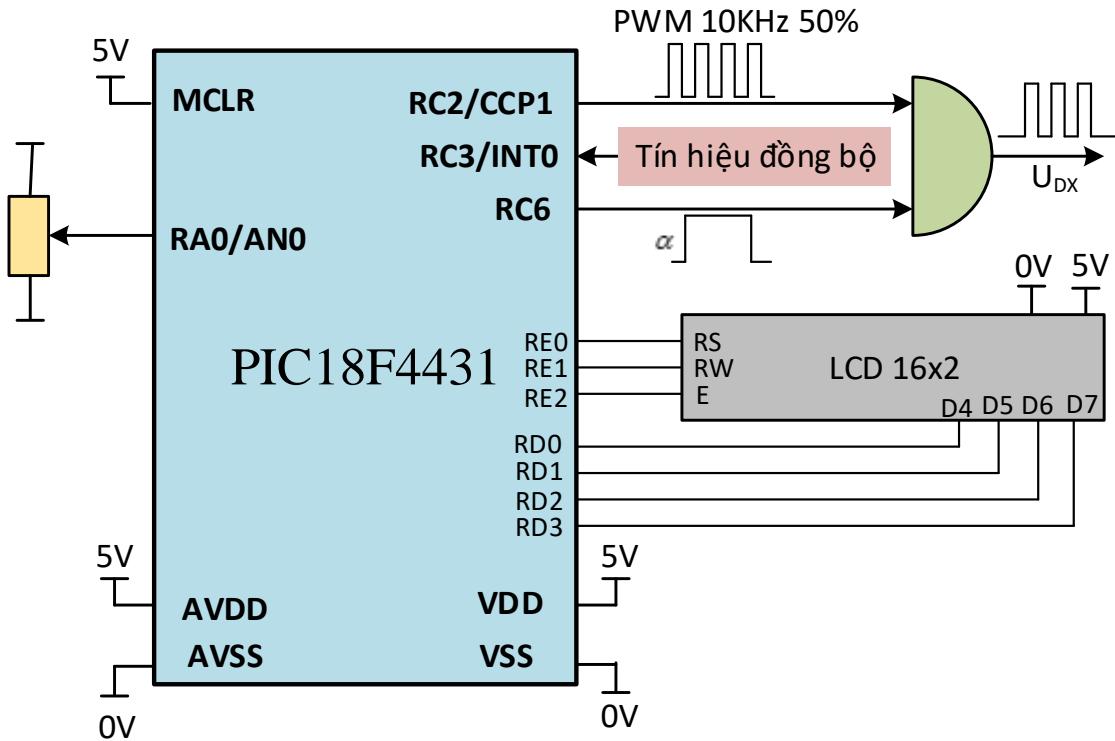
- Dễ lập trình .
- Giao diện bắt mắt, trực quan, thông tin hiển thị rõ ràng.

- LCD 20x4 có số lượng kí tự đủ lớn để hiển thị các thông tin cần thiết.
- **Khối Max232.**

Khi ghép nối cổng Com của máy tính với vi điều khiển hay mạch TTL cần phải có mạch chuyển mức  $TTL \rightarrow 232$  và ngược lại. Quy định về mức tín hiệu của hai chuẩn này không giống nhau do đó cần có mạch chuyển đổi để hai thiết bị làm việc theo hai chuẩn trên có thể giao tiếp với nhau.

Vì mạch chuyển đổi thường được sử dụng là MAX232(maxim) hoặc DS275(dallas). MAX232 thông dụng hơn cả vì chỉ cần nguồn 5V, mạch  $\pm 10V$  do mạch dao động 16KHz bên trong cung cấp.





Hình 4.6. Sơ đồ kết nối vi điều khiển PIC18F3341 khởi hiển thị, khởi tạo xung, khởi dạng xung

**a, Khởi hiển thị:** dùng LCD16x2 hiển thị góc điều khiển  $\alpha$ , các thông số điện áp, dòng điện, ...

Chương trình lập trình:

```

LCD_Init();
LCD_Clear();
LCD_Gotoxy(1,0);
LCD_Write_String(" CHINH LUU ");
LCD_Gotoxy(2,0);
LCD_Write_String("GOC ANPHA= ");
LCD_Gotoxy(2,10);
LCD_Write_String(text);

```

**b, Khởi tạo xung:** Tín hiệu đồng bộ đưa vào vi điều khiển PIC18F4431 qua chân RC3/INT0, chức năng ngắt ngoài tạo xung qua chân RC6.

Chương trình lập trình:

```

//Chuong trinh ngat ngoai INT0
void interrupt Ngat_INT0(void)
{

```

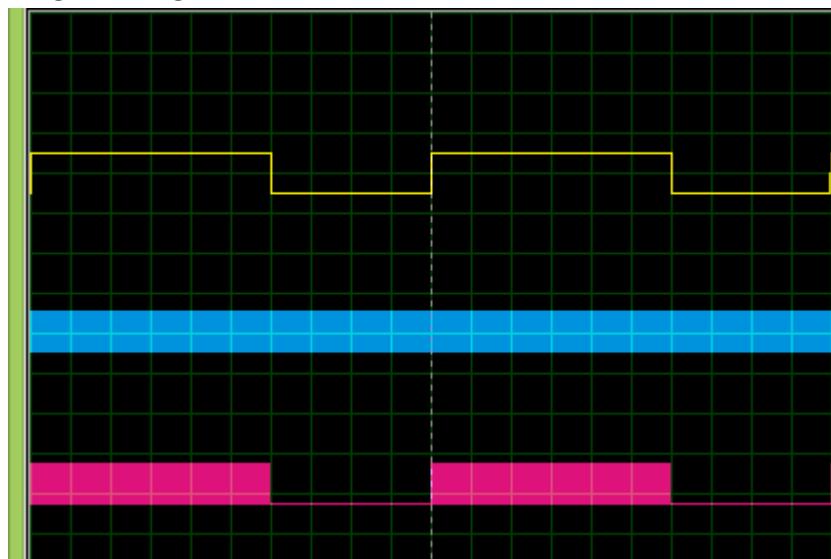
```

if(INT0IF)
{
    delay_ms(timer);
    LATCbits.LATC6=1;
    delay_ms(7-timer);
    LATCbits.LATC6=0;
    INT0IF=0;
}

```

### c, Khối dạng xung:

- **Dạng xung chùm (XC)**: là dạng thông dụng nhất, vì cho phép mở tốt van lực trong mọi trường hợp, với mọi dạng tải và nhiều sơ đồ chỉnh lưu khác nhau. Xung chùm thực chất là một chùm các xung có tần số cao gấp nhiều lần lưới điện ( $f_{xc} = 6 \div 12 \text{ kHz}$ ). Để dàng nhận thấy rằng để thực hiện tạo XC theo nguyên tắc thứ nhất chỉ cần một mạch logic và logic AND.



Hình 4.7. Mô phỏng khung tạo xung chùm

- **Tạo xung dao động tần số cao**: Sử dụng module CCP của vi điều khiển PIC18F4431 qua chân RC2/CCP1 tạo xung PWM tần số 10KHz:

Chương trình lập trình:

```

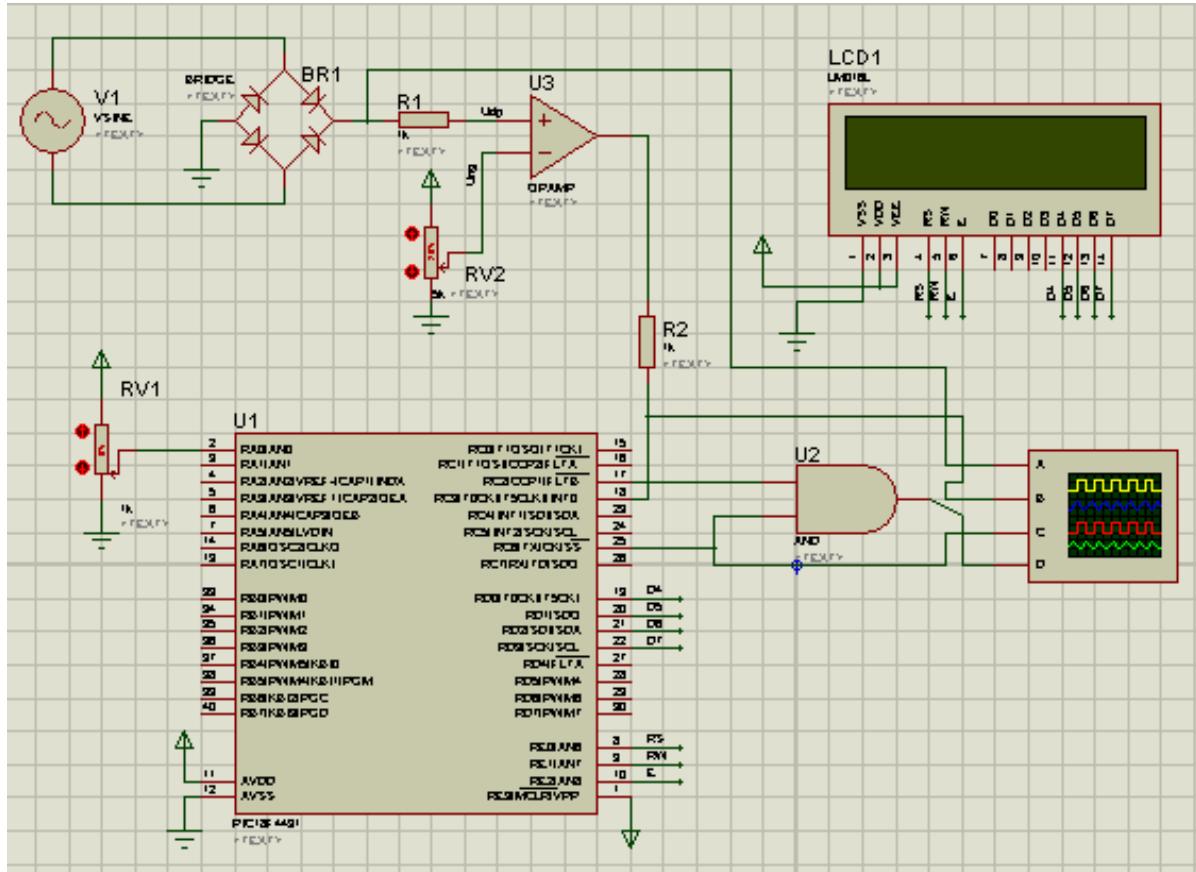
void Config_PWM()
{
    TRISCbits.RC2=0; //Chon chan CCP1 co chieu ra
    CCP1CON|=0b00001100; // Chon che do PWM
}

```

```

T2CONbits.T2CKPS1=0; //TMR2 Prescaler Value =4
T2CONbits.T2CKPS0=1;
T2CONbits.TMR2ON=1; //Bat Timer 2
PR2=124; // PWM Period = 100uS (F=10Khz)
CCPR1L=250/4; //Dua 8bit cao chua gia tri bo dem vao CCPR1L
CCP1CON|=(250%4)<<4; // Dua 2 bit thap vao CCP1CON<5:4>
}

```



Hình 4.8. Mô phỏng mạch điều khiển bộ biến đổi điện tử công suất phụ thuộc lưới điện

## CÂU HỎI ÔN TẬP VÀ THẢO LUẬN

**Câu 1:** Trình bày cách tạo một project bằng phần mềm Proteus. Cho ví dụ

**Câu 2:** Trình bày cách tạo một project bằng phần mềm Altium. Cho ví dụ

**Câu 3:** Thiết kế và mô phỏng mạch điều khiển sử dụng vi điều khiển PIC 18F.

## BÀI TẬP ÚNG DỤNG

**Câu 1:** Vẽ mạch nguyên lý và mô phỏng chương trình điều khiển cho mạch đo lường dòng điện sử dụng điện trở Shunt 30A/75mV, hiện thị trên LCD và bật đèn cảnh báo nháy 1Hz khi dòng điện trên 10A.

**Câu 2:** Vẽ mạch nguyên lý và mô phỏng chương trình điều khiển cho mạch đo lường điện áp 300VDC sử dụng mạch khuếch đại vi sai, hiện thị trên LCD và bật đèn cảnh báo nháy 1Hz khi điện áp quá 250V và dưới 100V.

**Câu 3:** Viết chương trình và mô phỏng bộ điều khiển PID cho bộ điều khiển tốc độ động cơ điện một chiều. Quá trình điều khiển tốc độ được thực hiện qua biến trờ và hiển thị giá trị góc mở  $\alpha$  trên LCD 16x2.

**Câu 4:** Viết chương trình và mô phỏng bộ điều khiển PID cho bộ biến đổi băm xung áp một chiều điều khiển tốc độ động cơ điện một chiều. Quá trình điều khiển tốc độ được thực hiện qua biến trờ và hiển thị giá trị độ rộng xung (%) trên LCD 16x2.

**Câu 5:** Viết chương trình và mô phỏng bộ điều khiển PID cho bộ biến đổi băm xung áp một chiều có đảo chiều quay điều khiển tốc độ động cơ điện một chiều. Quá trình điều khiển tốc độ được thực hiện qua biến trờ và hiển thị giá trị độ rộng xung (%) trên LCD 16x2.

## **TÀI LIỆU THAM KHẢO**

1. Kiều Xuân Thực, Vũ Thị Thu Hương, Vũ Trung Kiên (2008), “*Vi điều khiển Cấu trúc – lập trình - ứng dụng*”, Nhà xuất bản giáo dục.
2. Nguyễn Trường Thịnh, Nguyễn Tân Nó. “*Vi điều khiển PIC16F và ngôn ngữ lập trình Hi-Tech C*”. Nhà xuất bản Đại học Quốc gia TP. HCM-2014.
3. Vũ Trung Kiên, Phạm Văn Chiến, Nguyễn Văn Tùng. “*Giáo trình Kỹ thuật ứng dụng vi điều khiển vào điều khiển máy*”. JJCA, Đại học Công nghiệp Hà Nội – 2016.
4. Microchip. “*PIC18F2331/2431/4331/4431 Data Sheet*”. 2010 Microchip Technology Inc.