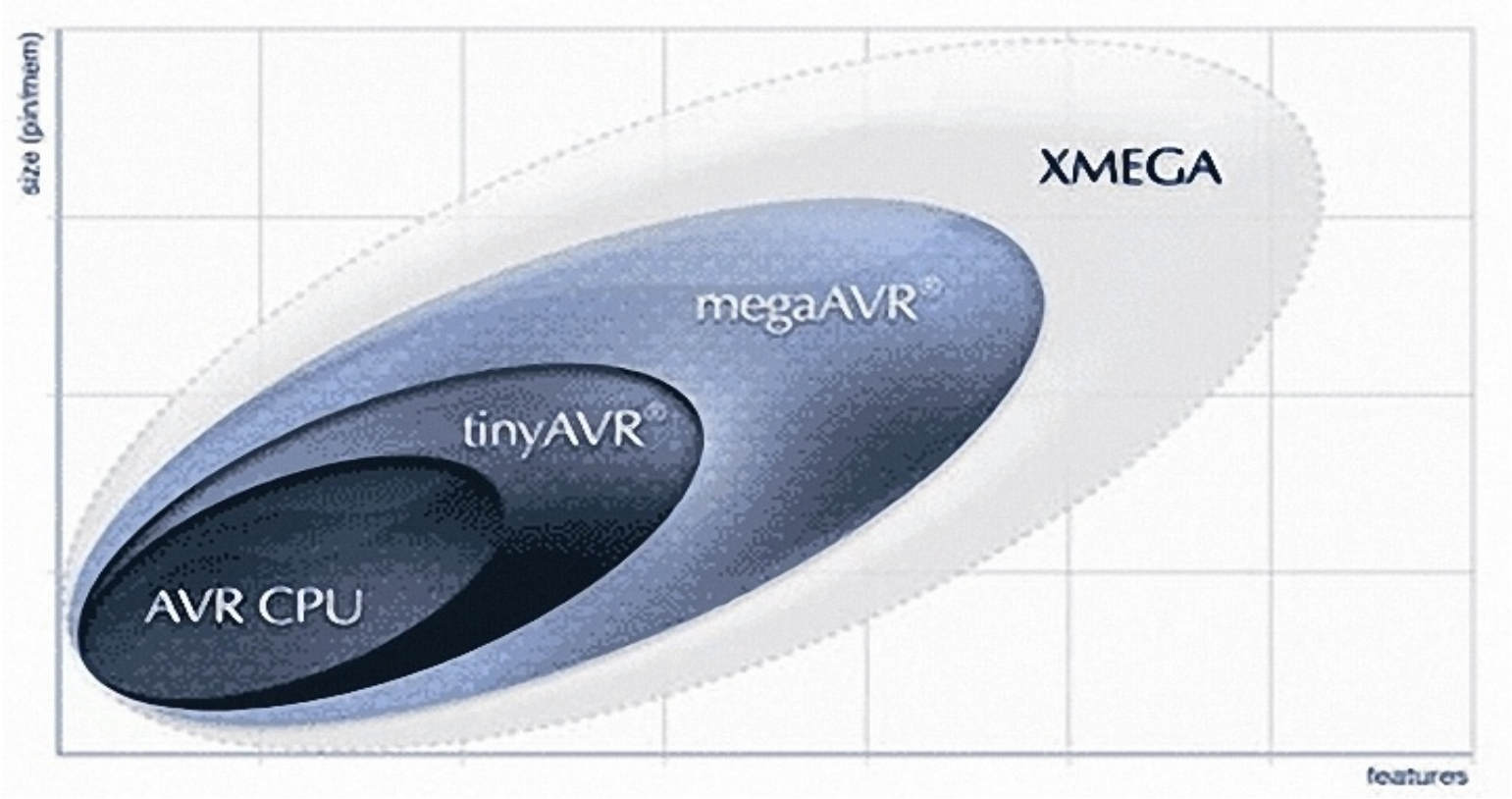
Seminar

Vi Điều Khiển AVR

ATmega128



Lê Trung Thắng

"Con người nhờ ước mơ mà vĩ đại"

…. Tài liệu này trước đây là seminar của tôi về vi điều khiển AVR vào khoảng cuối năm 2007, lúc đầu cũng chỉ ghi chép như một cuốn sổ tay để ghi nhớ, đến lúc xong cái seminar thì thấy con AVR này cũng rất thú vị, nên tôi đã chỉnh sửa lại bản ghi chép để soạn thành tài liệu này. Có lẻ là do quen với họ 8051 do Atmel sản xuất, nên khi chuyển sang AVR sẽ cảm thấy quen thuộc hơn.

Mục đích chính mà tôi viết tài liệu này là để chia sẻ với các bạn có cùng sở thích về AVR, qua đó chúng ta có thể tạo ra một cộng đồng AVR-Friends thật đông đảo và sôi nổi. Một cộng đồng AVR đông đảo là rất có ích cho chính tôi và cho các bạn, vì như thế chúng ta sẽ có nhiều cơ hội để trao đổi và học hỏi nhau hơn.

Tài liệu này tôi cũng muốn gởi tặng em trai Lê Trung Thông, hy vọng em có thể bổ sung cho anh những phần còn thiếu của tài liệu này.

Toàn bộ tài liệu này chủ yếu được dịch ra từ datasheet của con Atmega128, nhưng do không có nhiều thời gian nên tài liệu còn thiếu rất nhiều phần, nên tôi hi vọng các bạn nào có kinh nghiệm về AVR sẽ tiếp tục bổ sung, chỉnh sửa để chúng ta có một tài liệu hoàn chỉnh hơn, nếu cần, tôi có thể gởi file word cho các bạn để tiện lợi cho việc bổ sung, chỉnh sửa (*mail to: thangvl2a@yahoo.com*).

Sài Gòn, 08-2008.
Lê Trung Thắng.
ĐTVT - K2002.

Mục Lục:

Chương I

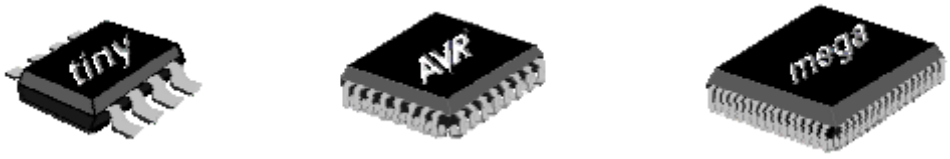
TỔNG QUAN

Những Tính Năng Chính Của ATmega128:

- ROM : 128 Kbytes
- SRAM: 4Kbytes
- EEPROM : 4Kbytes
- 64 thanh ghi I/O
- 160 thanh ghi vào ra mở rộng
- 32 thanh ghi đa mục đích.
- 2 bộ định thời 8 bit (0,2).
- 2 bộ định thời 16 bit (1,3).
- Bộ định thời watchdog
- Bộ dao động nội RC tần số 1 MHz, 2 MHz, 4 MHz, 8 MHz
- ADC 8 kênh với độ phân giải 10 bit (Ở dòng Xmega lên tới 12 bit)
- 2 kênh PWM 8 bit
- 6 kênh PWM có thể lập trình thay đổi độ phân giải từ 2 tới 16 bit
- Bộ so sánh tương tự có thể lựa chọn ngõ vào
- Hai khối USART lập trình được
- Khối truyền nhận nối tiếp SPI
- Khối giao tiếp nối tiếp 2 dây TWI
- Hỗ trợ boot loader
- 6 chế độ tiết kiệm năng lượng
- Lựa chọn tần số hoạt động bằng phần mềm
- Đóng gói 64 chân kiểu TQFP.
- Tần số tối đa 16MHz
- Điện thế : 4.5v - 5.5v

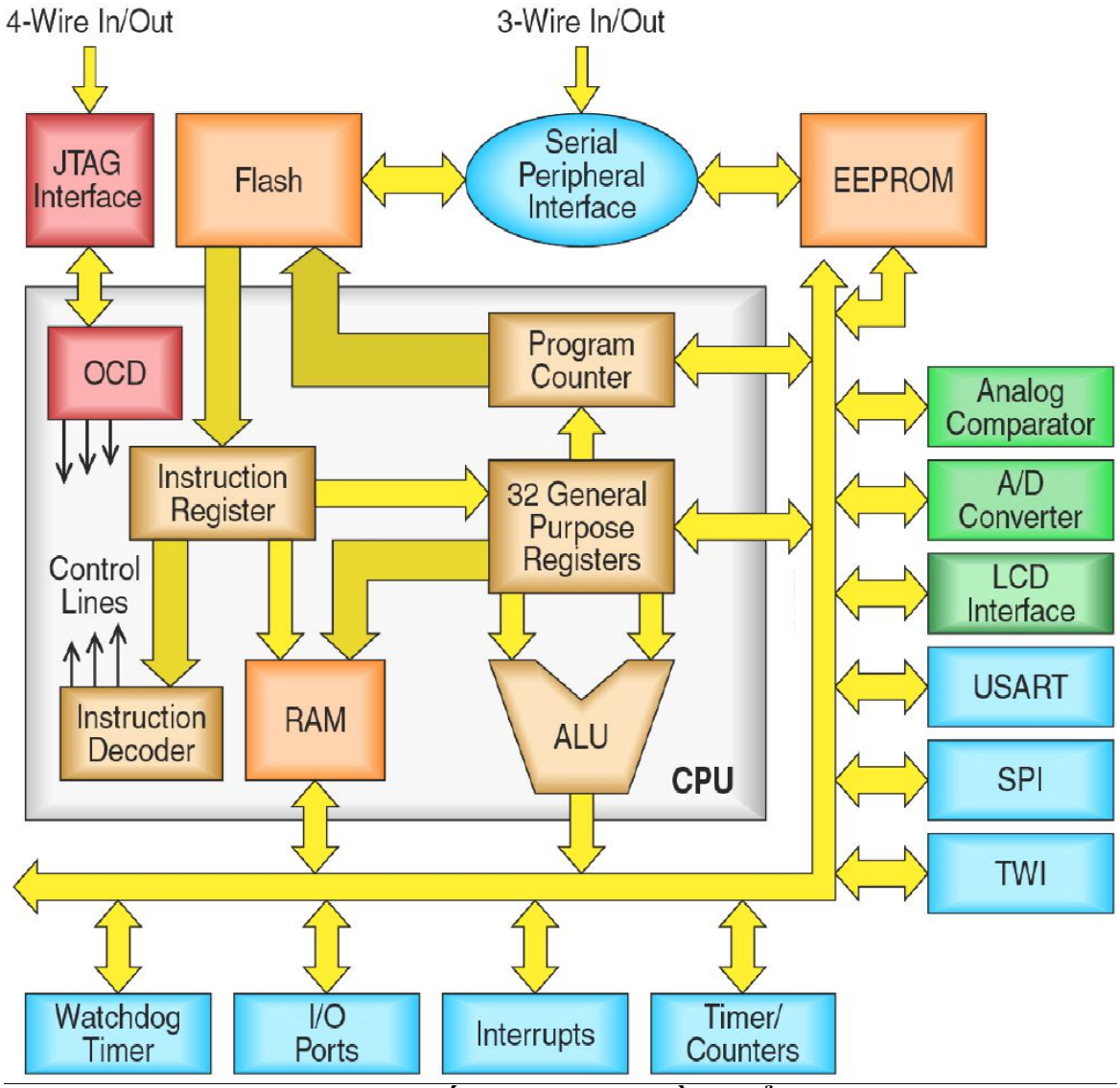
…v.v…

Vi điều khiển AVR do hãng Atmel (Hoa Kì) sản xuất được gới thiệu lần đầu năm 1996. AVR có rất nhiều dòng khác nhau bao gồm dòng Tiny AVR (như AT tiny 13, AT tiny 22…) có kích thước bộ nhớ nhỏ, ít bộ phận ngoại vi, rồi đến dòng AVR (chẳn hạn AT90S8535, AT90S8515,…) có kích thước bộ nhớ vào loại trung bình và mạnh hơn là dòng Mega (như ATmega32, ATmega128,…) với bộ nhớ có kích thước vài Kbyte đến vài trăm Kb cùng với các bộ ngoại vi đa dạng được tích hợp trên chip, cũng có dòng tích hợp cả bộ LCD trên chip (dòng LCD AVR). Tốc độ của dòng Mega cũng cao hơn so với các dòng khác. Sự khác nhau cơ bản giữa các dòng chình là cấu trúc ngoại vi, còn nhân thì vẫn như nhau, **Hình 1.1**. *Đặt biệt, năm 2008, Atmel lại tiếp tục cho ra đời dòng AVR mới là XmegaAVR, với những tính năng mạnh mẽ chưa từng có ở các dòng AVR trước đó. Có thể nói XmegaAVR là dòng MCU 8 bit mạnh mẽ nhất hiện nay.*



Hình1.1 Các dòng AVR khác nhau: Tiny, AVR và Mega

Cấu trúc cơ bản của vi điều khiển AVR được thể hiện ở **hình 1.2**.



Hình 1.2. Cấu trúc của Vi điều khiển AVR

Chương II

CẤU TRÚC BỘ NHỚ VÀ CỔNG VÀO - RA

I. CẤU TRÚC BỘ NHỚ

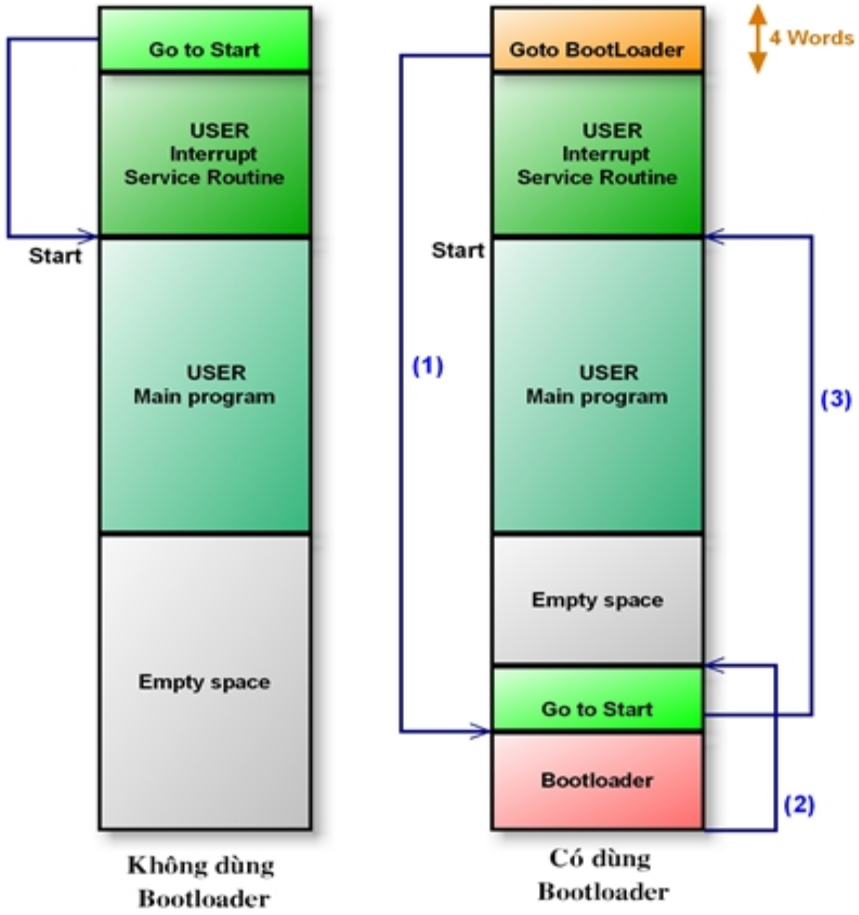
Giới Thiệu:

Bộ nhớ vi điều khiển AVR có cấu trúc Harvard là cấu trúc có đường Bus riêng cho bộ nhớ chương trình và bộ nhớ dữ liệu. Bộ nhớ AVR được chia làm 2 phần chính: Bộ nhớ chương trình (program memory) và bộ nhớ dữ liệu (Data memory).

- **Bộ Nhớ Chương Trình :** Bộ nhớ chương trình của AVR là bộ nhớ Flash có dung lượng 128 K bytes. Bộ nhớ chương trình có độ rộng bus là 16 bit. Những địa chỉ đầu tiên của bộ nhớ chương trình được dùng cho bảng véc tơ ngắt (xem chi tiết về bảng véc tơ ngắt ở chương 4). Cần để ý là ở vi điều khiển ATmega128 bộ nhớ chương trình còn có thể được chia làm 2 phần : phần boot loader (Boot loader program section) và phần ứng dụng (Application program section).

  Phần boot loader chứa chương trình boot loader. Chương trình Boot loader là một phần mềm nhỏ nạp trong vi điều khiển và được chạy lúc khởi động. Phần mềm này có thể tải vào trong vi điều khiển chương trình của người sử dụng và sau đó thực thi chương trình này. Mỗi khi reset vi điều khiển CPU sẽ nhảy tới thực thi chương trình boot loader trước, chương trình boot loader sẽ dò xem có chương trình nào cần nạp vào vi điều khiển hay không, nếu có chương trình cần nạp, boot loader sẽ nạp chương trình vào vùng nhớ ứng dụng (Application program section), rồi thực thi chương trình này. Ngược lại, boot loader sẽ chuyển tới chương trình ứng dụng có sẵn trong vùng nhớ ứng dụng để thực thi chương trình này.

  Phần ứng dụng (Application program section) là vùng nhớ chứa chương trình ứng dụng của người dùng. Kích thước của phần boot loader và phần ứng dụng có thể tùy chọn. **Hình 2.1** thể hiện cấu trúc bộ nhớ chương trình có sử dụng và không sử dụng boot loader, khi sử dụng phần boot loader ta thấy 4 word đầu tiên thay vì chỉ thị cho CPU chuyển tới chương trình ứng dụng của người dùng (là chương trình có nhãn start) thì chỉ thị CPU nhảy tới phần chương trình boot loader để thực hiện trước, rồi mới quay trở lại thực hiện chương trình ứng dụng.



Hình 2.1 Bộ nhớ chương trình có và không có sử dụng boot loader

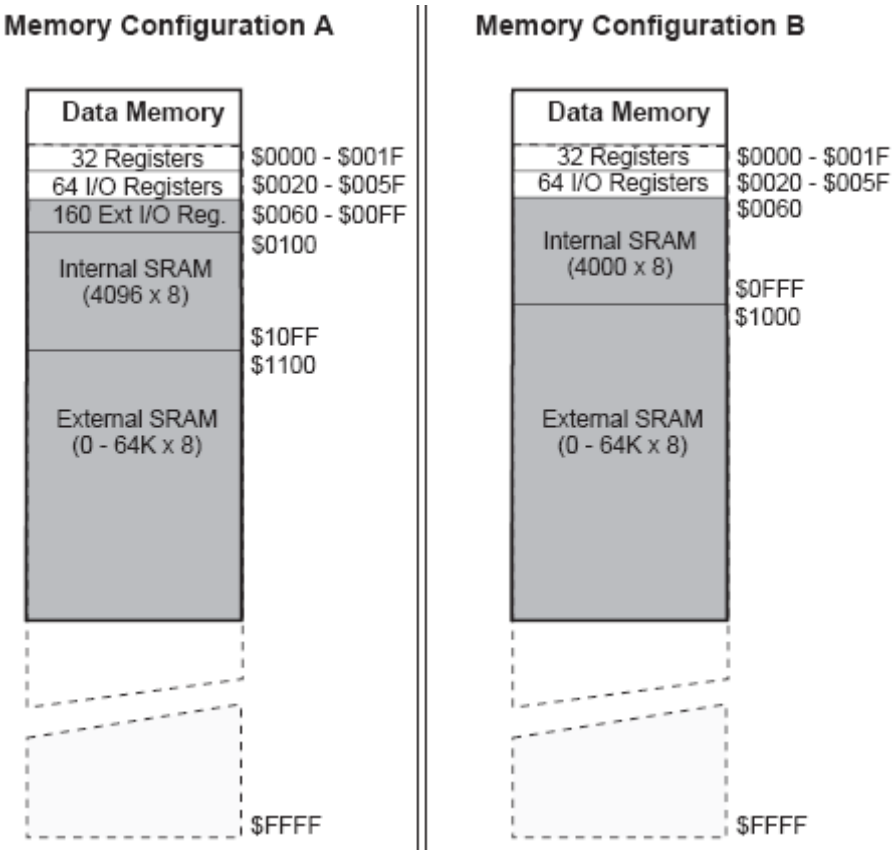
❖ **Bộ Nhớ Dữ Liệu :** Bộ nhớ dữ liệu của AVR chia làm 2 phần chính là bộ nhớ SRAM và bộ nhớ EEPROM. Tuy cùng là bộ nhớ dữ liệu nhưng hai bộ nhớ này lại tách biệt nhau và được đánh địa chỉ riêng.

● **Bộ nhớ SRAM** có dụng lượng 4 K bytes, Bộ nhớ SRAM có hai chế độ hoạt động là chế độ thông thường và chế độ tương thích với ATmega103, muốn thiết lập bộ nhớ SRAM hoạt động theo chế độ nào ta sử dụng bit cầu chì M103C (M103C fuse bit [9]).

Bộ nhớ SRAM ở chế độ bình thường : Ở chế độ bình thường bộ nhớ SRAM được chia thành 5 phần: Phần đầu là 32 thanh ghi chức năng chung (General Purpose Register) R0 đến R31 có địa chỉ từ $0000 tới $001F. Phần thứ 2 là không gian nhớ vào ra với 64 thanh ghi vào ra (I/O Register) có địa chỉ từ $0020 tới $005F. Phần thứ 3 dùng cho vùng nhớ dành cho các thanh ghi vào ra mở rộng (Extended I/O Registers) có địa chỉ từ $0060 tới $00FF. Phần thứ 4 là vùng SRAM nội với 4096 byte có địa chỉ từ $0100 tới $10FF. Phần thứ 5 là vùng nhớ SRAM ngoài (External SRAM) bắt đầu từ địa chỉ $1100, vùng SRAM mở rộng này có thể mở rộng lên đến 64 K byte. Khi nói bộ nhớ SRAM có dung

lượng 4 K byte là nói tới phần thứ 4 (SRAM nội). Nếu tính cả các thanh ghi thì bộ nhớ SRAM trong chế độ bình thường sẽ là 4.25 K byte = 4352 byte.

Bộ nhớ SRAM ở chế độ tương thích ATmega103 : Ở chế này bộ nhớ SRAM cơ bản cũng giống ở chế độ bình thường, ngoại trừ phần thứ 3 là vùng nhớ dành cho các thanh ghi vào ra mở rộng không tồn tại, ngoài ra kích thước của phần SRAM nội (internal SRAM) chỉ có 4000 byte so với 4096 byte ở chế độ bình thường. Hình 2.2 thể hiện sơ đồ bộ nhờ dữ liệu ở cả hai chế độ : Bình thường và tương thích ATmega103. Từ **hình 2.2** ta thấy nếu cấu hình để bộ nhớ SRAM hoạt động ở chế độ tương thích ATmega103 thì ta sẽ bị mất đi 160 thanh ghi vào ra mở rộng (extended I/O Register), là những thanh ghi đóng vai trò quan trọng trong các chế độ hoạt động của vi điều khiển.

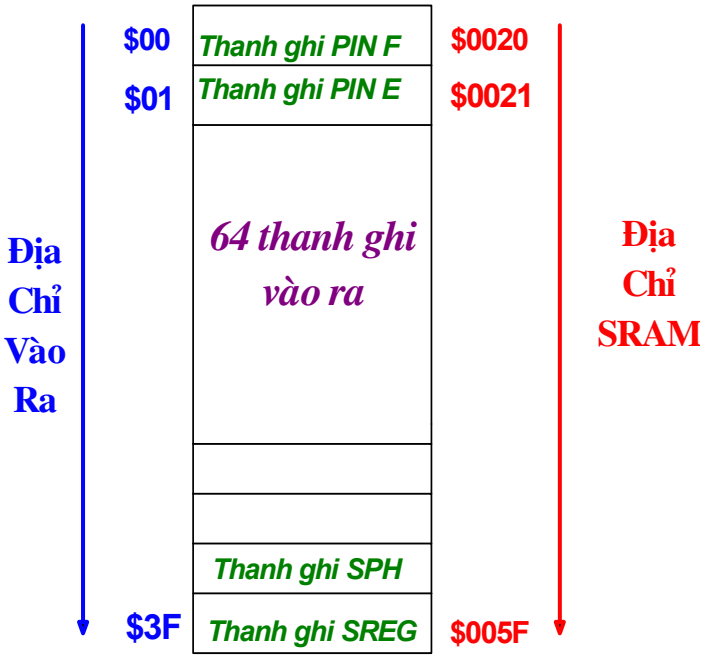


Hình 2.2 Bản đồ bộ nhớ dữ liệu
A : *Chế độ bình thường*
B: *Chế độ tương thích ATmega103*

Trong vùng nhớ vào ra mở rộng ($0060 - $00FF) chỉ có 6 lệnh sau là có thể được sử dụng, là : ST / STS / STD và LD / LDS / LDD.

Lệnh CBI và SBI chỉ có thể làm việc với 32 thanh ghi thấp hơn trong vùng nhớ vào ra , tức các thanh ghi I/O có địa chỉ từ $20 tới $3F (địa chỉ SRAM).

64 thanh ghi vào ra trong vùng nhớ vào ra (phần số 2) có 2 kiểu chọn địa chỉ : Nếu xem chúng là vùng nhớ vào ra thì địa chỉ sẽ là $00 - $3F, khi sử dụng các lệnh in, out … ta phải sử dụng địa chỉ này. Nếu xem chúng như là một phần của bộ nhớ SRAM thì sẽ có địa chỉ là $0020 - $005F, khi ta dùng các lệnh như LD, ST… ta phải sử dụng kiểu địa chỉ này. (**hình 2.3**). *Trong tài liệu này các địa chỉ được sử dụng sẽ được hiểu như là địa chỉ SRAM nếu không có giải thích gì thêm.* Để ý là 160 thanh ghi vào ra mở rộng ($0060 - $00FF) không có 2 kiểu chọn địa chỉ như trên, địa chỉ của chúng chính là các địa chỉ SRAM .

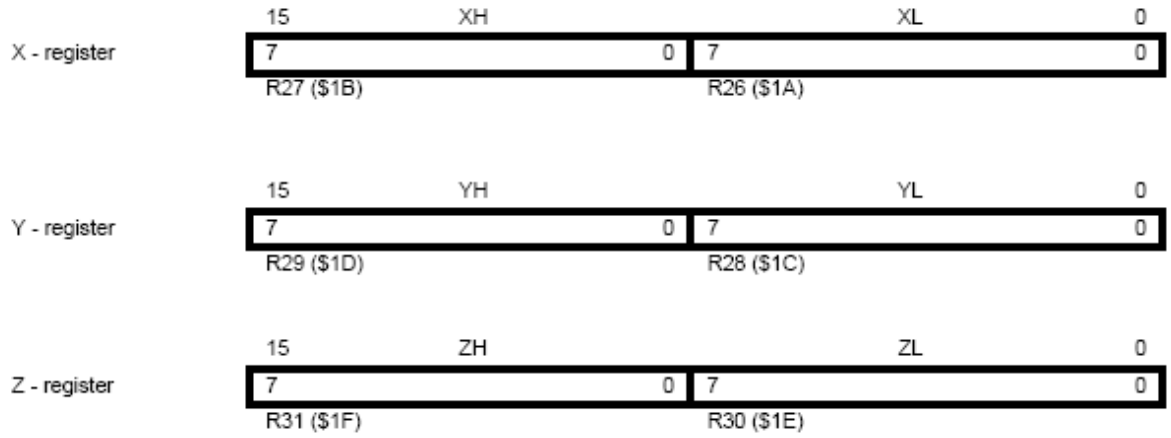


Hình 2.3 Vùng nhớ 64 thanh ghi vào ra có 2 cách chọn địa chỉ

Chi tiết về 64 thanh ghi vào ra và 160 thanh ghi vào ra mở rộng có thể tìm thấy ở datasheet của vi điều khiển ATmega128.

Tiệp ghanh ghi (register file) : Tiệp 32 thanh ghi đa chức năng ($0000 - $001F) đã được nói ở trên, ngoài chức năng là các thanh ghi đa chức năng, thì các thanh ghi từ R26 tới R31 từng đôi một tạo thành các thanh ghi 16 bit X, Y, Z được dùng làm con trỏ trỏ tới bộ nhớ chương trình và bộ nhớ dữ liệu (**Hình 2.4**). Thanh ghi con trỏ X, Y có thể dùng làm con trỏ trỏ tới bộ nhớ dữ liệu, còn thanh ghi Z có thể dùng làm con trỏ trỏ tới bộ nhớ chương trình. Các trình biên dịch C thường dùng các thanh ghi con trỏ này để quản lí **Data stack** của chương trình C.

Figure 5. The X-, Y-, and Z-registers



Hình 2.4. Chức năng con trỏ của các thanh ghi R26 –R31

● **Bộ nhớ EEPROM :** Đây là bộ nhớ dữ liệu có thể ghi xóa ngay trong lúc vi điều khiển đang hoạt động và không bị mất dữ liệu khi nguồn điện cung cấp bị cắt. Có thể ví bộ nhớ dữ liệu EEPROM giống như là ổ cứng (Hard disk) của máy vi tính. Với vi điều khiển ATmega128, bộ nhớ EEPROM có kích thước là 4 Kbyte. EEPROM được xem như là một bộ nhớ vào ra được đánh địa chỉ độc lập với SRAM, điều này có nghĩa là ta cần sử dụng các lệnh in, out … khi muốn truy xuất tới EEPROM. Để điều khiển vào ra dữ liệu với EEPROM ta sử dụng 3 thanh ghi sau :

1. Thanh Ghi EEAR (EEARH và EEARL)

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	EEAR11	EEAR10	EEAR9	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	X	X	X	
	X	X	X	X	X	X	X	X	

EEAR là thanh ghi 16 bit lưu giữ địa chỉ của các ô nhớ của EEPROM, thanh ghi EEAR được kết hợp từ 2 thanh ghi 8 bit là EEARH và thanh ghi EEARL. Vì bộ nhớ

EEPROM của ATmega128 có dung lượng 4 Kbyte = 4096 byte = 2^{12} byte nên ta chỉ cần 12 bit của thanh ghi EEAR , 4 bit từ 15 -12 được dự trữ, ta nên ghi 0 vào các bit dự trữ này.

2. Thanh Ghi EEDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Đây là thanh ghi dữ liệu của EEPROM, là nơi chứa dữ liệu ta định ghi vào hay lấy ra từ EEPROM.

3. Thanh Ghi EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

Đây là thanh ghi điều khiển EEPROM, ta chỉ sử dụng 4 bit đầu của thanh ghi này, 4 bit cuối là dự trữ, ta nên ghi 0 vào các bit dự trữ. Sau đây ta xét chức năng của từng bit.

• **Bit 3 – EERIE: EEPROM Ready Interrupt Enable :** Đây là bit cho phép EEPROM ngắt CPU, khi bit này được set thành 1 và ngắt toàn cục được cho phép (bằng cách set bit I trong thanh ghi SREG lên 1) thì EEPROM sẽ tạo ra một ngắt với CPU khi bit EEWE được xóa, điều này có nghĩa là khi các ngắt được cho phép (bit I trong thanh ghi SREG và bit EERIE trong thanh ghi EECR được set thành 1) và quá trình ghi vào ROM vừa xong thì sẽ tạo ra một ngắt với CPU, chương trình sẽ nhảy tới véc tơ ngắt có địa chỉ là $002C để thực thi chương trình phục vụ ngắt (ISR). Khi bit EERIE là 0 thì ngắt không được cho phép.

• **Bit 2 – EEMWE: EEPROM Master Write Enable :** Khi bit EEMWE và bit EEWE là 1 sẽ ra lệnh cho CPU ghi dữ liệu từ thanh ghi EEDR vào EEPROM, địa chỉ của ô nhớ cần ghi trong EEPROM được lưu trong thanh ghi EEAR . Khi bit này là 0 thì không cho phép ghi vào EEPROM. Bit EEMWE sẽ được xóa bởi phần cứng sau 4 chu kì máy.

• **Bit 1 – EEWE: EEPROM Write Enable :** Bit này vừa đóng vai trò như một bit cờ, vừa là bit điều khiển việc ghi dữ liệu vào EEPROM. Ở vai trò của một bit điều khiển nếu bit EEMWE đã được set lên 1 thì khi ta set bit EEWE lên 1 sẽ bắt đầu quá trình ghi dữ

liệu vào EEPROM. Trong suốt quá trình ghi dữ liệu vào EEPROM bit EEWE luôn giữ là 1. Ở vai trò của một bit cờ khi quá trình ghi dữ liệu vào EEPROM hoàn tất, phần cứng sẽ tự động xóa bit này về 0. Trước khi ghi dữ liệu vào EEPROM ta cần phải biết chắc là không có quá trình ghi EEPROM nào khác đang xảy ra, để biết được điều này ta cần kiểm tra bit EEWE. Nếu bit EEWE là 1 tức là EEPROM đang được ghi, ta phải chờ cho cho quá trình ghi vào EEPROM hoàn tất thì mới ghi tiếp. Nếu bit EEWE là 0 tức là không có quá trình ghi EEPROM nào đang diễn ra, lúc này ta có thể bắt đầu ghi dữ liệu vào EEPROM. Khi bit EEWE được set lên 1 (bắt đầu ghi vào EEPROM) CPU sẽ tạm nghỉ trong 2 chu kì máy trước khi thực hiện lệnh kế tiếp.

• **Bit 0 – EERE: EEPROM Read Enable** : Khi bit này là 1, sẽ cho phép đọc dữ liệu từ EEPROM, dữ liệu từ EEPROM có địa chỉ lưu trong thanh ghi EEAR lập tức được chuyển vào thanh ghi EEDR. Khi bit EERE là 0 thì không cho phép đọc EEPROM. Trước khi đọc dữ liệu từ EEPROM ta cần biết chắc là không diễn ra quá trình ghi EEPROM bằng cách kiểm tra bit EEWE. Để ý là sau khi quá trình đọc EEPROM hoàn tất, bit EERE sẽ được tự động xoá bởi phần cứng. Nếu EEPROM đang được ghi thì ta không thể đọc được dữ liệu từ EEPROM. Khi bắt đầu quá trình đọc dữ liệu từ EEPROM, CPU sẽ tạm nghỉ 4 chu kì máy trước khi thực hiện lệnh kế tiếp.

Tóm lại để ghi vào EEPROM ta cần thực hiện các bước sau:

1. Chờ cho bit EEWE về 0.
2. Cấm tất cả các ngắt.
3. Ghi địa chỉ vào thanh ghi EEAR.
4. Ghi dữ liệu mà ta cần ghi vào EEPROM vào thanh ghi EEDR.
5. Set bit EEMWE thành 1.
6. Set bit EEWE thành 1 .
7. Cho phép các ngắt trở lại.

Nếu một ngắt xảy ra giữa bước 5 và 6 sẽ làm hỏng quá trình ghi vào EEPROM bởi vì bit EEMWE sau khi set lên 1 chỉ được giữ trong 4 chu kì máy, chương trình ngắt sẽ làm hết thời gian (time out) duy trì bit này ở mức 1.

Một ngắt xuất hiện ở cuối bước 4 cũng có thể làm cho địa chỉ và dữ liệu cần ghi vào EEPROM trở nên không chính xác nếu trong chương trình phục vụ ngắt có chỉnh sửa lại các thanh ghi EEAR và EEDR. Đó là lí do ta cần cấm các ngắt trước khi thực hiện tiếp các bước 3, 4, 5, 6.

Quá trình ghi dữ liệu vào EEPROM cũng có thể không an toàn nếu điện thế nguồn nuôi (Vcc) quá thấp.

Đoạn chương trình sau thực hiện quá trình ghi dữ liệu vào EEPROM.

```
EEPROM_write:
; chờ cho bit EEWE về 0
```

```
sbic EECR,EEWE
rjmp EEPROM_write
;cấm các ngắt
cli
; ghi địa chỉ vào thanh ghi EEAR
out EEARH, r18
out EEARL, r17
; Ghi dữ liệu vào thanh ghi EEDR
out EEDR,r16
; set bit EEMWE thành 1
sbi EECR,EEMWE
; Set bit EEWE lên 1 để bắt đầu ghi vào EEPROM
sbi EECR,EEWE
; cho phép các ngắt hoạt động trở lại
sei
ret
```

Đọc dữ liệu từ EEPROM:

Việc đọc dữ liệu từ EEPROM đơn giản hơn ghi dữ liệu vào EEPROM, để đọc dữ liệu từ EEPROM ta thực hiện các bước sau:

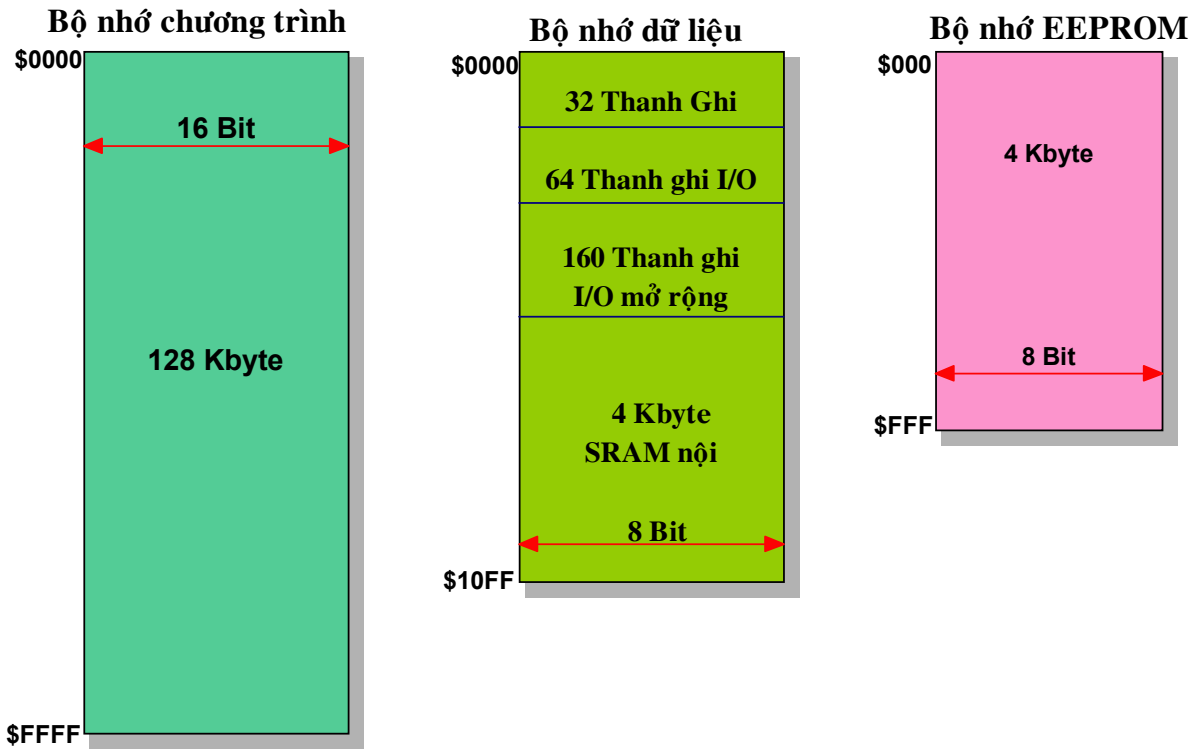
1. Chờ cho bit EEWE về 0.
2. Ghi địa chỉ vào thanh ghi EEAR.
3. Set bit EERE lên 1.

Đoạn chương trình sau thực hiện quá trình đọc dữ liệu từ EEPROM.

```
EEPROM_read:
; chờ cho bit EEWE về 0
sbic EECR,EEWE
rjmp EEPROM_read
; Đưa địa chỉ vào thanh ghi EEAR
out EEARH, r18
out EEARL, r17
; Set bit EERE để bắt đầu đọc EEPROM
sbi EECR,EERE

; Đưa dữ liệu vào thanh ghi R16
in r16,EEDR
ret
```

Tóm tắt: Bản đồ bộ nhớ bên trong của ATmega128 có thể tóm tắt lại như sau:



Bản đồ bộ nhớ ATmega128

Hình 2.5. Tóm tắt bản đồ bộ nhớ ATmega128

II. CỔNG VÀO RA

II.1. GIỚI THIỆU

Cổng vào ra là một trong số các phương tiện để vi điều khiển giao tiếp với các thiết bị ngoại vi. ATmega128 có cả thảy 7 cổng (port) vào ra 8 bit là : PortA, PortB, PortC, PortD, PortE, PortF, PortG, tương ứng với 56 đường vào ra. Các cổng vào ra của AVR là cổng vào ra hai chiều có thể định hướng, tức có thể chọn hướng của cổng là hướng vào (input) hay hướng ra (output). Tất các các cổng vào ra của AVR điều có tính năng Đọc – Chỉnh sửa – Ghi (Read – Modify – write) khi sử dụng chúng như là các cổng vào ra số thông thường. Điều này có nghĩa là khi ta thay đổi hướng của một chân nào đó thì nó không làm ảnh hưởng tới hướng của các chân khác. Tất cả các chân của các cổng (port)

điều có điện trở kéo lên (pull-up) riêng, ta có thể cho phép hay không cho phép điện trở kéo lên này hoạt động.

Điện trở kéo lên là một điện trở được dùng khi thiết kế các mạch điện tử logic. Nó có một đầu được nối với nguồn điện áp dương (thường là Vcc hoặc Vdd) và đầu còn lại được nối với tín hiệu lối vào/ra của một mạch logic chức năng. Điện trở kéo lên có thể được lắp đặt tại các **lối vào** của các khối mạch logic để thiết lập mức logic lối vào của khối mạch khi không có thiết bị ngoài nối với lối vào. Điện trở kéo lên cũng có thể được lắp đặt tại các **giao diện giữa hai khối mạch logic** không cùng loại logic, đặc biệt là khi hai khối mạch này được cấp nguồn khác nhau. Ngoài ra, điện trở kéo lên còn được lắp đặt tại **lối ra** của khối mạch khi lối ra không thể nối nguồn để tạo dòng, ví dụ các linh kiện logic TTL có cực góp hở. Đối với họ logic lưỡng cực với nguồn nuôi 5 Vdc thì giá trị của điện trở kéo lên thường nằm trong khoảng 1000 đến 5000 Ohm, tùy theo yêu cầu cấp dòng trên toàn giải hoạt động của mạch. Với lôgíc CMOS và lôgíc MOS chúng ta có thể sử dụng các điện trở có giá trị lớn hơn nhiều, thường từ vài ngàn đến một triệu Ohm do dòng rò rỉ cần thiết ở lối vào là rất nhỏ. Trong việc thiết kế các vi mạch ứng dụng, nếu một IC có ngõ ra loại cực thu để hở giao tiếp với nhiều IC khác thì giá trị của điện trở kéo lên sẽ tương đối nhỏ (khoảng vài trăm Ohm). Bởi vì lúc này hệ số fanout lớn dẫn đến dòng ngõ ra của IC phải lớn để đủ cung cấp cho các ngõ vào của các IC khác, nếu không vi mạch sẽ hoạt động chập chờn hoặc có thể không hoạt động.

II.2. CÁCH HOẠT ĐỘNG :

Khi khảo sát các cổng như là các cổng vào ra số thông thường thì tính chất của các cổng (PortA, PortB,…PortG) là tương tự nhau, nên ta chỉ cần khảo sát một cổng nào đó trong số 7 cổng của vi điều khiển là đủ.

Mỗi một cổng vào ra của vi điều khiển được liên kết với 3 thanh ghi : PORTx, DDRx, PINx. (ở đây x là để thay thế cho A, B,…G). Ba thanh ghi này sẽ được phối hợp với nhau để điều khiển hoạt động của cổng, chẳn hạn thiết lập cổng thành lối vào có sử dụng điện trở pull-up, ..v.v.. . Sau đây là diễn tả cụ thể vai trò của 3 thanh ghi trên.

a. Thanh Ghi DDRx.

Đây là thanh ghi 8 bit (có thể đọc ghi) có chức năng điều khiển hướng của cổng (là lối ra hay lối vào). Khi một bit của thanh ghi này được set lên 1 thì chân tương ứng với nó được cấu hình thành ngõ ra. Ngược lại, nếu bit của thanh ghi DDRx là 0 thì chân tương ứng với nó được thiết lập thành ngõ vào. Lấy ví dụ: Khi ta set tất cả 8 bit của thanh ghi DDRA đều là 1, thì 8 chân tương ứng của portA là PA1, PA2, … PA7 (tương ứng với các chân số 50, 49, …44 của vi điều khiển) được thiết lập thành ngõ ra.

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi DDRA

b. Thanh Ghi PORTx.

PORTx là thanh ghi 8 bit có thể đọc ghi. Đây là thanh ghi dữ liệu của PORTx, Nếu thanh ghi DDRx thiết lập cổng là lối ra, khi đó giá trị của thanh ghi PORTx cũng là giá trị của các chân tương ứng của PORTx, nói cách khác, khi ta ghi một giá trị logic lên 1 bit của thanh ghi này thì chân tương ứng với bit đó cũng có cùng mức logic. Khi thanh ghi DDRx thiết lập cổng thành lối vào thì thanh ghi PORTx đóng vai trò như một thanh ghi điều khiển cổng. Cụ thể , nếu một bit của thanh ghi này được ghi thành 1 thì điện trở treo (pull-up resistor) ở chân tương ứng với nó sẽ được kích hoạt, ngược lại nếu bit được ghi thành 0 thì điện trở treo ở chân tương ứng sẽ không được kích hoạt, cổng ở trạng thái cao trở (Hi-Z).

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi PORTA

c. Thanh Ghi PINx.

PINx không phải là một thanh ghi thực sự, đây là địa chỉ trong bộ nhớ I/O kết nối trực tiếp tới các chân của cổng. Khi ta đọc PORTx tức ta đọc dữ liệu được chốt trong PORTx, còn khi đọc PINx thì giá trị logic hiện thời ở chân của cổng tương ứng được đọc. Vì thế đối với thanh ghi PINx ta chỉ có thể đọc mà không thể ghi. Bảng 25 thể hiện các các thiết lập cách hoạt có thể có của cổng.

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

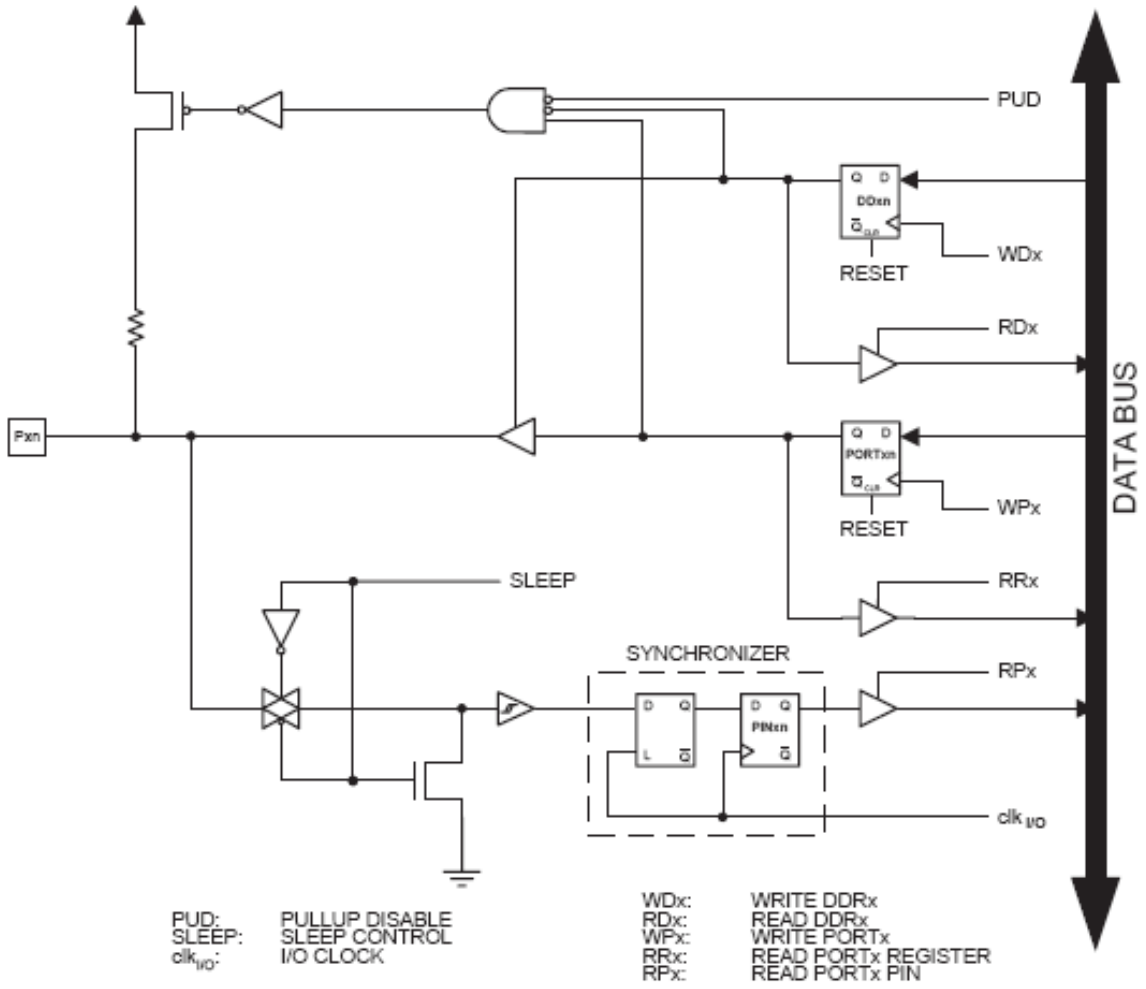
Thanh ghi PINA

DDRxn	PORTxn	PUD (Trong thanh ghi SFIOR	I/O	Pull-up	**Chú thích**
0	0	x	Ngõ vào	không	Cao trở
0	1	0	Ngõ vào	có	Như một nguồn dòng
0	1	1	Ngõ vào	không	Cao trở
1	0	x	Ngõ ra	không	Ngõ ra thấp
1	1	x	Ngõ ra	không	Ngõ ra cao

Bảng 25. Cấu hình cho các chân cổng

DDRxn là bit thứ n của thanh ghi DDRx
PORTxn là bit thứ n của thanh ghi PORTx
Dấu "x" ở cột thứ 3 để chỉ giá trị logic là tùy ý

Figure 30. General Digital I/O(1)



Hình 30. Sơ đồ một cổng vào ra

Hình 30 thể hiện sơ đồ của một chân của cổng vào ra. Ở sơ đồ trên ta thấy ngoài 2 bit của các thanh ghi DDRx và PORTx tham gia điều khiển điện trở treo (pull-up resistor), còn có một tín hiệu nữa điều khiển điện trở treo, đó là tín hiệu PUD, đây là bit nằm trong thanh ghi SFIOR, khi set bit này thành 1 thì điện trở kéo lên sẽ không được cho phép bất kể các thiết lập của các thanh ghi DDRx và PORTx. Khi bit này là 0 thì điện trở kéo lên được cho phép nếu { DDRxn, PORTxn } = { 0, 1 } .

Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi SFIOR

Dưới đây là địa chỉ của tất cả các port :

Tên PORT	Địa chỉ I/O	Địa chỉ SRAM
PORTA	$1B	$3B
DDRA	$1A	$3A
PINA	$19	$39
PORTB	$18	$38
DDRB	$17	$37
PINB	$16	$36
PORTC	$15	$35
DDRC	$14	$34
PINC	$13	$33
PORTD	$12	$32
DDRD	$11	$31
PIND	$10	$30
PORTE	$03	$23
DDRE	$02	$22
PINE	$01	$21
PORTF	Không có	$62
DDRF	Không có	$61
PINF	$00	$20
PORTG	Không có	$65
DDRG	Không có	$64
PING	Không có	$63

Để ý : 3 bit cuối (bit 5, 6, 7) của các thanh ghi PORTG, DDRG và PING không sử dụng được. Khi đọc ta luôn nhận được giá trị 0.

Chương III

BỘ ĐỊNH THỜI CỦA ATmega128

ATmega128 có 4 bộ định thời , bộ định thời 1 và 3 là bộ định thời 16 bit, bộ định thời 0 và 2 là bộ định thời 8 bit. Dưới đây là mô tả chi tiết của 4 bộ định thời.

I. BỘ ĐỊNH THỜI 1.

Sơ đồ khối bộ định thời 1 (3):

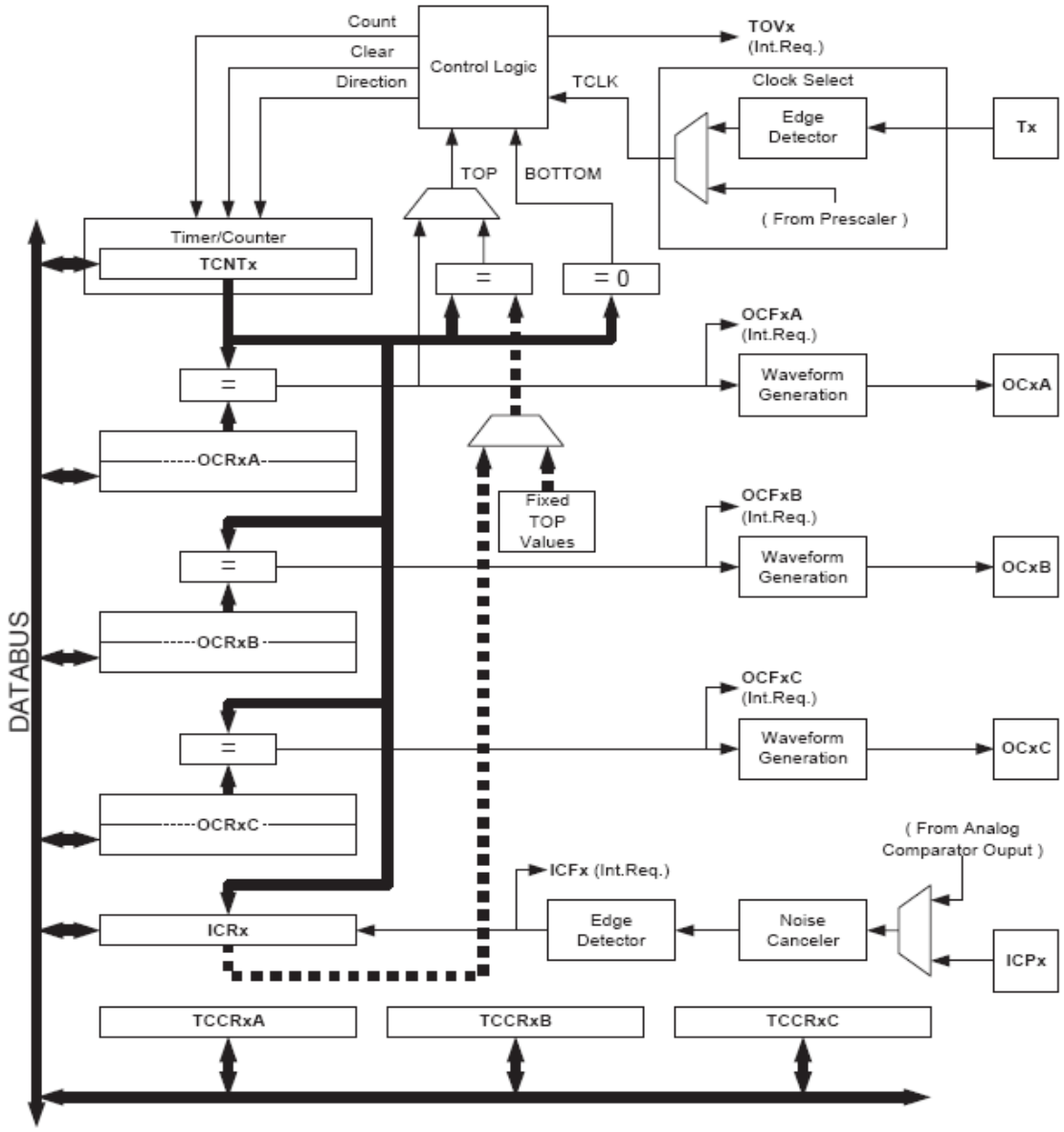


Figure 46. 16-bit Timer/Counter Block Diagram

Bộ định thời 1 và 3 là bộ định thời 16 bit, bộ định thời 1 sử dụng 13 thanh ghi liên quan, còn bộ định thời 3 sử dụng 11 thanh ghi liên quan với nhiều chế độ thực thi khác nhau.Vì bộ định thời 1 và 3 hoạt động giống nhau nên ở đây chỉ trình bày bộ định thời 1. Một đểm cần để ý là trong các thanh ghi liên quan tới bộ định thời 1 và 3 thì có nhiều thanh ghi được chia sẽ cho cả hai bộ định thời, chẳn hạn thanh ghi ETIPR có bít cuối là OCF1C được dùng cho bộ định thời 1, các bit còn lại là dùng cho bộ định thời 3. Thậm chí có những thanh ghi chia sẽ cho bộ định thời 0 hoặc 2, chẳn hạn thanh ghi TIMSK có hai bit cuối dùng cho bộ định thời 2, hai bit đầu dùng cho bộ định thời 0, các bit còn lại dùng cho bộ định thời 1. Các thanh ghi liên quan tới bộ định thời 3 cũng được liệt kê ra mà không cần giải thích chi tiết, tuy vậy cũng có vài khác biệt nhỏ giữa bộ định thời 1 và 3 được chú thích cho từng trường hợp cụ thể trong mục "Bộ Định Thời 3". Để tìm hiểu về bộ định thời 1 (3) ta cần nắm vững các thanh ghi liên quan tới bộ định thời 1(3) và các chế độ hoạt động của bộ định thời.

❖ CÁC ĐỊNH NGHĨA:

Các định nghĩa sau sẽ được sử dụng cho bộ định thời 1 và 3 :

BOTTOM Bộ đếm đạt tới giá trị BOTTOM khi nó có giá trị 0000h

MAX Bộ đếm đạt tới giá trị MAX khi nó bằng FFFFh

TOP Bộ đếm đạt giá trị TOP khi nó bằng với giá trị cao nhất trong chuỗi đếm, giá trị cao nhất trong chuỗi đếm không nhất thiết là FFFFh mà có thể là bất khì giá trị nào được qui định trong thanh ghi OCRnX (X=A,B,C) hay ICRn, tùy theo chế độ thực thi.

❖ CÁC THANH GHI BỘ ĐỊNH THỜI 1.

1. Thanh ghi TCCR1A (Timer/Counter1 Control Register)

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**
- **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

• **Bit 7:2 – COMnX1:0 (X=A, B, C): Compare Output Mode for Channel X** : Điều khiển cách hoạt động của *ngõ ra so sánh (compare output)* của lần lượt các chân OCnA, OCnB và OCnC. Nếu một hay cả hai bit COMnA1:0 được set lên 1 thì ngõ ra

OCnA sẽ ưu tiên hơn chức năng port I/O thông thường mà nó kết nối tới . Nếu một hay cả hai bit COMnB1:0 được set lên 1 thì ngõ ra OCnB sẽ ưu tiên hơn chức năng port I/O thông thường mà nó kết nối tới . Nếu một hay cả hai bit COMnC1:0 được set lên 1 thì ngõ ra OCnC sẽ ưu tiên hơn chức năng port I/O thông thường mà nó kết nối tới, điều này có nghĩa là mỗi một chân của vi điều khiển có thể thực hiện nhiều chức năng khác nhau, bình thường các chân OCnA, OCnB, OCnC hoạt động như các chân vào ra thông thường, nhưng khi bộ định thời đang hoạt động ở các chế độ có sử dụng tới chức năng **so sánh khớp (compare match)** như các chế độ CTC, PWM,…của bộ định thời thì hành vi của chân ngõ ra OCnA, OCnB, OCnC sẽ do bộ định thời điều khiển. Tuy nhiên chú ý là bit của thanh ghi DDR tương ứng với các chân OCnA, OCnB, OCnC phải được set để cho phép ngõ ra. Khi OCnA, OCnB, OCnC được kết nối tới chân thì tác dụng của các bit COMnX1:0 còn phụ thuộc vào lựa chọn của các bit WGM3:0, nghĩa là khi ta set một hay cả hai Bit COMn1:0 lên 1 thì chức năng ngõ ra so sánh được ưu tiên, tuy nhiên cách hoạt động ở ngõ ra OCnX như thế nào thì còn phụ thuộc vào việc lựa chọn của các bit WGMn3:0, được thể hiện trong các bảng dưới (Bảng 58, 59, 60).
Trong các chế độ PWM, khi giá trị các thanh ghi dùng để **so sánh** (OCRnX, ICRn) có giá trị bằng với TOP, thì sự kiện so sánh khớp (compare match) bị bỏ qua. Tuy vậy các chân OCnX vẫn bị set hay xóa (tùy vào các bit COMnX 1:0) ở BOTTOM.

Table 58. Compare Output Mode, non-PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	Toggle OCnA/OCnB/OCnC on compare match.
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level).
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level).

Bảng 58. Hành vi của các chân OCnX (X=A, B, C; n=1, 3) phụ thuộc vào các thiết lập của các bit COMnA1:0, COMnB1:0, COMnC1:0 trong chế độ non-PWM

Table 59. Compare Output Mode, Fast PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGMn3:0 = 15: Toggle OCnA on Compare Match, OCnB/OCnC disconnected (normal port operation). For all other WGMn settings, normal port operation, OCnA/OCnB/OCnC disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at TOP
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at TOP

Bảng 59. Hành vi của các chân OCnX (X=A, B, C; n=1, 3) phụ thuộc vào các thiết lập của các bit COMnA1:0, COMnB1:0, COMnC1:0 tromg chế độ Fast-PWM

Table 60. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGMn3:0 = 9 or 14: Toggle OCnA on Compare Match, OCnB/OCnC disconnected (normal port operation). For all other WGMn settings, normal port operation, OCnA/OCnB/OCnC disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting. Set OCnA/OCnB/OCnC on compare match when downcounting.
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting. Clear OCnA/OCnB/OCnC on compare match when downcounting.

Bảng 60. Hành vi của các chân OCnX (X=A, B, C; n=1, 3) phụ thuộc vào các thiết lập của các bit COMnA1:0, COMnB1:0, COMnC1:0 tromg chế độ PWM hiệu chỉnh pha và tần số

• **Bit 1:0 – WGMn1:0: Waveform Generation Mode :** Kết hợp với các bit WGMn3:2 tìm trong thanh ghi TCCRnB , những bit này cho phép ta lựa chọn chế độ thực thi của bộ định thời, nhờ đó có thể điều khiển việc đếm tuần tự của bộ đếm. Giá trị bộ đếm lớn nhất là TOP và dạng sóng tạo ra ở chân OCnX (X=A, B, C; n=1, 3) được sử dụng cho nhiều mục đích khác nhau (bảng 61). Các chế độ thực thi được hỗ trợ bởi khối Timer/counter là : Normal mode (counter), Clear Timer on Compare match (CTC) mode , PWM mode. Để ý là với bộ định thời 1 thì có 4 bit WGM là: WGM13, WGM12,WGM11 và WGM10.

Table 61. Waveform Generation Mode Bit Description

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation(1)	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	TOP	TOP
15	1	1	1	1	Fast PWM	OCRnA	TOP	TOP

Bảng 61. Lựa chọn các chế độ thực thi của bộ định thời 1(3)

2. Thanh ghi TCCR1B

7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

• **Bit 7 – ICNCn: Input Capture Noise Canceler**
• **Bit 6 – ICESn: Input Capture Edge Select**
• **Bit 5 – Reserved Bit**

- **Bit 4:3 – WGMn3:2: Waveform Generation Mode**
- **Bit 2:0 – CSn2:0: Clock Select**

• **Bit 7 – ICNCn: Input Capture Noise Canceler** (viết tắt: ICNC): Việc set bit này tới 1 sẽ kích hoạt chức năng chống nhiễu của ***bộ chống nhiễu lối vào*** (ICNC). Khi chức năng ICNC được kích hoạt thì ngõ vào từ chân ICPn sẽ được lọc. Chức năng lọc đòi hỏi 4 mẫu có giá trị bằng nhau liên tiếp ở chân ICPn cho sự thay đổi ngõ ra của nó (xem chi tiết về khối **Input Capture**).

• **Bit 6 – ICESn: Input Capture Edge Select**: Bit này lựa chọn cạnh ở chân Input Capture Pin (ICPn) dùng để bắt "sự kiện trigger" (Trigger event (10)). Khi bit ICESn được thiết lập thành 0 thì một cạnh dương xuống (falling (3)) được dùng như một trigger (tín hiệu nảy). Ngược lại, khi bit này được set thành 1 thì một cạnh âm lên (rising (4)) được dùng như một trigger. Khi xảy ra sự kiện Input capture (2) (theo thiết lập của bit ICESn là 1 hay 0) thì giá trị của bộ đếm được ghi vào thanh ghi Input Capture Register ICRn (n=1, 3), và khi đó cờ ICFn (Input Capture Flag) được set. Điều này sẽ tạo ra một ngắt Input capture nếu ngắt này được cho phép. Khi thanh ghi ICRn được sử dụng như một giá trị TOP thì chân ICPn không được kết nối và vì thế chức năng Input capture không được cho phép.

• **Bit 5 :** Dự trữ.

• **Bit 4:3 – WGMn3:2: Waveform Generation Mode**: Đã nói ở phần thanh ghi TCCR1A.

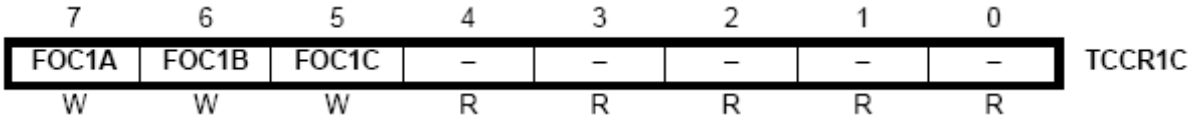
• **Bit 2:0 – CSn2:0: Clock Select :** Dùng để lựa chọn tốc độ xung clock (xem bảng 62). Để cấm bộ định thời hoạt động ta chỉ cần cho {CSn2, CSn1, CSn0} = {0, 0, 0}.

Table 62. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Bảng 62. ***Lựa chọn tốc độ xung clock***

3. Thanh ghi TCCR1C

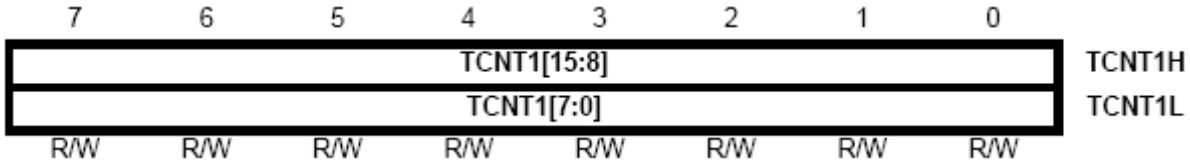


- **Bit 7 – FOCnA: Force Output Compare for Channel A**
- **Bit 6 – FOCnB: Force Output Compare for Channel B**
- **Bit 5 – FOCnC: Force Output Compare for Channel C**
- **Bit 4:0 – Reserved Bits**

Các bit FOCnA/FOCnB/FOCnC chỉ hoạt động khi các bit WGMn3:0 chỉ định chế độ Non-PWM. Khi các bit FOCnA/FFOCnB/FOCnC được set thành 1 thì ngay lập tức một sự kiện "**So sánh khớp cưỡng chế**" (Forced Compare Match [(1)]) xảy ra trong bộ tạo sóng. Ngõ ra OCnA/OCnB/OCnC được thay đổi theo thiết lập của các bit COMnX 1:0 (n=1, 3; X=A, B, C), nghĩa là bình thường sự kiện "**so sánh khớp**" chỉ xảy ra khi khi giá trị bộ định thời (thanh ghi TCNTn (n=1, 3)) bằng với giá trị thanh ghi OCRnX(n=1,3; X=A,B,C), nhưng khi các bit FOCnX(n=1, 3; X=A, B, C) được set thành 1 thì sự kiện "**so sánh khớp**" sẽ xảy ra mặc dù giá trị của bộ định thời không bằng với giá trị của thanh ghi OCRnX(n=1,3; X=A,B,C). Chú ý là các bit FOCnA/FOCnB/FOCnC cũng hoạt động như là những que dò (strobe), vì thế nó là giá trị hiện thời của các bit COMnX1:0 xác định tác động của "**so sánh cưỡng chế**" (forced compare). Các que dò FOCnA/FOCnB/FOCnC không tạo ra bất kì ngắt nào và cũng không xóa bộ định thời trong chế độ CTC sử dụng thanh ghi OCRnA như là giá trị TOP. Các bit FOCnA/FOCnB/FOCnC chỉ có thể ghi, khi đọc các bit này ta luôn nhận được giá trị 0.

- **Bit 4:0** dự trữ ,phải ghi thành 0 khi ghi vào thanh ghi TCCRnC.

4. Thanh Ghi Timer/Counter1 – TCNT1H and TCNT1L



Thanh ghi bộ định thời TCNT1 là thanh ghi 16 bit được kết hợp từ hai thanh ghi TCNT1H và thanh ghi TCNT1L. Thanh ghi TCNT1 có thể đọc hay ghi. Để cả 2 byte của TCNT 1 được đọc hay ghi đồng thời người ta dùng một thanh ghi tạm 8 bit byte cao 8-bit Temporary High Byte Register (TEMP). Thanh ghi TEMP được chia sẽ cho tất cả các thanh ghi 16 bit khác. Không nên chỉnh sửa thanh ghi TCNTn (n=1,3) khi nó đang đếm để tránh bị hỏng Compare Match giữa TCNTn và một trong những thanh ghi OCRnX(n=1,3. X=A,B,C).

5. Thanh Ghi Output Compare Register 1 A– OCR1AH and OCR1AL

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

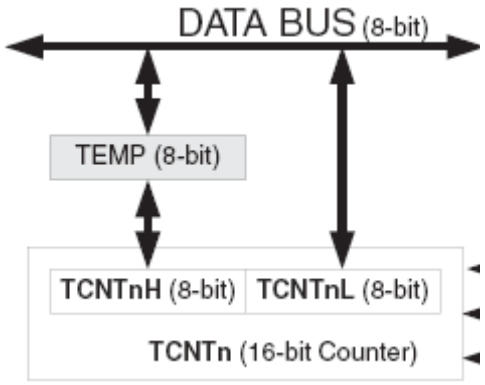
6. Thanh Ghi Output Compare Register 1 B– OCR1BH and OCR1BL

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

7. Thanh Ghi Output Compare Register 1 C– OCR1CH and OCR1CL

Bit	7	6	5	4	3	2	1	0	
	OCR1C[15:8]								OCR1CH
	OCR1C[7:0]								OCR1CL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi output compare register (OCR1A/OCR1B/OCR1C) là thanh ghi 16 bit, giá trị của nó được liên tục so sánh với bộ đếm (TCNT1). Khi có sự bằng nhau của hai thanh ghi này sẽ tạo ra một ngắt so sánh hay một dạng sóng ở chân ngõ ra so sánh OCnX (X=A,B,C). Giống như thanh ghi TCNT1 , thanh ghi OCRnX (X=A,B,C) cũng là thanh ghi 16 bit nên để cả hai byte cao và thấp của thanh ghi được ghi hay đọc đồng thời khi CPU cần truy xuất thanh ghi này, người ta dùng thanh ghi tạm byte cao (TEMP), thanh ghi TEMP luôn lưu giữ byte cao của các thanh ghi 16 bit khi các thanh ghi này cần dùng tới nó (xem hình **3.1**). Chú ý là khi ghi một giá trị vào thanh ghi OCRnX trong lúc bộ đếm đang chạy, thì giá trị của thanh ghi OCRnX có thể cập nhật tức thời, nhưng cũng có thể chỉ được cập nhật khi bộ đếm đạt tới một giá trị nào đó (**bảng 61**), chẳng hạn, giá trị TOP, BOTTOM…



Hình 3.1. Thanh ghi TEMP

8. Thanh Ghi Input Capture Register 1 –ICR1H and ICR1L

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi Input capture (ICR1n) sẽ cập nhật giá trị của bộ đếm TCNTn mỗi khi xảy ra sự kiện ở chân ICPn. Ngoài ra thanh ghi này còn được sử dụng để định nghĩa giá trị TOP của bộ đếm. Người ta cũng sử dụng thanh ghi TEMP khi cần truy xuất thanh ghi ICRn (n=1, 3).

9. Thanh Ghi Timer/Counter Interrupt Mask Register – TIMSK (Interrupt for Timer/counter 1)

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable**
- **Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**
- **Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**
- **Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

Bit 5 – TICIE1: Timer/Counter1, Input Capture Interrupt Enable: Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt bắt mẫu ngõ vào bộ Timer/couter1 (Timer/Counter1 Input Capture interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ ICF1 trong thanh ghi TIFR được set.

Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable: Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt so sánh ngõ ra 1A (Timer/Counter1 Output Compare A Match Interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ OCF1A trong thanh ghi TIFR được set.

Bit 3 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable: Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt so sánh ngõ ra 1B (Timer/Counter1 Output Compare B Match Interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ OCF1B trong thanh ghi TIFR được set.

Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable: Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt cờ tràn bộ định thời 1 (Timer/Counter1 overflow interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ TOV1 trong thanh ghi TIFR được set.

10. Thanh Ghi Extended Timer/Counter Interrupt Mask Register –ETIMSK (Interrupt for Timer/counter 3)

Bit	7	6	5	4	3	2	1	0	
	–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C	ETIMSK
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – Reserved Bits**
- **Bit 5 – TICIE3: Timer/Counter3, Input Capture Interrupt Enable**
- **Bit 4 – OCIE3A: Timer/Counter3, Output Compare A Match Interrupt Enable**
- **Bit 3 – OCIE3B: Timer/Counter3, Output Compare B Match Interrupt Enable**
- **Bit 2 – TOIE3: Timer/Counter3, Overflow Interrupt Enable**
- **Bit 1 – OCIE3C: Timer/Counter3, Output Compare C Match Interrupt Enable**
- **Bit 0 – OCIE1C: Timer/Counter1, Output Compare C Match Interrupt Enable**

Thanh ghi ETIMSK liên quan đến cả hai bộ định thời 1 và 3.

• **Bit 7:6 – Reserved Bits:** Dự trữ , phải ghi các bit này thành 0 khi ghi vào thanh ghi ETIMSK

• **Bit 5 – TICIE3: Timer/Counter3, Input Capture Interrupt Enable:** Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt bắt mẫu ngõ

vào bộ Timer/couter 3 (Timer/Counter3 Input Capture interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ ICF3 trong thanh ghi ETIFR được set.

• **Bit 4 – OCIE3A: Timer/Counter3, Output Compare A Match Interrupt Enable:** Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt so sánh ngõ ra 3A (Timer/Counter1 Output Compare A Match Interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ OCF3A trong thanh ghi ETIFR được set.

• **Bit 3 – OCIE3B: Timer/Counter3, Output Compare B Match Interrupt Enable:** Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt so sánh ngõ ra 3B (Timer/Counter3 Output Compare B Match Interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ OCF3B trong thanh ghi ETIFR được set.

• **Bit 2 – TOIE3: Timer/Counter3, Overflow Interrupt Enable:** Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt cờ tràn bộ định thời 3 (Timer/Counter3 overflow interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ TOV4 trong thanh ghi ETIFR được set.

• **Bit 1 – OCIE3C: Timer/Counter3, Output Compare C Match Interrupt Enable:** Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt so sánh ngõ ra 3C (Timer/Counter3 Output Compare C Match Interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ OCF3C trong thanh ghi ETIFR được set.

• **Bit 0 – OCIE1C: Timer/Counter1, Output Compare C Match Interrupt Enable:** Khi bit này được set thành 1 và ngắt toàn cục (global interrupt) được cho phép thì ngắt so sánh ngõ ra 1C (Timer/Counter1 Output Compare C Match Interrupt) được cho phép. Vector ngắt tương ứng sẽ được thực thi khi cờ OCF1C trong thanh ghi ETIFR được set.

11. Thanh Ghi Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**
• **Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag**
• **Bit 3 – OCF1B: Timer/Counter1, Output Compare B Match Flag**
• **Bit 2 – TOV1: Timer/Counter1, Overflow Flag**

Thanh ghi TIFR liên quan tới bộ định thời 1 và 2.

• **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag:** Cờ này được set khi xảy ra sự kiện bắt mẫu ngõ vào (Input Capture) của chân ICP1. Khi thanh ghi ICR1 (Input Capture Register) được thiết lập bởi các bit WGMn3:0 để sử dụng như một giá trị TOP thì cờ ICF1 sẽ được set khi bộ đếm đạt tới giá trị TOP. Cờ ICF1 sẽ tự động xóa khi ngắt tương ứng được thực thi, hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

• **Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag:** Cờ này được set ngay sau khi giá trị bộ đếm (TCNT1) bằng với giá trị thanh ghi OCR1A (Output Compare Register A). Chú ý là một so sánh cưỡng bức (FOC1A) sẽ không set cờ này. Cờ OCF1A sẽ tự động xóa khi ngắt tương ứng được thực thi, hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

• **Bit 3 – OCF1B: Timer/Counter1, Output Compare B Match Flag:** Cờ này được set ngay sau khi giá trị bộ đếm (TCNT1) bằng với giá trị thanh ghi OCR1B (Output Compare Register B). Chú ý là một so sánh cưỡng bức (FOC1B) sẽ không set cờ này. Cờ OCF1B sẽ tự động xóa khi ngắt tương ứng được thực thi, hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

• **Bit 2 – TOV1: Timer/Counter1, Overflow Flag:** Việc thiết lập cờ này phụ thuộc vào thiết lập của các bit WGMn3:0, trong chế độ bình thường và CTC cờ TOV1 được set khi bộ định thời tràn. Xem lại bảng 61 và mục "**Các chế độ thực thi**" để biết các trường hợp khác.

12. Thanh Ghi Extended Timer/Counter Interrupt Flag Register –ETIFR

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C	ETIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7:6 – Reserved Bits**
• **Bit 5 – ICF3: Timer/Counter3, Input Capture Flag**
• **Bit 4 – OCF3A: Timer/Counter3, Output Compare A Match Flag**
• **Bit 3 – OCF3B: Timer/Counter3, Output Compare B Match Flag**
• **Bit 2 – TOV3: Timer/Counter3, Overflow Flag**
• **Bit 1 – OCF3C: Timer/Counter3, Output Compare C Match Flag**
• **Bit 0 – OCF1C: Timer/Counter1, Output Compare C Match Flag**

• **Bit 7:6 – Reserved Bits**: Dự trữ, phải ghi 0 khi ghi vào thanh ghi ETIFR.

• **Bit 5 – ICF3: Timer/Counter3, Input Capture Flag:** Cờ này được set khi xảy ra sự kiện bắt ngõ vào (Input Capture) của chân ICP3. Khi thanh ghi ICR3 (Input Capture Register) được thiết lập bởi các bit WGMn3:0 để sử dụng như một giá trị TOP thì cờ ICF3

sẽ được set khi bộ đếm đạt tới giá trị TOP. Cờ ICF3 sẽ tự động xóa khi ngắt tương ứng được thực thi,hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

• **Bit 4 – OCF3A: Timer/Counter3, Output Compare A Match Flag: :** Cờ này được set ngay sau khi giá trị bộ đếm (TCNT3) bằng với giá trị thanh ghi OCR3A (Output Compare Register A). Chú ý là một so sánh cưỡng bức (FOC3A) sẽ không set cờ này. Cờ OCF3A sẽ tự động xóa khi ngắt tương ứng được thực thi, hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

• **Bit 3 – OCF3B: Timer/Counter3, Output Compare B Match Flag:** Cờ này được set ngay sau khi giá trị bộ đếm (TCNT3) bằng với giá trị thanh ghi OCR3B (Output Compare Register B).Chú ý là một so sánh cưỡng bức (FOC3B) sẽ không set cờ này. Cờ OCF3B sẽ tự động xóa khi ngắt tương ứng được thực thi, hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

• **Bit 2 – TOV3: Timer/Counter3, Overflow Flag:** Việc thiết lập cờ này phụ thuộc vào thiết lập của các bit WGMn3:0, trong chế độ bình thường và CTC cờ TOV3 được set khi bộ định thời tràn. Xem lại bảng 52 và mục "**Các chế độ thực thi**" để biết các trường hợp khác.

• **Bit 1 – OCF3C: Timer/Counter3, Output Compare C Match Flag:** Cờ này được set ngay sau khi giá trị bộ đếm (TCNT3) bằng với giá trị thanh ghi OCR3C (Output Compare Register C). Chú ý là một so sánh cưỡng bức (FOC3C) sẽ không set cờ này. Cờ OCF3C sẽ tự động xóa khi ngắt tương ứng được thực thi, hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

• **Bit 0 – OCF1C: Timer/Counter1, Output Compare C Match Flag:** Cờ này được set ngay sau khi giá trị bộ đếm (TCNT1) bằng với giá trị thanh ghi OCR1C (Output Compare Register C). Chú ý là một so sánh cưỡng bức (FOC1C) sẽ không set cờ này. Cờ OCF1C sẽ tự động xóa khi ngắt tương ứng được thực thi, hoặc có thể xóa hay set bằng cách ghi một giá trị logic vào vị trí của nó.

13. Thanh Ghi Special Function IO Register –SFIOR

Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – TSM: Timer/Counter Synchronization Mode**

• **Bit 0 – PSR321: Prescaler Reset Timer/Counter3, Timer/Counter2, and Timer/Counter1**

• **Bit 7 – TSM: Timer/Counter Synchronization Mode:** Ghi bit này thành 1 sẽ kích hoạt chế độ "**Đồng bộ bộ định thời"**. Trong chế độ này giá trị ghi vào hai bit PSR0 và PSR321 được giữ, vì thế nó giữ cho tín hiệu reset của bộ chia trước (prescaler [8]) tương ứng được xác nhận (do đó bộ chia trước prescaler vẫn ở trạng thái Reset). Điều này để chắc chắn là các bộ Timer/couter tương ứng được tạm dừng để có thể được cấu hình với giá trị như nhau mà không làm hỏng các cấu hình sẵn có khác. Khi TMS là 0 thì các bit PSR0 và PSR321 được xóa bởi phần cứng và các bộ định thời (1,2,3) bắt đầu đếm đồng thời. (Xem thêm mục : **Chế Độ Đồng Bộ Bộ Định Thời**).

• **Bit 0 – PSR321: Prescaler Reset Timer/Counter3, Timer/Counter2, and Timer/Counter1:** Khi bit này là 1 thì bộ chia trước (prescaler) của ba bộ định thời 1,2,3 được reset. Bit PSR321 được xóa bởi phần cứng ngoại trừ trường hợp bit TSM được set. Chú ý là ba bộ định thời 1, 2, 3 cùng chia sẽ một bộ chia trước (prescaler) nên việc reset bộ chia trước (prescaler) sẽ tác động lên cả ba bộ định thời này.

II. BỘ ĐỊNH THỜI 3

Bộ định thời 3 giống bộ định thời 1 nên ở đây chỉ trình bày các thanh ghi liên quan tới bộ định thời 3, chức năng của từng thanh ghi có thể xem các thanh ghi tương ứng với nó ở bộ định thời 1.

1. Thanh ghi TCCR3A (Timer/Counter3 Control Register A)

Bit	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7:6 – COM3A1:0: Compare Output Mode for Channel A**
• **Bit 5:4 – COM3B1:0: Compare Output Mode for Channel B**
• **Bit 3:2 – COM3C1:0: Compare Output Mode for Channel C**
• **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

2. Thanh ghi TCCR3B (Timer/Counter3 Control Register B)

Để ý là khối **Input Capture Unit** của bộ định thời 3 có khác chút ít so với của bộ định thời 1. Xem chi tiết về khối **Input Capture Unit** ở phần mô tả "**Khối Input Capture Unit**".

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC3: Input Capture Noise Canceler**
- **Bit 6 – ICES3: Input Capture Edge Select**
- **Bit 5 – Reserved Bit**
- **Bit 4:3 – WGM3 3:2: Waveform Generation Mode**
- **Bit 2:0 – CS3 2:0: Clock Select**

3. Thanh ghi TCCR3C (Timer/Counter3 Control Register C)

Bit	7	6	5	4	3	2	1	0	
	FOC3A	FOC3B	FOC3C	–	–	–	–	–	TCCR3C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC3A: Force Output Compare for Channel A**
- **Bit 6 – FOC3B: Force Output Compare for Channel B**
- **Bit 5 – FOC3C: Force Output Compare for Channel C**
- **Bit 4:0 – Reserved Bits**

4. Thanh Ghi Timer/Counter1 – TCNT3H and TCNT3L

Bit	7	6	5	4	3	2	1	0	
	TCNT3[15:8]								TCNT3H
	TCNT3[7:0]								TCNT3L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5. Thanh Ghi Output Compare Register 3 A– OCR3AH and OCR3AL

Bit	7	6	5	4	3	2	1	0	
	OCR3A[15:8]								OCR3AH
	OCR3A[7:0]								OCR3AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

6. Thanh Ghi Output Compare Register 3 B– OCR3BH and OCR3BL

Bit	7	6	5	4	3	2	1	0	
	OCR3B[15:8]								OCR3BH
	OCR3B[7:0]								OCR3BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

7. Thanh Ghi Output Compare Register 3C– OCR3CH and OCR3CL

Bit	7	6	5	4	3	2	1	0	
	OCR3C[15:8]								OCR3CH
	OCR3C[7:0]								OCR3CL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

8. Thanh Ghi Input Capture Register 3 –ICR3H and ICR3L

Bit	7	6	5	4	3	2	1	0	
	ICR3[15:8]								ICR3H
	ICR3[7:0]								ICR3L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

9. Thanh Ghi Extended Timer/Counter Interrupt Mask Register –ETIMSK (Interrupt for Timer/counter 3)

Để ý là ở bộ định thời 1 có sử dụng thanh ghi TIMSK và ETIMSK , còn bộ định thời 3 chỉ sử dụng thanh ghi ETIMSK.

Bit	7	6	5	4	3	2	1	0	
	–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C	ETIMSK
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

10. Thanh Ghi Extended Timer/Counter Interrupt Flag Register –ETIFR

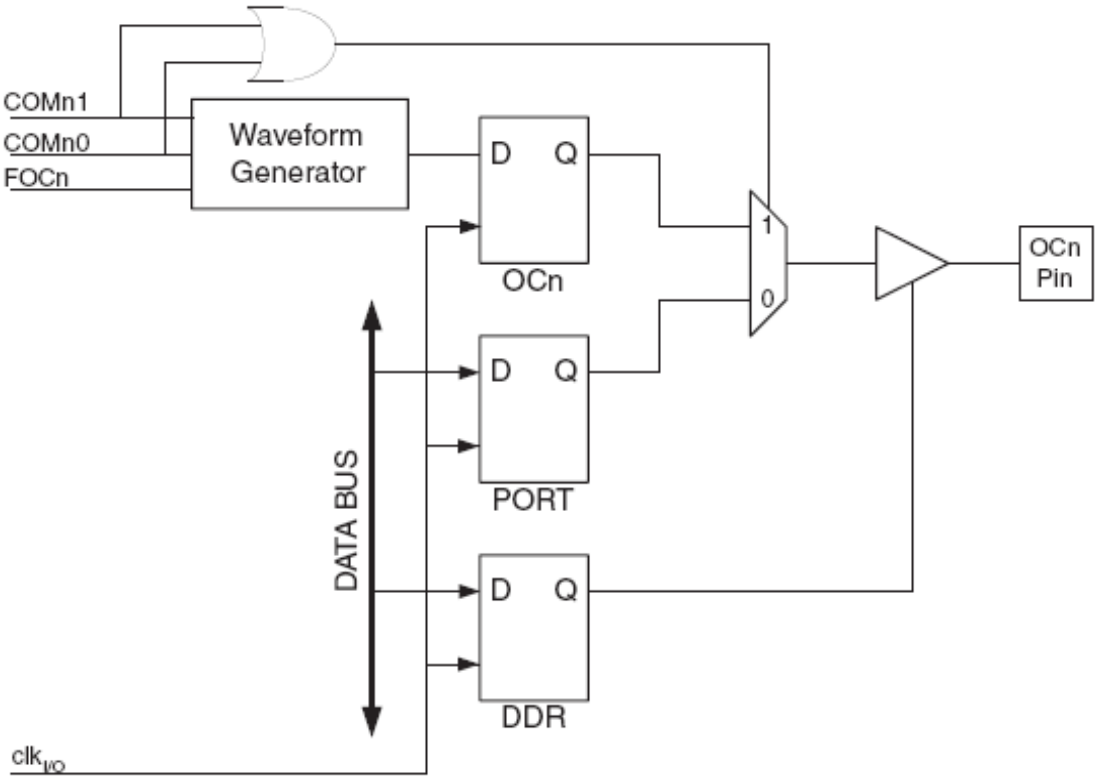
Để ý là bộ định thời 1 sử dụng cả 2 thanh ghi TIFR và ETIFR , còn bộ định thời 3 chỉ sử dụng thanh ghi TIFR.

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C	ETIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

11. Thanh Ghi Special Function IO Register –SFIOR

Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

12. Ngõ Ra Khối Compare Match Output Unit



Hình 3.2. Ngõ ra khối Compare Match Output Unit

Nhìn **hình 3.2** trên ta thấy Pin OCnX (chẳn hạn pin 15 của IC tương ứng với OC1A), là ngõ ra của khối **Compare Match Output Unit**, có thể được nối với 3 thanh ghi là OCnX, PortX và DDRX . Thanh ghi nào được nối với OCn là phụ thuộc vào các bit COMn1:0 (tức tùy theo chế độ hoạt động của bộ định thời), giả sử ta thiết lập các bit COMn1:0 để cho thanh ghi OCn được nối với PIN OCn, thì hoạt động của PIN OCn (tức dạng sóng ở ngõ ra OCn) lại phụ thuộc vào thiết lập của các bit WGMn3:0, các bit WGMn3:0 sẽ qui định dạng sóng ngõ ra tại OCn như thế nào (xem **bảng 61**). Ngược lại, nếu ta thiết lập bộ định thời hoạt động ở chế độ bình thường (tức không sử dụng chức năng "**so sánh khớp**" thì chân OCn trở thành chân vào ra số thông thường . Ngõ ra khối Compare Match Output Unit của bộ định thời 1 cũng giống như ở bộ định thời 3.

III. BỘ ĐỊNH THỜI 0

Sơ đồ khối bộ định thời 0

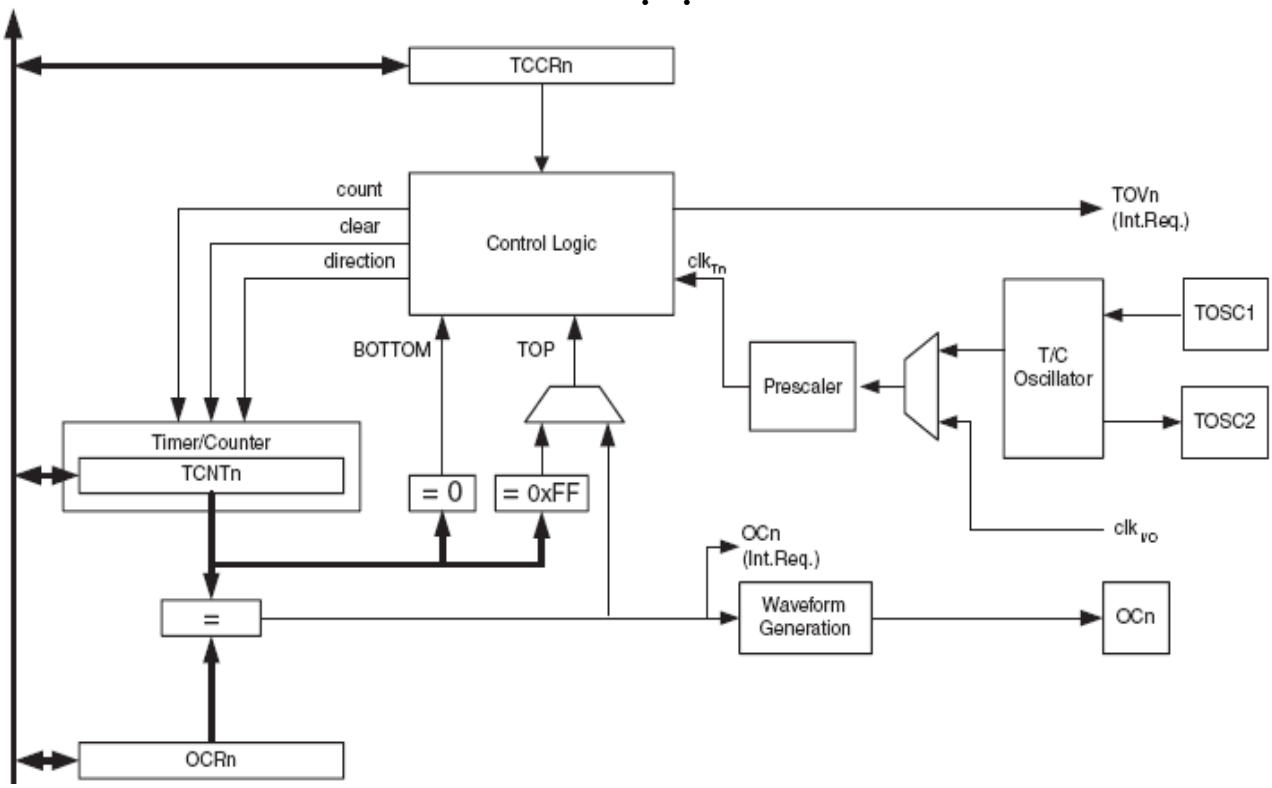


Figure 61. 8-Bit Timer/Counter Block Diagram

Bộ định thời 0 là bộ định thời 8 bit, bộ định thời 0 liên quan tới 7 thanh ghi với nhiều chế độ thực thi khác nhau.

❖ CÁC ĐỊNH NGHĨA:

Các định nghĩa sau sẽ được sử dụng cho bộ định thời 0 và 2:

BOTTOM Bộ đếm đạt tới giá trị BOTTOM khi nó có giá trị 00h.

MAX Bộ đếm đạt tới giá trị MAX khi nó bằng FFh.

TOP Bộ đếm đạt giá trị TOP khi nó bằng với giá trị cao nhất trong chuỗi đếm, giá trị cao nhất trong chuỗi đếm không nhất thiết là FFh mà có thể là bất khì giá trị nào được qui định trong thanh ghi OCRn (n=0,2), tùy theo chế độ thực thi.

Bộ định thời 0 có vài đặc điểm chính như: Bộ đếm đơn kênh, xóa bộ định thời khi có sự kiện **so sánh khớp** (compare match) và tự nạp lại, có thể đếm từ bộ dao động 32 KHz bên ngoài, chế độ PWM hiệu chỉnh pha,…Dưới đây là mô tả chức năng của các thanh ghi liên quan tới bộ định thời 0.

1. Thanh Ghi Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0: Force Output Compare** (6)
- **Bit 6, 3 – WGM01:0: Waveform Generation Mode**
- **Bit 5:4 – COM01:0: Compare Match Output Mode**
- **Bit 2:0 – CS02:0: Clock Select**

• **Bit 7 – FOC0: Force Output Compare:** Bit này chỉ hoạt động khi các bit WGM chỉ định chế độ non-PWM (chẳn hạn chế độ CTC,…). Khi ở chế độ PWM nên ghi bit này thành 0. Ở chế độ non-PWM, khi bit FOC0 được ghi thành 1 lập tức một sự kiện "**so sánh khớp cưỡng bức**" (Force compare match) xảy ra ở bộ tạo sóng, tức là sự kiện so sánh khớp bị bắt buộc xảy ra mặt dù giá trị bộ định thời không bằng với giá trị ghi sẵn trong thanh ghi OCR0. Lúc này ngõ ra OC0 sẽ thay đổi tùy theo thiết lập của những bit COM01:0 tương ứng với nó. Bit FOC0 sẽ tự động xóa bởi phần cứng sau 1 chu kì clock. Bit này không thể đọc.

• **Bit 6, 3 – WGM01:0: Waveform Generation Mode** : Những bit này điều khiển các chế độ thực thi của bộ đếm, theo đó dạng sóng tương ứng được tạo ra từ bộ tạo sóng. Các chế độ thực thi được hỗ trợ là : Normal, CTC, PWM. Cụ thể xem **bảng 52.**

Table 52. Waveform Generation Mode Bit Description

Mode	WGM01(1) (CTC0)	WGM00(1) (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Bảng 52. Lựa chọn các chế độ thực thi của bộ định thời 0.

(1): Tên các bit CTC0 và PWM0 đã không được sử dụng nữa và được thay thế bằng các tên khác là WGM01 và WGM00. Khi lập trình nên chú ý điều này.

• **Bit 5:4 – COM01:0: Compare Match Output Mode :** Hai bit này điều khiển hành vi của chân OC0. Nếu một trong hai bit này được set thành 1 thì ngõ ra OC0 được ưu tiên hơn chức năng I/O thông thường. Chú ý là các bit tương ứng của OC0 trong thanh ghi DDR phải được set để cho phép ngõ ra. Khi bit OC0 được kết nối với chân ngõ ra OC0 thì

tác động của các bit COM01:0 đối với hành vi của chân OC0 còn phụ thuộc vào các thiết lập của các bit WGM01:0. Chi tiết xem **bảng 53, 54, 55.**

Chẳn hạn, khi ta set bit { WGM00, WGM01, COM00, COM01} = { 0, 0, 1, 0 } thì bộ định thời 0 sẽ hoạt động ở chế độ **Normal** và ở chế độ này hành vi của chân OC0 là: OC0 sẽ thay đổi mức logic mỗi khi có sự kiện So sánh khớp (Compare match). Để ý là ở chế độ **Normal**, với thiết lập các bit WGM00, WGM01, COM00, COM01như trên, giá trị thanh ghi OCR0 được cập nhật ngay tức thời, khác với ở chế độ PWM giá trị thanh ghi OCR0 chỉ được cập nhật khi bộ định thời đếm tới giá trị TOP (giả định trong đoạn chương trình ứng dụng có sự thay đổi giá trị thanh ghi OCR0).

Đoạn chương trình sau sẽ thiết lập bộ định thời hoạt động ở chế độ CTC và set chân OC0 lên 1 mỗi khi có sự kiện so sánh.

```
ldi  r17,0xFF       ; configured as output
out  DDRB,r17

ldi  r16,0xF0
out  OCR0,r16       ; match value

ldi  r16,0x39
out  TCCR0,r16      ; CTC mode
```

Table 53. Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Bảng 53. Điều khiển hành vi của chân OC0 bằng các bit COM00:1 trong chế độ non-PWM

Table 55. Compare Output Mode, Phase Correct PWM Mode(1)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

Bảng 55. Điều khiển hành vi của chân OC0 bằng các bit COM00:1 trong chế độ PWM hiệu chỉnh pha

Table 54. Compare Output Mode, Fast PWM Mode(1)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

Bảng 54. Điều khiển hành vi của chân OC0 bằng các bit COM00:1 trong chế độ PWM nhanh

Chú ý (1): Có trường hợp đặt biệt là khi thanh ghi OCR0 có giá trị là TOP và bit COM01 được set , trong trường hợp này việc so sánh khớp (Compare match) bị bỏ qua, nhưng việc set hay xóa OC0 ở TOP vẫn được thực hiện.

• **Bit 2:0 – CS02:0: Clock Select:** Đây là 3 bit dùng để lựa chọn xung clock cho bộ định thời. Xem **Bảng 56.** Để dừng bộ định thời ta chọn { CS00, CS01, CS02 } = {0, 0, 0 }.

Table 56. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0s}/256 (From prescaler)
1	1	1	clk_{T0s}/1024 (From prescaler)

Bảng 56. Lựa chọn tốc độ xung clock cho bộ định thời 0

2. Thanh Ghi Timer/Counter Register –TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Đây là thanh ghi đếm 8 bit của bộ định thời 0 .Giá trị thanh ghi này tăng hoặc giảm 1 đơn vị sau mỗi chu kì clock. Không nên ghi vào thanh ghi này khi nó đang đếm.

3. Thanh Ghi Output Compare Register –OCR0

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

OCR0 là thanh ghi 8 bit, giá trị của nó được liên tục so sánh với giá trị của thanh ghi TCNT0. Khi hai giá trị của hai thanh ghi này bằng nhau thì xảy ra một sự kiện "**so sánh khớp**" (compare match). Sự kiện so sánh khớp sẽ tạo ra một ngắt, nếu ngắt được cho phép. Hay tạo ra một dạng sóng ở chân ngõ ra OC0, tùy theo chế độ thực thi của bộ định thời.

4. Thanh Ghi Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**
- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable:** Khi bit OCIE0 ghi là 1 và bit I của thanh ghi trạng thái SREG được set thành 1 thì ngắt sự kiện "**so sánh khớp**" (compare match interrupt) được cho phép. Khi đó một ngắt sẽ được thực thi khi xảy ra một sự kiện "so sánh khớp".
- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable:** Khi bit này được ghi là 1 và ngắt toàn cục được cho phép thì ngắt tràn bộ định thời (Timer/Counter0 Overflow interrupt) được cho phép. Khi đó một ngắt tương ứng sẽ được thực thi khi bộ định thời tràn.

5. Thanh Ghi Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – OCF0: Output Compare Flag 0**
- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

- **Bit 1 – OCF0: Output Compare Flag 0:** Bit này sẽ được set lên 1 khi xảy ra "so sánh khớp " (compare match) giữa bộ định thời (tức thanh ghi TCCN0) với thanh ghi OCR0. Cờ OCF0 sẽ tự động xóa khi ngắt tương ứng được thực thi. Ngoài ra ta cũng có thể xóa cờ OCF0 bằng cách ghi một giá trị logic vào nó. Khi bit I trong thanh ghi SREG, bit OCIE0 (Timer/Counter0 Compare Match Interrupt Enable) và bit OCF0 được set lên 1 thì ngắt "so sánh khớp" (Compare Match Interrupt) sẽ được thực thi.
- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag:** Bit TOV0 được set thành 1 khi bộ định thời tràn và nó được xóa khi ngắt tương ứng được thực thi. Ngoài ra cũng có thể xóa bằng cách ghi một giá trị logic vào vị trí của nó . Khi bit I trong thanh ghi SREG, bit TOIE0 (Timer/Counter0 Overflow interrupt) và bit TOV0 được set lên 1 thì ngắt tràn bộ định thời 0 (Timer/Counter0 Overflow Interrupt) sẽ được thực thi. Trong chế độ PWM cờ TOV0 được Set khi bộ định thời 0 đổi hướng đếm tại giá trị 00h.

6. Thanh Ghi Special Function IO Register –SFIOR

Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – TSM: Timer/Counter Synchronization Mode**
- **Bit 1 – PSR0: Prescaler Reset Timer/Counter0**

- **Bit 7 – TSM: Timer/Counter Synchronization Mode:** Ghi bit này thành 1 sẽ kích hoạt chế độ đồng bộ bộ định thời (Timer/Counter Synchronization). Trong chế độ này, một giá trị được ghi vào các bit PSR0 và PSR321 sẽ được giữ lại, vì thế nó giữ cho tín hiệu reset của bộ chia trước (prescaler) tương ứng được xác nhận (do đó bộ chia trước (prescaler) vẫn ở trạng thái Reset). Điều này là để chắc chắn là các bộ định thời tương ứng sẽ được tạm nghỉ và có thể được cấu hình với các giá trị như nhau mà không làm ảnh hưởng đến một trong những cấu hình nâng cao khác của chúng. Khi bit này được ghi thành 0 thì các bộ định thời sẽ bắt đầu đếm đồng thời.
- **Bit 1 – PSR0: Prescaler Reset Timer/Counter0:** Khi bit này là 1 thì bộ chia trước của bộ định thời 0 (Timer/couter 0 prescaler) sẽ được đặt lại. Bit này thường được xóa tức thời bởi phần cứng. Nếu bit này được ghi khi bộ định thời 0 đang thực thi chế độ không đồng bộ thì nó vẫn giữ nguyên giá trị của nó cho đến khi bộ chia trước được đặt lại. Bit này sẽ không được xóa bởi phần cứng nếu như bit TSM được set thành 1.

7. Thanh Ghi Asynchronous Status Register – ASSR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	AS0	TCN0UB	OCR0UB	TCR0UB	ASSR
Read/Write	R	R	R	R	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3 – AS0: Asynchronous Timer/Counter0**
- **Bit 2 – TCN0UB: Timer/Counter0 Update Busy**
- **Bit 1 – OCR0UB: Output Compare Register0 Update Busy**
- **Bit 0 – TCR0UB: Timer/Counter Control Register0 Update Busy**

- **Bit 3 – AS0: Asynchronous Timer/Counter0:** Khi bit AS0 là 0 thì bộ định thời được đếm từ nguồn xung clock I/O, tức $Clk_{I/O}$. Khi AS0 được ghi thành 1 bộ định thời

được đếm từ xung thạch anh ở chân TOSC1. Khi giá trị của AS0 bị thay đổi thì nội dụng của các thanh ghi TCNT0, OCR0 và TCCR0 có thể bị hỏng.

• **Bit 2 – TCN0UB: Timer/Counter0 Update Busy:** Khi bộ định thời 0 thực thi quá trình không đồng bộ và thanh ghi TCNT0 đang được ghi thì bit TCN0UB sẽ set lên 1. Khi thanh ghi TCNT0 vừa được cập nhật từ thanh ghi lưu trữ tạm thì bit này bị xóa bởi phần cứng. Mức logic 0 trong trường hợp này là để chỉ ra rằng thanh ghi TCNT0 đã sẵn sàng để cập nhật một giá trị mới.

• **Bit 1 – OCR0UB: Output Compare Register0 Update Busy:** Khi bộ định thời 0 thực thi quá trình không đồng bộ và thanh ghi OCR0 đang được ghi thì bit OCR0UB sẽ set lên 1. Khi thanh ghi OCR0 vừa được cập nhật từ thanh ghi lưu trữ tạm thì bit này bị xóa bởi phần cứng .Mức logic 0 trong trường hợp này là để chỉ ra rằng thanh ghi OCR0 đã sẵn sàng để cập nhật một giá trị mới.

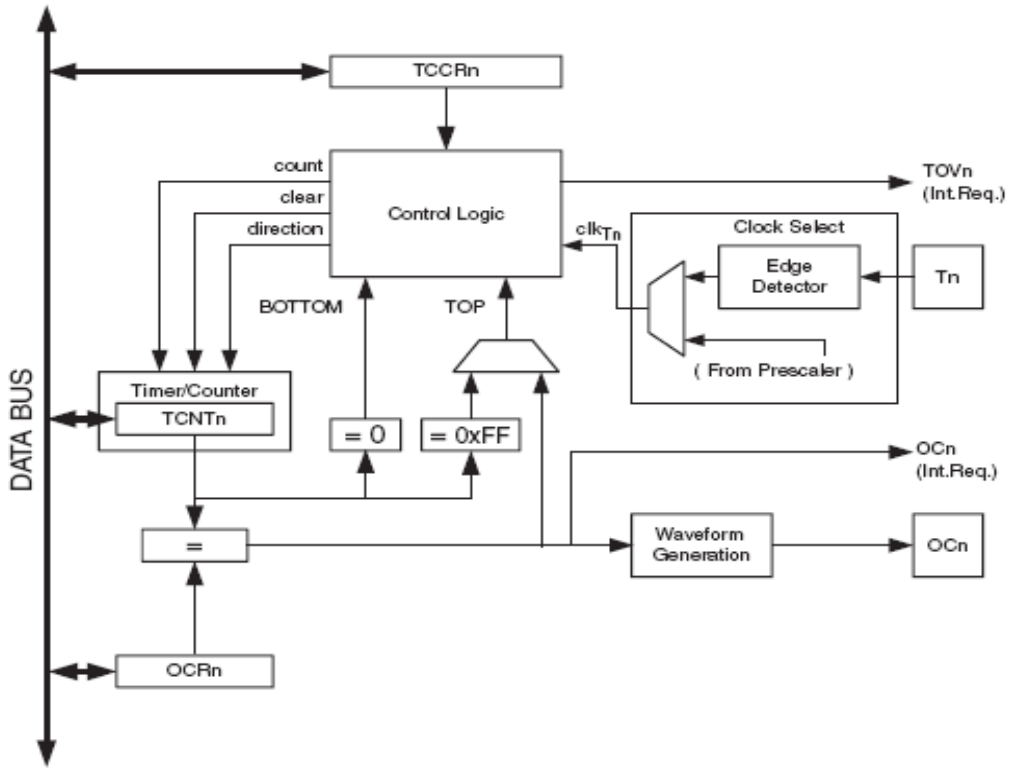
• **Bit 0 – TCR0UB: Timer/Counter Control Register0 Update Busy:** Khi bộ định thời 0 thực thi quá trình không đồng bộ và thanh ghi TCCR0 đang được ghi thì bit TCR0UB sẽ set lên 1. Khi thanh ghi TCCR0 vừa được cập nhật từ thanh ghi lưu trữ tạm thì bit này bị xóa bởi phần cứng. Mức logic 0 trong trường hợp này là để chỉ ra rằng thanh ghi TCCR0 đã sẵn sàng để cập nhật một giá trị mới.

Nếu ghi vào một trong ba thanh ghi của bộ định thời 0 (TCNT0, OCR0, TCCR0) trong lúc cờ báo bận cập nhật (update busy flag) của chúng được set, thì giá trị cập nhật có thể bị hỏng và sẽ tạo ra một ngắt không biết trước.

IV. BỘ ĐỊNH THỜI 2

Sơ đồ khối bộ định thời 2

Figure 61. 8-Bit Timer/Counter Block Diagram



Bộ định thời 2 là bộ định thời 8 bit, bộ định thời 2 liên quan tới 5 thanh ghi với nhiều chế độ thực thi khác nhau. Các thuộc tính chính của bộ định gồm: Bộ đếm đơn kênh, xóa bộ định thời khi có sự kiện "so sánh khớp" và tự động nạp lại, PWM hiệu chỉnh pha, đếm sự kiện bên ngoài…

Các Thanh Ghi Bộ Định Thời 2.

1. Thanh ghi Timer/Counter Control Register – TCCR2

Bit	7	6	5	4	3	2	1	0	
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC2: Force Output Compare**
- **Bit 6, 3 – WGM21:0: Waveform Generation Mode**

- **Bit 5:4 – COM21:0: Compare Match Output Mode**
- **Bit 2:0 – CS22:0: Clock Select**

• **Bit 7 – FOC2: Force Output Compare :** Bit FOC2 chỉ hoạt động khi bit WGM20 chỉ định chế độ Non-PWM, trong chế PWM nên ghi bit này thành 0. Ở chế độ non-PWM, khi bit FOC0 được ghi thành 1 lập tức một "**so sánh khớp**" (compare match) xảy ra ở bộ tạo sóng, ngõ ra OC2 thay đổi tùy theo thiết lập của những bit COM21:0 tương ứng với nó. Bit này không thể đọc, khi đọc ta luôn nhận giá trị 0. Bit này hoạt động giống như bit FOC0 của bộ định thời 0.

• **Bit 6, 3 – WGM21:0: Waveform Generation Mode** : Những bit này điều khiển các chế độ thực thi của bộ đếm, theo đó dạng sóng tương ứng được tạo ra từ bộ tạo sóng. Các chế độ thực thi được hỗ trợ là : Normal, CTC, PWM. Xem **bảng 64**.

Table 64. Waveform Generation Mode Bit Description

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation	TOP	Update of OCR2 at	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Bảng 64. Lựa chọn các chế độ thực thi của bộ định thời 2

***Chú ý** : Tên các bit CTC2 và PWM2 đã không được sử dụng nữa và được thay thế bằng các tên khác là WGM21 và WGM20.*

• **Bit 5:4 – COM21:0: Compare Match Output Mode:** Hai bit này điều khiển hoạt động của chân OC2. Nếu một trong hai bit này được set thành 1 thì ngõ ra OC2 được ưu tiên hơn chức năng I/O thông thường . Chú ý là các bit tương ứng của OC2 trong thanh ghi DDR phải được set để cho phép ngõ ra. Khi OC2 được kết nối với chân ngõ ra OC2 thì vai trò của các bit COM21:0 còn phụ thuộc vào các thiết lập của các bit WGM21:0. Chi tiết xem **bảng 65, 66, 67**. Các bit này hoạt động giống với các bit COM01:0 của bộ định thời 0. Xem lại bộ định thời 0.

Table 65. Compare Output Mode, Non-PWM Mode

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Toggle OC2 on compare match
1	0	Clear OC2 on compare match
1	1	Set OC2 on compare match

Bảng 65. Điều khiển hành vi của chân OC2 bằng các bit COM20:1 trong chế độ non-PWM

Table 66. Compare Output Mode, Fast PWM Mode(1)

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match, set OC2 at TOP
1	1	Set OC2 on compare match, clear OC2 at TOP

Bảng 66. Điều khiển hành vi của chân OC2 bằng các bit COM20:1 trong chế độ PWM nhanh

Table 67. Compare Output Mode, Phase Correct PWM Mode(1)

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match when up-counting. Set OC2 on compare match when downcounting.
1	1	Set OC2 on compare match when up-counting. Clear OC2 on compare match when downcounting.

Bảng 67. Điều khiển hành vi của chân OC2 bằng các bit COM20:1 trong chế độ PWM hiệu chỉnh pha

***Chú ý**: Có trường hợp đặt biệt là khi thanh ghi OCR2 có giá trị là TOP và bit COM21 được set , trong trường hợp này sự kiện so sánh khớp (Compare match) bị bỏ qua, nhưng việc set hay xóa OC2 ở TOP vẫn được thực hiện.*

• **Bit 2:0 – CS22:0: Clock Select:** Dùng để lựa chọn nguồn xung clock cho bộ định thời 2. Xem chi tiết **bảng 68.**

Table 68. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T2 pin. Clock on falling edge
1	1	1	External clock source on T2 pin. Clock on rising edge

Bảng 68. Lựa chọn tốc độ xung clock cho bộ định thời 0

***Để ý**: Bộ định thời 2 có thể được đếm từ nguồn clock bên ngoài thông qua chân T2. Khi chuyển sang nguồn clock ngoài, bộ đếm vẫn đếm bình thường ngay cả khi chân T2 được cấu hình là ngõ ra.*

2. Thanh ghi Timer/Counter Register – TCNT2

Bit	7	6	5	4	3	2	1	0	
	TCNT2[7:0]								TCNT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Đây là thanh ghi đếm 8 bit của bộ định thời 2. Giá trị thanh ghi này tăng hoặc giảm 1 đơn vị sau mỗi chu khì clock. Thanh ghi TCNT2 được truy xuất trực tiếp khi đọc hay ghi (Điều này khác với bộ định thời 1 và 3 là khi truy xuất các thanh ghi TCNT1 hay TCNT3 cần phải thông qua thanh ghi tạm trung gian 8 bit). Không nên chỉnh sửa thanh ghi TCNT2 khi bộ định thời đang chạy.

3. Thanh ghi Output Compare Register – OCR2

Bit	7	6	5	4	3	2	1	0	
	OCR2[7:0]								OCR2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi OCR2 là thanh ghi 8 bit, giá trị của thanh ghi OCR2 sẽ được liên tục so sánh với giá trị của bộ đếm, tức thanh ghi TCNT2. Khi giá trị của hai thanh ghi này bằng nhau sẽ tạo ra sự kiện "so sánh khớp" (Compare match). Một **ngắt so sánh khớp** (compare match interrupt) có thể được tạo ra nếu ngắt được cho phép, hay một dạng sóng sẽ được tạo ra ở chân OC2.Thanh ghi này hoạt động tương tự như thanh ghi OCR0 ở bộ định thời 0.

4. Thanh ghi Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – OCIE2: Timer/Counter2 Output Compare Match Interrupt Enable**
- **Bit 6 – TOIE2: Timer/Counter2 Overflow Interrupt Enable**

• **Bit 7 – OCIE2: Timer/Counter2 Output Compare Match Interrupt Enable:** Khi bit OCIE2 được set thành 1 và bit I trong thanh ghi trạng thái được set thành 1 thì ngắt "so sánh khớp" (compare match interrupt) của bộ định thời 2 được cho phép. Khi đó môt ngắt tương ứng sẽ được thực thi khi xảy ra một sự kiện "so sánh khớp" ở bộ định thời 2. Chẳn hạn, để xảy ra một "so sánh khớp" (compare match) ở bộ định thời 2 ta có thể set bit OCF2 trong thanh ghi TIFR, hoặc là chờ cho đến khi nào giá trị của hai thanh ghi TCNT2 và OCR2 bằng nhau thì một "so sánh khớp" (compare match) sẽ xảy ra.

• **Bit 6 – TOIE2: Timer/Counter2 Overflow Interrupt Enable:** Khi bit này được ghi là 1 và ngắt toàn cục được cho phép (bit I trong thanh ghi trạng thái SREG được set thành 1) thì ngắt tràn bộ định thời 2 (Timer/Counter2 Overflow interrupt) được cho phép. Khi đó một ngắt tương ứng sẽ được thực thi khi bộ định thời 2 tràn. Chẳn hạn, ta set bit TOV2 trong thanh ghi TIFR thành 1 hoặc là chờ cho bộ định thời 2 bị tràn khi vượt quá giá trị TOP (hay MAX).

5. Thanh ghi Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – OCF2: Output Compare Flag 2**
- **Bit 6 – TOV2: Timer/Counter2 Overflow Flag**

• **Bit 7 – OCF2: Output Compare Flag 2** : Bit này sẽ được set lên 1 khi xảy ra sự kiện "so sánh khớp" (compare match) giữa bộ định thời 2 (tức thanh ghi TCCN2) với thanh ghi OCR2. Cờ OCF2 sẽ tự động xóa khi ngắt tương ứng được thực thi. Ngoài ra ta cũng có thể xóa cờ OCF2 bằng cách ghi một giá trị logic vào nó. Khi bit I trong thanh ghi SREG, bit OCIE2 (Timer/Counter2 Compare Match Interrupt Enable) và bit OCF2 được set lên 1 thì ngắt sự kiện "so sánh khớp" (Compare Match Interrupt) của bộ định thời 2 sẽ được thực thi.

• **Bit 6 – TOV2: Timer/Counter2 Overflow Flag**: Bit TOV2 được set thành 1 khi bộ định thời tràn và nó được xóa khi ngắt tương ứng được thực thi. Ngoài ra cũng có thể xóa bằng cách ghi một giá trị logic vào vị trí của nó. Khi bit I trong thanh ghi SREG, bit TOIE2 (Timer/Counter2 Overflow interrupt) và bit TOV2 được set lên 1 thì ngắt tràn bộ định thời 2 (Timer/Counter2 Overflow Interrupt) sẽ được thực thi. Trong chế độ PWM cờ TOV2 được set khi bộ định thời 2 đổi hướng đếm tại giá trị 00h.

Chương IV

CẤU TRÚC NGẮT CỦA ATmega128

I. KHÁI NIỆM VỀ NGẮT

Ngắt là một sự kiện bên trong hay bên ngoài làm ngắt bộ vi điều khiển để báo cho nó biết rằng thiết bị cần dịch vụ của nó.

Một bộ vi điều khiển có thể phục vụ một vài thiết bị, có hai cách để thực hiện điều này đó là sử dụng các ngắt (interrupt) và thăm dò (polling). Trong phương pháp sử dụng các ngắt thì mỗi khi có một thiết bị bất kỳ cần đến dịch vụ của nó thì nó báo cho bộ vi điều khiển bằng cách gửi một tín hiệu ngắt. Khi nhận được tín hiệu ngắt thì bộ vi điều khiển ngắt tất cả những gì nó đang thực hiện để chuyển sang phục vụ thiết bị. Chương trình đi cùng với ngắt được gọi là trình dịch vụ ngắt ISR (Interrupt Service Routine) hay còn gọi là trình quản lý ngắt (Interrupt handler). Còn trong phương pháp thăm dò thì bộ vi điều khiển hiển thị liên tục tình trạng của một thiết bị đã cho và điều kiện thoả mãn thì nó phục vụ thiết bị. Sau đó nó chuyển sang hiển thị tình trạng của thiết bị kế tiếp cho đến khi tất cả đều được phục vụ.

Mặc dù phương pháp thăm dò có thể thể hiển thị tình trạng của một vài thiết bị và phục vụ mỗi thiết bị khi các điều kiện nhất định được thoả mãn nhưng nó không tận dụng hết cộng dụng của bộ vi điều khiển. Điểm mạnh của phương pháp ngắt là bộ vi điều khiển có thể phục vụ được rất nhiều thiết bị (tất nhiên là không tại cùng một thời điểm). Mỗi thiết bị có thể nhận được sự chú ý của bộ vi điều khiển dựa trên mức ưu tiên được gán cho nó. Đối với phương pháp thăm dò thì không thể gán mức ưu tiên cho các thiết bị vì nó kiểm tra tất cả mọi thiết bị theo kiểu quay vòng . Quan trọng hơn là trong phương pháp ngắt thì bộ vi điều khiển cũng còn có thể che hoặc làm lơ một yêu cầu dịch vụ của thiết bị. Điều này lại một lần nữa không thể thực hiện được trong phương pháp thăm dò. Lý do quan trọng nhất mà phương pháp ngắt được ưu chuộng nhất là vì phương pháp thăm dò làm hao phí thời gian của bộ vi điều khiển bằng cách hỏi dò từng thiết bị kể cả khi chúng không cần đến dịch vụ.

II. TRÌNH PHỤC VỤ NGẮT VÀ BẢNG VECTOR NGẮT

Đối với mỗi ngắt thì phải có một trình phục vụ ngắt ISR (Interrupt Service Routine) hay trình quản lý ngắt (Interrupt handler). Khi một ngắt được gọi thì bộ vi điều khiển phục vụ ngắt. Khi một ngắt được gọi thì bộ vi điều khiển chạy trình phục vụ ngắt. Đối với mỗi ngắt thì có một vị trí cố định trong bộ nhớ để giữ địa chỉ ISR của nó. Nhóm các vị trí nhớ được dành riêng để gửi các địa chỉ của các ISR được gọi là **bảng véc tơ ngắt**.

Khi kích hoạt một ngắt bộ vi điều khiển đi qua các bước sau:

● Vi điều khiển kết thúc lệnh đang thực hiện và lưu địa chỉ của lệnh kế tiếp (PC) vào ngăn xếp.

● Nó nhảy đến một vị trí cố định trong bộ nhớ được gọi là bảng véc tơ ngắt nơi lưu giữ địa chỉ của một trình phục vụ ngắt.

● Bộ vi điều khiển nhận địa chỉ ISR từ bảng véc tơ ngắt và nhảy tới đó. Nó bắt đầu thực hiện trình phục vụ ngắt cho đến lệnh cuối cùng của ISR là RETI (trở về từ ngắt).

● Khi thực hiện lệnh RETI bộ vi điều khiển quay trở về nơi nó đã bị ngắt. Trước hết nó nhận địa chỉ của bộ đếm chương trình PC từ ngăn xếp bằng cách kéo hai byte trên đỉnh của ngăn xếp vào PC. Sau đó bắt đầu thực hiện các lệnh từ địa chỉ đó.

III. BẢNG VECTOR NGẮT CỦA ATmega128.

Dưới đây là bảng véc tơ ngắt của ATmega128 , cùng với địa chỉ của nó trong bộ nhớ chương trình (**bảng 23**).

Table 23. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	$001C	TIMER1 OVF	Timer/Counter1 Overflow

16	$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	$0022	SPI, STC	SPI Serial Transfer Complete
19	$0024	USART0, RX	USART0, Rx Complete
20	$0026	USART0, UDRE	USART0 Data Register Empty
21	$0028	USART0, TX	USART0, Tx Complete
22	$002A	ADC	ADC Conversion Complete
23	$002C	EE READY	EEPROM Ready
24	$002E	ANALOG COMP	Analog Comparator
25	$0030(3)	TIMER1 COMPC	Timer/Countre1 Compare Match C
26	$0032(3)	TIMER3 CAPT	Timer/Counter3 Capture Event
27	$0034(3)	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	$0036(3)	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	$0038(3)	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	$003A(3)	TIMER3 OVF	Timer/Counter3 Overflow
31	$003C(3)	USART1, RX	USART1, Rx Complete
32	$003E(3)	USART1, UDRE	USART1 Data Register Empty
33	$0040(3)	USART1, TX	USART1, Tx Complete
34	$0042(3)	TWI	Two-wire Serial Interface
35	$0044(3)	SPM READY	Store Program Memory Ready

Bảng 23. Bảng Vector Ngắt Của ATmega128

IV. THỨ TỰ ƯU TIÊN NGẮT.

Không như vi điều khiển họ 8051, ở đó thứ tự ưu tiên của các ngắt có thể thay đổi được (bằng cách lập trình). Với vi điều khiển AVR thứ tự ưu tiên các ngắt là không thể thay đổi và theo qui tắc: " ***Một vec tơ ngắt có địa chỉ thấp hơn trong bộ nhớ chương trình có mức độ ưu tiên cao hơn*** ". Chẳn hạn ngắt ngoài 0 (INT0) có mức độ ưu tiên cao hơn ngắt ngoài 1 (INT1).

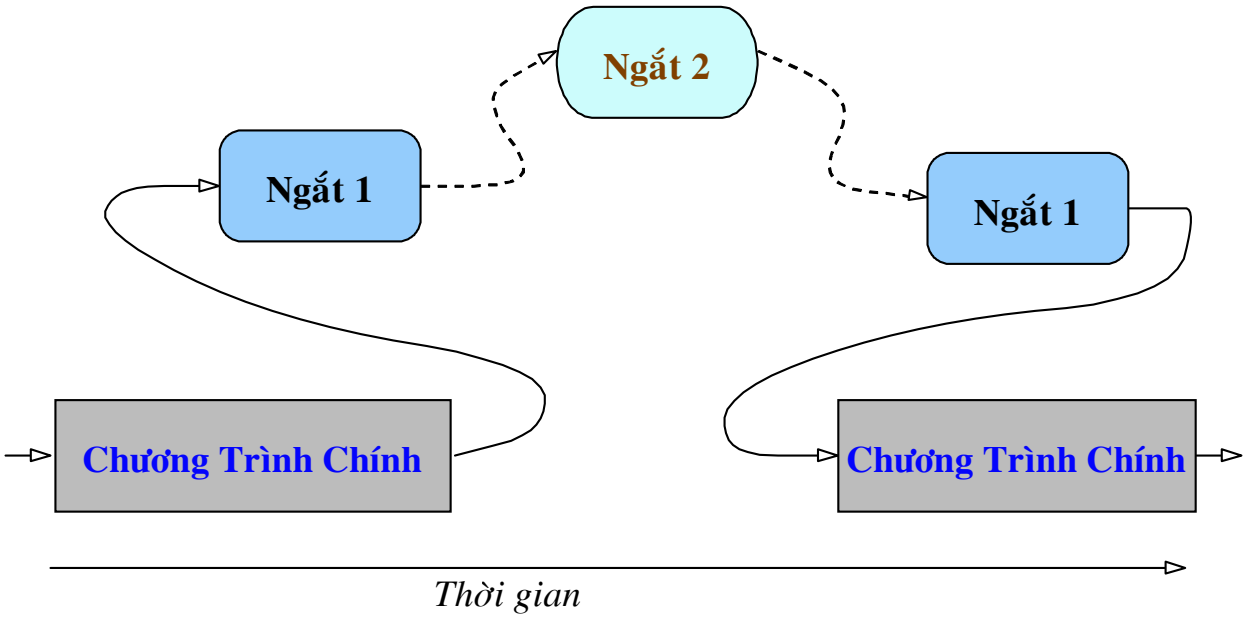
Để cho phép một ngắt người dùng cần cho phép ngắt toàn cục (set bit I trong thanh ghi SREG) và các bit điều khiển ngắt tương ứng.

Khi một ngắt xảy ra và đang được phục vụ thì bit I trong thanh ghi SREG bị xóa, như thế khi có một ngắt khác xảy ra nó sẽ không được phục vụ, do đó để cho phép các ngắt

trong khi một ISR (interrupt service routine) khác đang thực thi, thì trong chương trình ISR phải có lệnh SEI để set lại bit I trong SREG.

V. NGẮT TRONG NGẮT.

Khi AVR đang thực hiện một trình phục vụ ngắt thuộc một ngắt nào đó thì lại có một ngắt khác được kích hoạt. Trong những trường hợp như vậy thì một ngắt có mức ưu tiên cao hơn có thể ngắt một ngắt có mức ưu tiên thấp hơn. Lúc này ISR của ngắt có mức ưu tiên cao hơn sẽ được thực thi (*). Khi thực hiện xong ISR của ngắt có mức ưu tiên cao hơn thì nó mới quay lại phục vụ tiếp ISR của ngắt có mức ưu tiên thấp hơn trước khi trở về chương trình chính. Đây gọi là ngắt trong ngắt. (**hình 4.1**).



Hình 4.1. Các ngắt lồng nhau

Chú ý:

- Giả định là khi một ISR nào đó đang thực thi thì xảy ra một yêu cầu ngắt từ một ISR khác có mức ưu tiên thấp hơn thì ISR có mức ưu tiên thấp hơn không được phục vụ, nhưng nó sẽ không bị bỏ qua luôn mà ở trạng thái chờ. Nghĩa là ngay sau khi ISR có mức ưu tiên cao hơn thực thi xong thì đến lượt ISR có mức ưu tiên thấp hơn sẽ được phục vụ.
- **(*)** : *Điều này chỉ xảy ra khi trong code của ISR của ngắt có mức ưu tiên thấp hơn có lệnh set bit I trong thanh ghi SREG* (*đó là lệnh* **SEI**).

VI. CÁC NGẮT NGOÀI.

ATmega128 có 8 ngắt ngoài từ INT0 đến INT7 (ở đây chưa kể tới ngắt reset). Tám ngắt này tương ứng với 8 chân của MCU là INT0 ,INT1, …, INT7. Để ý là ngay cả khi các chân INT0, INT1, …, INT7 của MCU được cấu hình như là chân lối ra, thì các ngắt ngoài vẫn có tác dụng nếu được cho phép.

Các ngắt ngoài có thể bắt mẫu theo kiểu cạnh lên (Rising), cạnh xuống (Falling) hay mức thấp (Low level). Điều này được qui định trong hai thanh ghi EICRA và EICRB. Dưới đây là mô tả chi tiết 2 thanh ghi EICRA và EICRB và các thanh ghi liên quan tới các ngắt ngoài.

1. Thanh ghi External Interrupt Control Register A – EICRA

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bits 7..0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits

Tám bit của thanh ghi EICRA sẽ điều khiển kiểu bắt mẫu cho 4 ngắt INT3, INT2, INT1, INT0. Qui định cụ thể được thể hiện trong **Bảng 48** .

ISCn1	ISCn0	Kiểu bắt mẫu
0	0	Mức thấp sẽ tạo yêu cầu ngắt
0	1	Dự trữ
1	0	Cạnh xuống (Falling) sẽ tạo yêu cầu ngắt
1	1	Cạnh lên (Rising) sẽ tạo yêu cầu ngắt

n = 3, 2, 1, 0

Bảng 48 . Điều khiển kiểu bắt mẫu ngắt

2. Thanh Ghi External Interrupt Control Register B – EICRB

Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bits 7..0 – ISC71, ISC70 - ISC41, ISC40: External Interrupt 7 - 4 Sense Control Bits.**

Tám bit của thanh ghi EICRA sẽ điều khiển kiểu bắt mẫu cho 4 ngắt INT7, INT6, INT5, INT4. . Qui định cụ thể được thể hiện trong Bảng 50 .

ISCn1	ISCn0	Kiểu bắt mẫu
0	0	Mức thấp sẽ tạo yêu cầu ngắt
0	1	Bất cứ sự thay đổi mức logic nào ở chân INTn sẽ tao ra một yêu cầu ngắt
1	0	Cạnh xuống (Falling) giữa hai mẫu sẽ tạo yêu cầu ngắt
1	1	Cạnh lên (Rising) giữa hai mẫu sẽ tạo yêu cầu ngắt

n = 7,6, 5, 4

Bảng 50 . Điều khiển kiểu bắt mẫu ngắt

3. Thanh Ghi External Interrupt Mask Register – EIMSK

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bits 7..0 – INT7 – INT0: External Interrupt Request 7 - 0 Enable :** Khi cho phép ngắt toàn cục (set bit I trong thanh ghi SREG thành 1) thì các ngắt vẫn chưa thể thực thi, để ngắt có thể thực thi ta cần phải cho phép nó, 8 bit trong thanh ghi EIMSK sẽ quyết định 8 ngắt ngoài tương ứng (từ INT7 ...INT0) có được cho phép hay không. Khi một trong số 8 bit (từ INT7 ...INT0) được set thành 1 và ngắt toàn cục được cho phép thì ngắt ngoài tương ứng được cho phép. Còn tín hiệu ngắt là mức hay cạnh sẽ do các thanh ghi

EICRA và EICRB (nêu ở trên) qui định. Kích hoạt bất cứ chân (Pin) nào trong 8 chân của ngắt ngoài cũng tạo ra yêu cầu ngắt ngay cả khi chân được thiết lập thành ngõ ra.

4. Thanh Ghi External Interrupt Flag Register – EIFR

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bits 7..0 – INTF7 - INTF0: External Interrupt Flags 7 – 0 :** Đây là tám cờ ngắt tương ứng với tám ngắt ngoài INT7..INT0. Khi có tín hiệu yêu cầu ngắt ngoài thì cờ ngắt tương ứng sẽ được set thành 1, nếu ngắt tương ứng được cho phép thì MCU sẽ nhảy tới bảng véc tơ ngắt, cờ ngắt sẽ được xóa khi chương trình phục vụ ngắt (ISR) được thực thi. Ngoài ra ta cũng có set hay xóa cờ ngắt bằng cách ghi trực tiếp một giá trị logic vào nó.

5. Thanh Ghi MCU Control Register – MCUCR

Bit	7	6	5	4	3	2	1	0	
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Trong phần này ta chỉ quan tâm tới hai bit là: IVCE (Interrupt Vector Select) và bit IVSEL (Interrupt Vector Change Enable) của thanh ghi MCUCR. Bit này liên quan đến việc thiết lập vị trí bảng véc tơ ngắt.

• **Bit 1 – IVSEL: Interrupt Vector Select:** Khi bit này là 0 vị trí của bảng véc tơ ngắt được đặt ở phần đầu bộ nhớ chương trình. Khi bit này là 1 bảng véc tơ ngắt được di chuyển tới phần đầu của vùng nhớ Boot Loader.

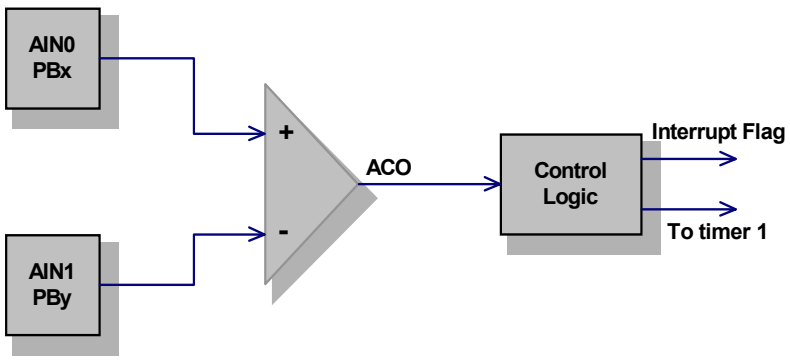
• **Bit 0 – IVCE: Interrupt Vector Change Enable** : Bit này phải được ghi thành 1 để cho phép thay đổi bit IVSEL. Bit IVCE được xóa sau 4 chu kì máy sau khi nó được set hay bit IVSEL được ghi. Trong lúc bit ICVE đang set các ngắt sẽ bị cấm cho tới khi bit IVSEL được ghi, nếu bit IVSEL không được ghi thì các ngắt vẫn bị cấm trong 4 cho kì máy liên tiếp (sau 4 chu kì máy thì bit IVCE sẽ tự động bị xóa nên các ngắt được cho phép trở lại).

Chương V
CÁC BỘ PHẬN NGOẠI VI KHÁC

Ngoài các bộ phận ngoại vi đã được giới thiệu ở các chương trước như : Bộ định thời, các cổng vào ra, EEPROM … Vi điều khiển ATmega128 có có nhiều bộ phận ngoại vi khác, các bộ ngoại vi này rất tiện lợi trong các ứng dụng điều khiển (bộ PWM), xử lí số liệu (bộ ADC, bộ so sánh Analog), giao tiếp (bộ USART, SPI, I_2C)… Việc tích hợp các bộ ngoại vi này vào trong chip giúp cho các thiết kế trở nên thuận tiện hơn, kích thước bo mạch cũng gọn gàng hơn ...

I. BỘ SO SÁNH TƯƠNG TỰ

Sơ đồ đơn giản của bộ so sánh tương tự (Analog Comparator) như hình 5.1. Bộ so sánh có hai ngõ vào tương tự là AIN0 và AIN1 và một ngõ ra số ACO. Nguyên tắc hoạt động của bộ so sánh tương tự là : **Khi ngõ vào AIN0 có điện thế cao hơn ngõ vào AIN1 thì ngõ ra ACO sẽ ở mức cao (tương ứng với logic 1), ngược lại khi ngõ vào AIN0 có điện thế thấp hơn ngõ vào AIN1 thì ngõ ra ACO sẽ ở mức thấp (tương ứng với logic 0)**. Thường thì trong hai ngõ vào, có một ngõ vào có điện thế được giữ cố định để dùng làm điện thế tham chiếu (V_{Ref}), điện thế ngõ còn lại có thể thay đổi để tham chiếu với ngõ vào V_{Ref} . Trạng thái của ngõ ra ACO của bộ so sánh có thể được dùng để tạo ra một ngắt, kết nối tới bộ định thời 1 để sử dụng chức năng input capture của bộ định thời này (xem mô tả sau).



Hình 5.1. Sơ đồ giản lượt của bộ so sánh tương tự

Cần chú ý là có sự khác biệt về chi tiết **ở bộ so sánh tương tự** đối với các dòng AVR khác nhau, chẳng hạn bộ so sánh tương tự của AT90S8535 hơi khác với bộ so sánh tương tự ở ATmega128, tuy nhiên cấu trúc cơ bản thì vẫn như nhau. Sau đây là mô tả cụ thể về bộ so sánh tương tự của ATmega128.

Ở hình 5.1 ta thấy hai ngõ vào AIN0 và AIN1 tương ứng với hai chân PBx và PBy (x = 2, y = 3 đối với AT90S8535), ở ATmega128 ta có nhiều lựa chọn ngõ vào hơn, các thanh ghi liên quan sẽ giúp ta thết lập các lựa chọn này.

1. Thanh ghi Special Function IO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 3 – ACME: Analog Comparator Multiplexer Enable

Ở thanh ghi này ta chỉ sử dụng bit Bit 3 – ACME , khi bit này là 1 và chức năng ADC không cho phép hoạt động (bit ADEN trong thanh ghi ADCSRA là 0) thì ngõ vào âm của bộ so sánh tương tự có thể là 1 trong số 8 ngõ vào ADC tùy theo thiết lập của các bit MUX 2, MUX 1, MUX 0 (xem **bảng 94**), chẳng hạn nếu { ACME, ADEN, MUX 2, MUX 1, MUX 0 } = { 1, 0, 0, 0, 0 } thì ngõ ADC0 (tương ứng với chân số 61 của vi điều khiển) được chọn làm ngõ vào âm. Nếu bit ACME là 0 thì ngõ vào âm của bộ so sánh tương tự là AIN1 (tương ứng với chân số 5 của vi điều khiển).

Table 94. Analog Comparator Multiplexed Input

ACME	ADEN	MUX2..0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

Bảng 94. Lựa chọn lối vào âm

2. Thanh ghi Analog Comparator Control and Status Register – ACSR

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: Analog Comparator Disable**
- **Bit 6 – ACBG: Analog Comparator Bandgap Select**
- **Bit 5 – ACO: Analog Comparator Output**
- **Bit 4 – ACI: Analog Comparator Interrupt Flag**
- **Bit 3 – ACIE: Analog Comparator Interrupt Enable**
- **Bit 2 – ACIC: Analog Comparator Input Capture Enable**
- **Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select**

• **Bit 7 – ACD**: **Analog Comparator Disable** : Khi bit này là 1 sẽ không cho phép bộ so sánh tương tự hoạt động. Khi bit này là 0 bộ so sánh tương tự được phép hoạt động. Ta có thể thay đổi bit này bất cứ lúc nào để cho phép hay không cho phép bộ so sánh tương tự hoạt động. Nhưng cần chú ý là bất cứ sự thay đổi nào của bit ACD cũng có thể tạo ra một ngắt (ngắt của bộ so sánh tương tự), do đó nếu không cần thiết ta nên cấm ngắt của bộ so sánh tương tự bằng cách xóa bit ACIE của thanh ghi ACSR.

• **Bit 6 – ACBG**: **Analog Comparator Bandgap Select** : Khi bit này là 1 ngõ vào dương sẽ được giữ ở mức điện thế cố định khoảng 1,23 V (ở 25^0C và V_{cc} = 5 V) và được dùng làm điện thế tham chiếu, gọi là điện thế tham chiếu nội (Internal voltage reference). Như vậy, trong trường hợp này ngõ vào âm sẽ thay đổi giá trị và tham chiếu tới giá trị 1,23 V. Chú ý là khi ta sử dụng điện thế tham chiếu nội 1,23 V như đã nêu trên thì ta cần thiết lập bit ACBG thành 1 trước khi cho phép bộ so sánh tương tự hoạt động, bởi vì khi điện thế tham chiếu nội được cho phép nó cần một khoảng thời gian khởi động là 40 µs để có thể ổn định ở điện thế 1,23 V. Khi bit này là 0 chân AIN0 (tương ứng với chân số 4 của vi điều khiển) trở thành ngõ vào dương.

• **Bit 5 – ACO: Analog Comparator Output** : Bit này chính là trạng thái ở ngõ ra của bộ so sánh, đọc bit này ta sẽ biết được trạng thái hiện thời của ngõ vào. Khi tương quan so sánh ở hai ngõ vào thay đổi, cần từ 1 tới 2 chu kì máy để phản ánh kết quả này ở ngõ ra ACO.

• **Bit 4 – ACI: Analog Comparator Interrupt Flag** : Đây là bit cờ ngắt của bộ so sánh tương tự, khi xảy ra ngắt ở bộ so sánh tương tự bit này sẽ được set lên 1 bởi phần cứng, trình phục vụ ngắt được thực thi nếu ngắt được cho phép (bằng cách set bit ACIE trong ghi ACSR và bit I trong thanh ghi SREG). Véc tơ ngắt của bộ so sánh tương tự có địa chỉ là $002E. Bit ACI sẽ được tự động xóa bởi phần cứng khi trình phục vụ ngắt được thực thi. **Chú ý : Bit ACI sẽ tự động xóa khi có bất cứ sự thay đổi nào của thanh ghi ACSR. Chẳn hạn khi ta ghi vào bit này giá trị logic 1 thì sau khi thực hiện xong lệnh**

ghi ta vẫn nhận được giá trị logic 0 ở bit này . Do đó ta không thể nào set được bit này bằng phần mềm.

• Bit 3 – ACIE: Analog Comparator Interrupt Enable : Đây là bit cho phép ngắt của bộ so sánh tương tự. Khi bit này là 1 thì ngắt bộ so sánh tương tự được cho phép. Ngược lại, khi bit này là 0 thì ngắt bộ so sánh tương tự bị cấm.

• Bit 2 – ACIC: Analog Comparator Input Capture Enable : bit này liên quan tới tính năng input capture của bộ định thời 1 (xem lại bộ định thời 1). Khi bit này là 1 ngõ ra của bộ so sánh được nối trực tiếp tới lối vào của khối input capture của bộ định thời 1, nhờ cách này ta có thể tận dụng tính năng khử nhiễu ở ngõ vào input capture của bộ định thời 1, trong cách thiết lập này ngắt input capture vẫn có thể hoạt động nếu được cho phép (bằng cách cho phép ngắt toàn cục và set bit TICIE1 trong thanh ghi TIMSK lên 1). Khi bit này là 0 ngõ ra của bộ so sánh tương tự không được kết nối với ngõ vào của khối input capture của bộ định thời 1.

• Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select : Hai bit này qui định cách thức tạo ra ngắt khi có sự thay đổi trạng thái ở ngõ ra ACO. Chẳng hạn, khi ta thiết lập { ACIS1, ACIS0 } = { 0, 0 } thì khi có sự thay đổi mức (bao gồm mức cao xuống mức thấp hoặc mức thấp lên mức cao) ở ngõ ra ACO sẽ tạo ra ngắt. Các thiết lập khác được mô tả ở bảng 93.

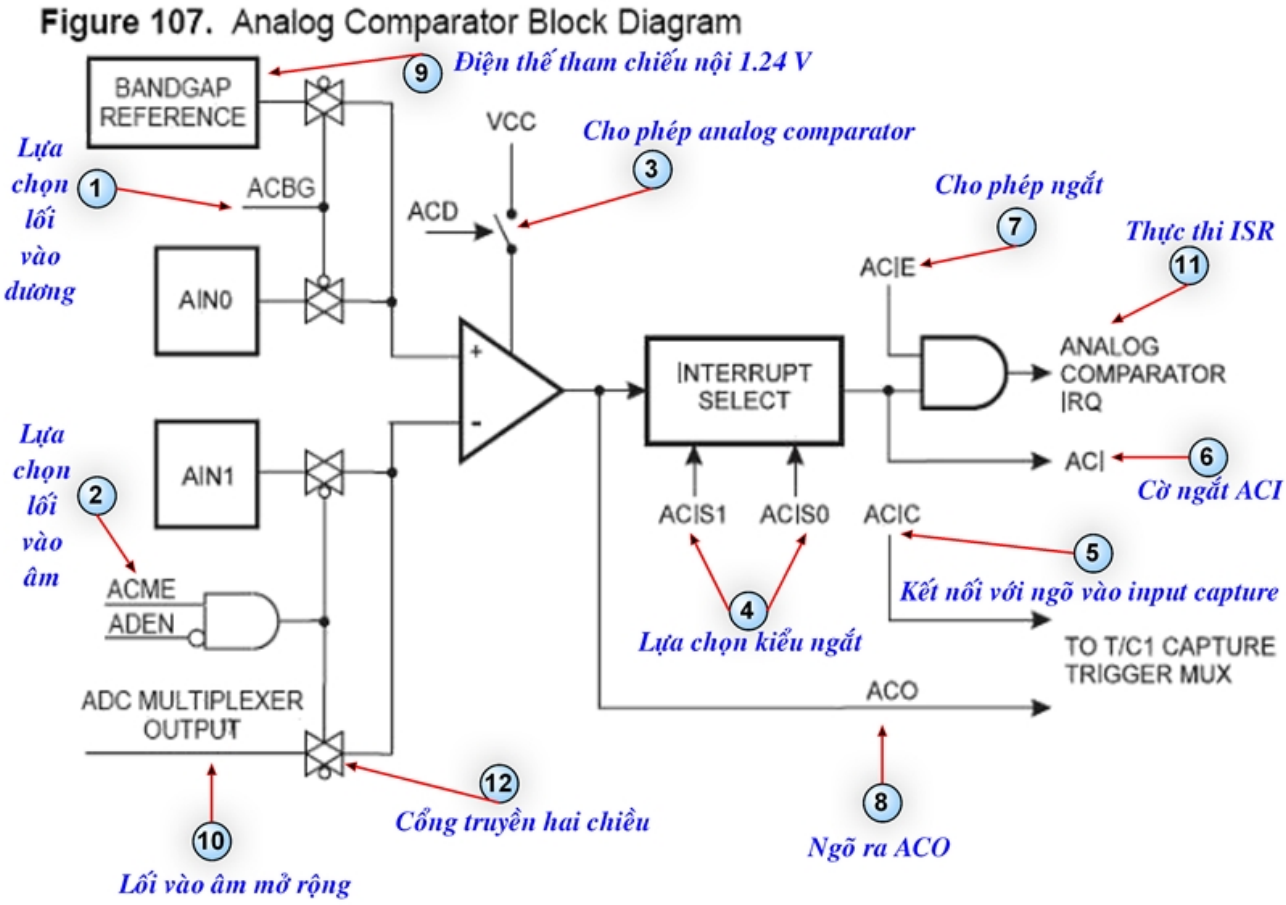
ACIS1	ACIS0	Mô tả
0	0	Thay đổi mức tạo ra ngắt
0	1	Không sử dụng (dự trữ)
1	0	Cạnh xuống ở ngõ ra tạo ra ngắt
1	1	Cạnh lên ở ngõ ra tạo ra ngắt

Bảng 93. Các cách thức tạo ra ngắt ở bộ so sánh tương tự

Chú ý : Khi ta thay đổi một trong hai (hoặc cả hai) bit ACIS1, ACIS0 có thể tạo ra ngắt của bộ so sánh tương tự nếu ngắt được cho phép. Do đó, nếu không cần thiết ta nên cấm ngắt của bộ so sánh tương tự trước khi thay đổi hai bit này.

Hình 107 mô tả cấu trúc của bộ so sánh tương tự của ATmega128, ta có thể phân tích hoạt động của bộ so sánh tương tự thông qua sơ đồ này. Đầu tiên là tín hiệu ACBG (nút số 1), khi ACBG là 1 chân AIN0 bị cấm, điện thế tham chiếu nội (nút số 9) đi qua cổng truyền hai chiều tới ngõ vào dương. Ngược lại khi ACBG là 0 điện thế tham chiếu nội bị cấm. Xét tín hiệu ACME và ADEN (nút số 2) hai tín hiệu này điều khiển 2 cổng truyền nối với nó để cho phép ngõ vào âm là AIN1 hay các chân ADC (nút số 10). Tín hiệu ACD

(nút số 3) là 1 sẽ cấp nguồn cho bộ so sánh tương tự hoạt động, ngược lại nguồn nuôi của bộ so sánh tương tự bị ngắt. Bạn đọc có thể tự phân tích các tín hiệu còn lại.



Hình 107. Bộ So Sánh Tương Tự

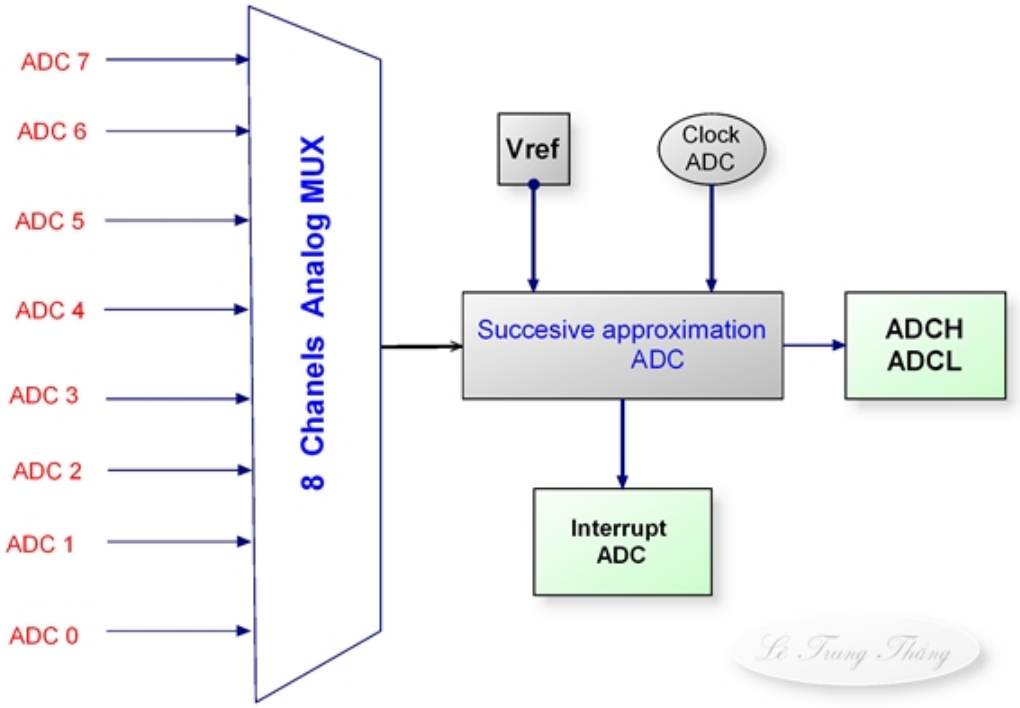
Tóm lại để lập trình cho bộ so sánh tương tự ta thực hiện các bước sau:

1. Chọn ngõ vào dương (là điện thế tham chiếu nội hay chân AIN0) bằng cách thiết lập bit ACBG.
2. Chọn ngõ vào âm (là các chân ADC hay chân AIN1) bằng cách thiết lập các bit ACME và ADEN.
3. Chọn kiểu hoạt động của bộ so sánh tương tự như: sử dụng ngắt, kết nối tới bộ định thời 1…
4. Ghi bit ACD thành 0 để cho phép bộ so sánh tương tự hoạt động.

II. BỘ BIẾN ĐỔI ADC

1. Gới Thiệu Bộ ADC Của ATmega128.

Bộ biến đổi ADC có chức năng biến đổi tín hiệu tương tự (analog signal) có giá trị thay đổi trong một dải biết trước thành tín hiệu số (digital signal). Bộ ADC của ATmega128 có độ phân giải 10 bit, sai số tuyệt đối ± 2 LSB, dải tín hiệu ngõ vào từ 0v – V_{CC}, tín hiệu ngõ vào có nhiều lựa chọn như : có 8 ngõ vào đa hợp đơn hướng (Multiplexed Single Ended), 7 ngõ vào vi sai (Differential Input), … Bộ ADC của ATmega128 là loại ADC xấp xỉ liên tiếp (succesive approximation ADC) với hai chế độ hoạt động có thể lựu chọn là **chuyển đổi liên tục** (Free Running) và **chuyển đổi từng bước** (Single Conversion). Sơ đồ khối đơn giản của một bộ ADC được thể hiện như **hình 5.2.**



Hình 5.2. Sơ đồ đơn giản của một khối ADC

Nguyên tắc hoạt động của khối ADC : Tín hiệu tương tự đưa vào các ngõ ADC0:7 được lấy mẫu và biến đổi thành tín hiệu số tương ứng. Tín hiệu số được lưu trong hai thanh ghi ACDH và ADCL. Một ngắt có thể được tạo ra khi hoàn thành một chu trình biến đổi ADC.

Thực tế, bộ ADC của ATmega128 phức tạp hơn nhiều, tuy nhiên cơ sở vẫn dựa vào nguyên tắc trên. Để khảo sát bộ ADC của ATmega128 ta cần tìm hiểu các khối chức năng sau:

❖ **Điện Thế Tham Chiếu:** là giá trị điện thế dùng để so sánh với điện thế của tín hiệu tương tự cần biến đổi ở ngõ vào ADC. ATmega128 có 3 lựa chọn điện thế tham chiếu là AVCC bằng với VCC, điện thế tham chiếu nội 2.56v, và Vref là tùy chọn. Bạn đọc cần để ý là AVR có 2 nguồn điện thế tham chiếu nội là **internal reference** = 2.56v và **bandgap reference** = 1.24v. Điện thế **bandgap reference** là một hằng số vật lý, nó luôn là 1.24v, còn điện thế **internal reference** thì có thể thay đổi tùy theo các dòng chip khác nhau. Trong AVR, **internal reference** được tạo ra từ **bandgap reference**. Trong tài liệu này, tác giả điều dịch hai dạng điện thế trên điều là **điện thế tham chiếu nội**, tuy vậy, bạn đọc nên hiểu sự khác nhau giữa hai khái niệm trên.

❖ **Tần Số Clock ADC**: là tần số clock cung cấp cho bộ biến đổi ADC, giá trị có thể thay đổi từ vài KHz đến vài MHz. Tuy nhiên, tần số thích hợp khoảng từ 50KHz đến 200KHz cho độ phân giải 10 bit và có thể cao hơn 200KHz nếu độ phân giải thấp hơn.

❖ **Ngõ Vào Tương Tự**: ATmega128 có hai lựa chọn ngõ vào tương tự:

- **10 ngõ vào đơn hướng (single ended):** 10 ngõ vào này là ADC0:7, AGND và bandgap reference. Thực tế ta thường dùng 8 ngõ vào ADC0:7. Vì có 8 ngõ vào ADC0:7 nên ta có thể đưa vào 8 tín hiệu tương tự khác nhau. Khi lựa chọn ngõ vào kiểu này (tức kiểu single ended) thì kết quả chuyển đổi được tính như sau:

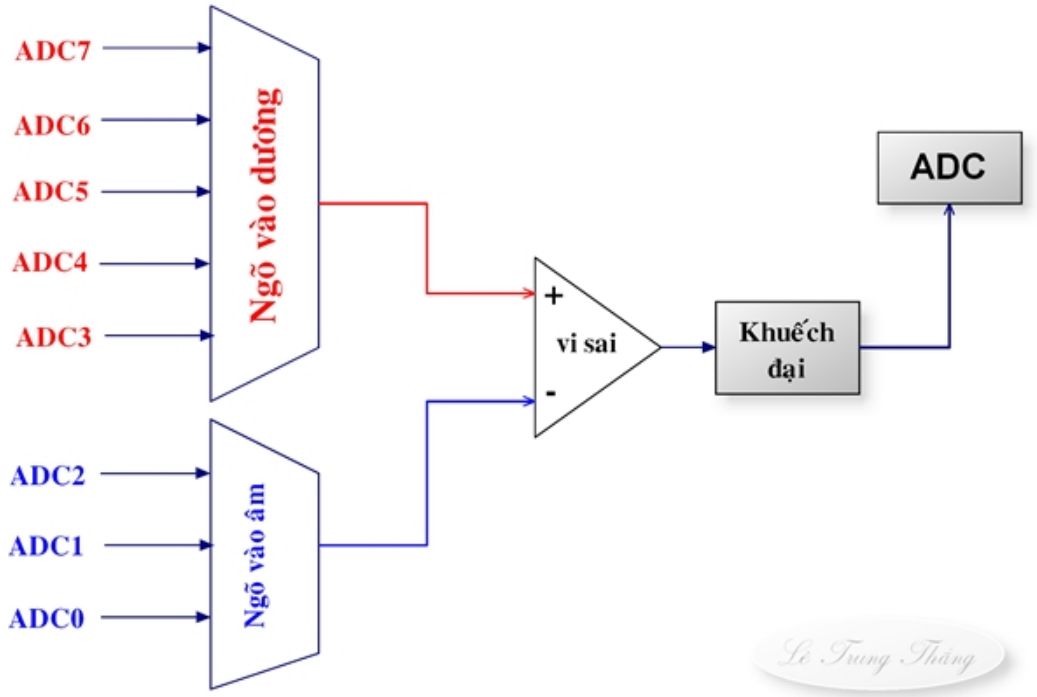
$$ADC = \frac{Vin * 1024}{Vref}$$

- **Ngõ vào vi sai**: Ta có thể đưa hai tín hiệu tương tự vào ngõ vào ADC, hai tín hiệu tương tự này sẽ qua một bộ vi sai (mạch trừ), kết quả ở ngõ ra có thể được khuếch đại rồi sau đó mới đưa vào khối ADC để biến đổi. Bộ vi sai có 2 ngõ vào là **Vpos** (ngõ vào dương) và **Vneg** (ngõ vào âm). Các chân ADC3:7 dùng làm ngõ vào dương, các chân ADC0:2 là ngõ vào âm, **hình 5.3**. Đối với lựa chọn này, kết quả ADC sẽ là :

$$ADC = \frac{(V_{POS} - V_{NEG}) * Gain * 512}{V_{REF}}$$

Ở đây **Gain** là độ lợi có thể tùy chọn. Công thức trên cho thấy kết quả ADC có thể là số âm khi Vpos < Vneg. Do đó, dải giá trị của ADC trong trường hợp này là -512 tới 511. Vì vậy, kết quả trong thanh ghi ADC được biểu diễn dưới dạng số bù 2. Để biết được kết quả là số âm hay dương ta kiểm tra bit ADC9 (trong thanh ghi ADCH), nếu bit này là 1 thì kết quả là số âm, nếu bit này là 0 thì kết quả là số dương.

Chú ý: Điện thế qua bộ vi sai có thể âm, nhưng điện thế cấp ở các ngõ vào ADC0:7 (cho cả hai trường hợp ngõ vào vi sai và ngõ vào đơn hướng) phải luôn nằm trong khoảng 0v – AVCC.



Hình 5.3. Ngõ vào vi sai

❖ **Chế Độ Hoạt Động:** Có hai chế độ hoạt động của bộ ADC là **chuyển đổi liên tục** (Free Running) và **chuyển đổi từng bước** (Single Conversion).

- **Chuyển đổi liên tục:** là chế độ mà sau khi khởi động thì bộ ADC thực hiện chuyển đổi liên tục không ngừng.
- **Chuyển đổi từng bước:** là mà bộ ADC sau khi hoàn thành một chuyển đổi thì sẽ ngừng, một chuyển đổi tiếp theo chỉ được bắt đầu khi phần mềm có yêu cầu chuyển đổi tiếp.

2. Các Thanh Ghi Của Bộ ADC.

a. Thanh ghi ADC Multiplexer Selection - ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**
- **Bit 5 – ADLAR: ADC Left Adjust Result**
- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

- **Bit 7:6 – REFS1:0: Reference Selection Bits**: hai bit này dùng để lựa chọn điện thế tham chiếu là một trong 3 nguồn: AVCC, Điện thế tham chiếu nội 2.56v và VREF như **bảng 97**. Nếu chọn điện thế VREF thì các tùy chọn còn lại không được sử dụng để tránh bị ngắn mạch, điều này có nghĩa là nếu ta chọn điện thế tham chiếu là V_{REF} rồi, thì trong suốt quá trình hoạt động của bộ ADC ta không được lựa chọn điện thế tham chiếu khác, vì nếu không, nguồn điện thế V_{REF} bên ngoài do chưa được tháo đi sẽ làm hỏng chip do ngắn mạch.

Table 97. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Bảng 97. Lựa chọn điện thế tham chiếu

- **Bit 5 – ADLAR: ADC Left Adjust Result:** Bit này lựa chọn cách bố trí dữ liệu trong hai thanh ghi dữ liệu ADCH và ADCL. Xem phần mô tả hai thanh ghi dữ liệu ADCH và ADCL để biết chi tiết.
- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits:** Các bit này lựa chọn kiểu ngõ vào (đơn hay vi sai) và độ lợi, **xem bảng 98.**

Table 98. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000(1)		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(1)	N/A	ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.23V (V_{BG})	N/A		
11111	0V (GND)			

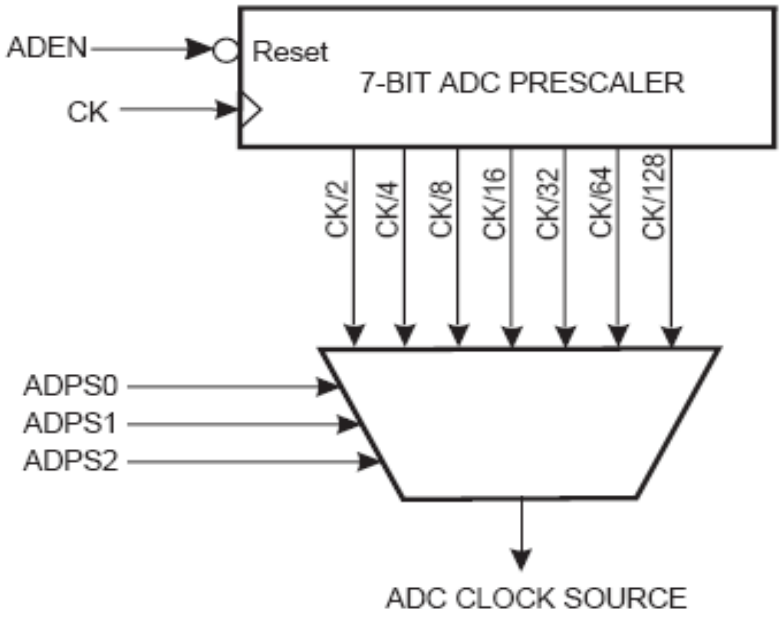
Bảng 98. Lựa chọn kiểu ngõ vào và độ lợi

b. Thanh ghi ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC Enable**
- **Bit 6 – ADSC: ADC Start Conversion**
- **Bit 5 – ADFR: ADC Free Running Select**
- **Bit 4 – ADIF: ADC Interrupt Flag**
- **Bit 3 – ADIE: ADC Interrupt Enable**
- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

• **Bit 7 – ADEN: ADC Enable:** Bit này là 1 sẽ cho phép bộ ADC hoạt động, ngược lại, sẽ ngừng bộ ADC ngay cả khi nó đang trong quá trình biến đổi.
• **Bit 6 – ADSC: ADC Start Conversion:** Ghi bit này thành 1 để bắt đầu quá trình chuyển đổi. Trong chế độ chuyển đổi từng bước, sau mỗi lần chuyển đổi hoàn thành bit này bị xóa về 0, ta phải set lại bit này để bắt đầu một biến đổi tiếp theo. Trong chế độ chuyển đổi liên tục, ta chỉ cần set bit này một lần.
• **Bit 5 – ADFR: ADC Free Running Select:** Set bit này lên 1 để lựa chọn chế độ hoạt động **biến đổi liên tục.** Bit này là 0 sẽ cho phép chế độ biến đổi từng bước.
• **Bit 4 – ADIF: ADC Interrupt Flag:** Bit này sẽ được set thành 1 khi một chu trình biến đổi ADC hoàn thành, bit này được xóa bởi phần cứng khi trình phục vụ ngắt tương ứng được thực thi. *Chú ý là khi ta chỉnh sửa thanh ghi ADCSRA (như dùng các lệnh CBI, SBI) thì bit này sẽ bị xóa. Vì vậy, để xóa bit này bởi phần mềm, ta chỉ cần ghi giá trị 1 vào nó.*
• **Bit 3 – ADIE: ADC Interrupt Enable:** Bit này cho phép ngắt ADC, khi bit ADIE (cho phép ngắt ADC) và bit I (cho phép ngắt toàn cục) trong thanh ghi SREG được set lên 1 sẽ cho phép ngắt ADC hoạt động.
• **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits:** Vì tần số clock ADC được lấy từ xung clock hệ thống (hình 109), nên các bit **ADPS2:0** sẽ cho phép chia xung clock hệ thống với các hệ số xác định (**bảng 99**) trước khi đưa vào nguồn clock ADC. Với độ phân giải 10 bit, tần số clock ADC khoảng từ 50 – 200 KHz, nên tùy theo tần số clock hệ thống mà ta lựa chọn hệ số chia thích hợp.
*Để ý: Trình biên dịch AVRStudio 4 của Atmel xem ADCSRA và ADCSR là một, cả hai điều chỉ tới thanh ghi ADCSRA. Chẳng hạn, lệnh **sbi** ADCSRA, ADSC và **sbi** ADCSR, ADSC là tương đương.*



Hình 109. Nguồn clock ADC

Table 99. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Bảng 99. Lựa chọn các hệ số chia cho nguồn clock ADC

c. Thanh ghi ADC Data Register – ADCL and ADCH

Đây là hai thanh ghi chứa kết quả ADC, tùy theo thiết lập của bit ADLAR (trong thanh ghi ADMUX) mà 10 bit dữ liệu ADC có thể được bố trí về phía phải hay trái của hai thanh ghi ADCH và ADCL, cụ thể như sau:

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ASDLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

Tóm tắt:
Để sử dụng bộ ADC ta thực hiện các bước sau:

1. Cấu hình cho bộ ADC: chọn điện thế tham chiếu, kiểu ngõ vào bằng cách cấu hình cho thanh ghi ADMUX.
2. Cho phép ADC hoạt động: Chọn chế độ hoạt động, các ngắt, tần số Clock ADC bằng cách cấu hình cho thanh ghi ADCSRA.

Ví dụ. Đoạn chương trình nhỏ sau cho phép bộ ADC hoạt động ở chế độ biến đổi từng bước, ngỏ vào là chân ADC3, không dùng ngắt. (viết bằng C có thể xem ở chương VII)

```
ADC_Init:

ldi r16,3  ;
out ADMUX, r16          // chọn ngõ vào ADC3, điện thế tham chiếu V_REF

ldi r16, 0b10000101
out ADCSRA, r16         // không dùng ngắt, hệ số chia clock là 32, chạy từng bước

sbi ADCSRA, ADSC    // khởi động bộ ADC

Wait:
sbis ADCSRA, ADIF   //đợi ADC hoàn thành
rjmp Wait

in r16, ADCL            // lưu kết quả ADC
in r17,ADCH
```

III. BỘ TRUYỀN NHẬN DỮ LIỆU NỐI TIẾP USART

1. Tóm Lược Về USART.

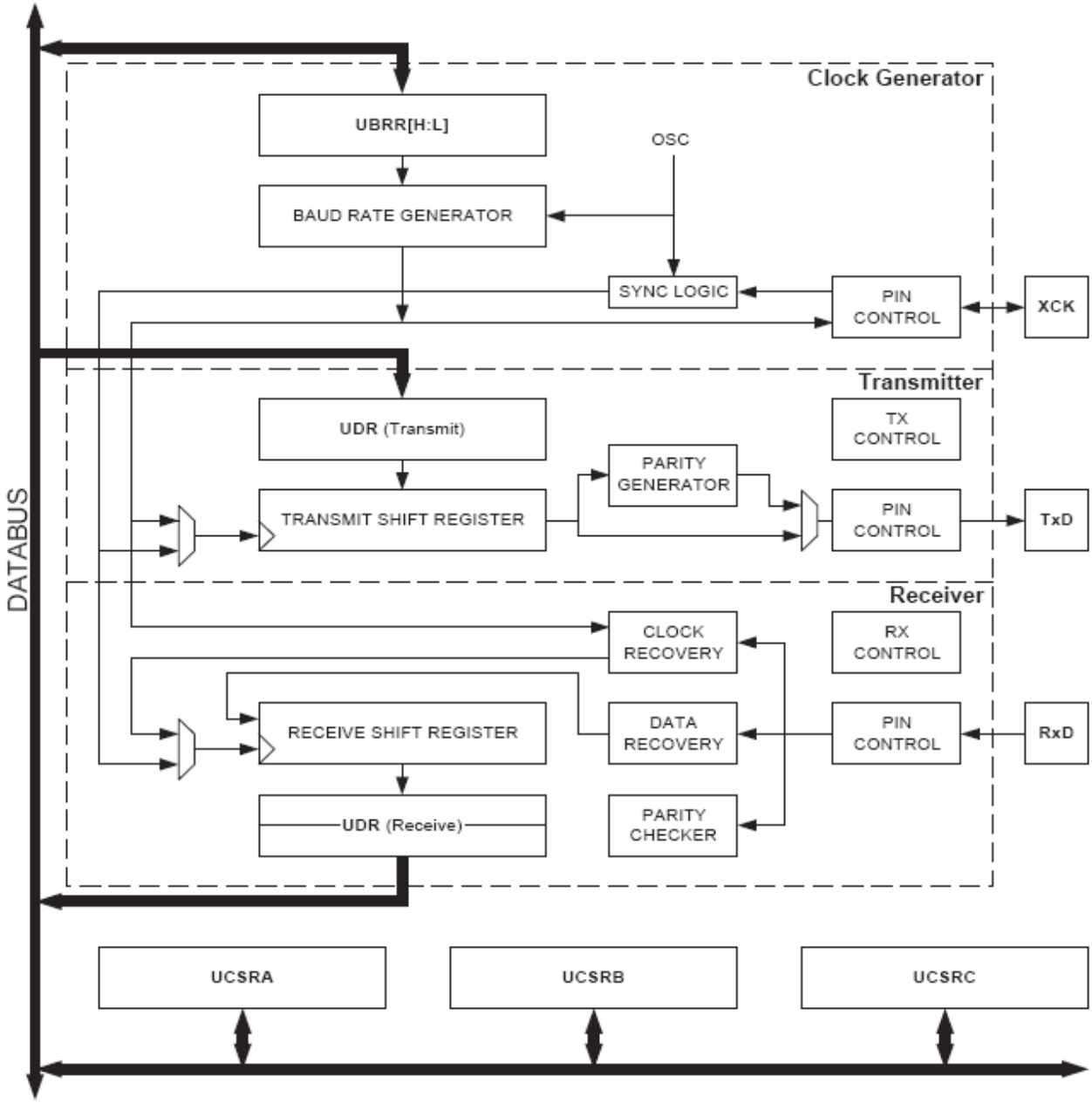
USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter): Bộ Truyền Nhận Nối Tiếp Đồng Bộ Và Bất Đồng Bộ Phổ Dụng, đây là khối chức năng dùng cho việc truyền thông giữa vi điều khiển với các thiết bị khác. Trong vấn đề truyền dữ liệu số, có thể phân chia cách thức (method) truyền dữ liệu ra hai chế độ (mode) cơ bản là : Chế độ truyền nhận **Đồng bộ** (Synchronous) và Chế độ truyền nhận **Bất đồng bộ** (Asynchronous). Ngoài ra, nếu ở góc độ phần cứng thì có thể phân chia theo cách khác đó là: Truyền nhận dữ liệu theo kiểu **Nối tiếp** (serial) và **Song song** (paralell).

- **Truyền Đồng Bộ**: là kiểu truyền dữ liệu trong đó bộ truyền (Transmitter) và bộ nhận (Receiver) sử dụng chung một xung đồng hồ (clock). Do đó, hoạt động truyền và nhận dữ liệu diễn ra đồng thời. Xung clock đóng vai trò là tín hiệu đồng bộ cho hệ thống (gồm khối truyền và khối nhận). Ưu điểm của kiểu truyền đồng bộ là tốc độ nhanh, thích hợp khi truyền dữ liệu khối (block).
- **Truyền Bất Đồng Bộ:** Là kiểu truyền dữ liệu trong đó mỗi bộ truyền (Transmitter) và bộ nhận (Receiver) có bộ tạo xung clock riêng, tốc độ xung clock ở hai khối này có thể khác nhau, nhưng thường không quá 10 $^{o}/_{o}$. Do không dùng chung xung clock, nên để đồng bộ quá trình truyền và nhận dữ liệu, người ta phải truyền các bit đồng bộ (Start, Stop,…) đi kèm với các bit dữ liệu. Các bộ truyền và bộ nhận sẽ dựa vào các bit đồng bộ này để quyết định khi nào thì sẽ thực hiện hay kết thúc quá trình truyền hoặc nhận dữ liệu. Do đó, hệ thống truyền không đồ bộ còn được gọi là hệ thống truyền "tự đồng bộ".

Từ hai kiểu truyền dữ liệu cơ bản trên, người ta đưa ra nhiều giao thức (Protocol) truyền khác nhau như: SPI (đồng bộ), USRT (đồng bộ), UART (bất đồng bộ),…Tuy vậy, cũng có giao thức truyền mà không thể xếp được vào kiểu nào: đồng bộ hay bất đồng bộ, chẳn hạn kiểu truyền I_2C (Trong AVR gọi là TWI), tuy vậy một cách hơi gượng ép thì có thể thấy giao thức truyền I_2C gần với kiểu đồng bộ hơn vì các thiết bị giao tiếp với nhau theo chuẩn I_2C điều dùng chung một xung clock.

2. Gới Thiệu Bộ USART Của ATmega128.

ATmega128 có hai bộ USART là USART0 và USART1. Hai bộ USART này là độc lập nhau, điều này có nghĩa là hai khối USART0 và USART1 có thể hoạt động cùng một lúc. Sơ đồ khối đơn giản của khối USART thể hiện trong **hình 79.**



Hình 79. Sơ đồ khối bộ USART

Sơ đồ khối của bộ USART phân chia thành ba phần rõ ràng: Khối tạo xung clock (clock Generator), Khối Truyền (Transmitter) và Khối nhận (Receiver). Còn các thanh ghi điều khiển USART được dùng chung.

.....

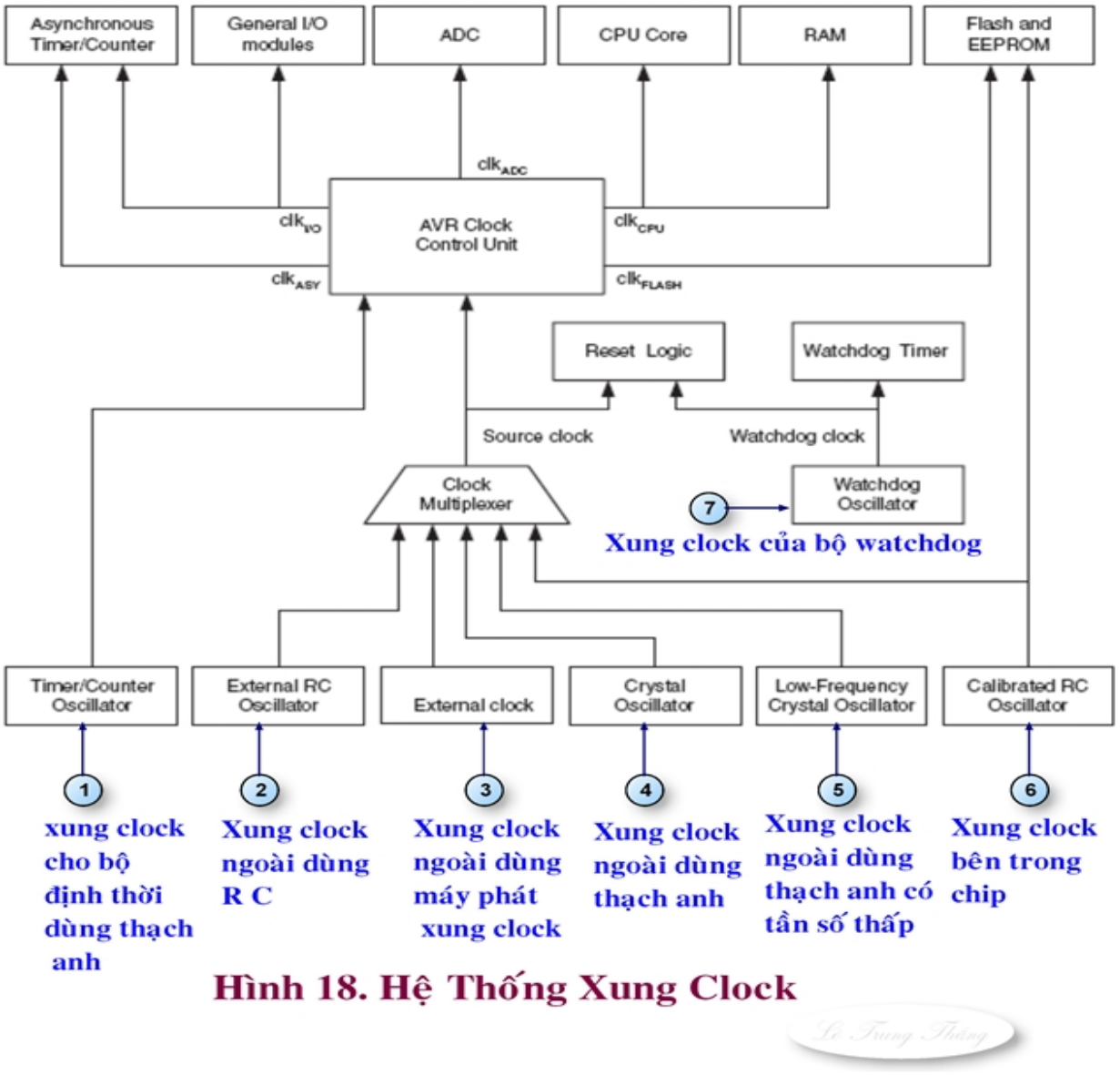
(Phần này chưa đầy đủ, cần được bạn đọc bổ sung)

Chương VI

HỆ THỐNG XUNG CLOCK VÀ LẬP TRÌNH BỘ NHỚ ON-CHIP

I. HỆ THỐNG XUNG CLOCK

Hệ thống xung clock của ATmega128 được chia thành nhiều khối khác nhau, mỗi khối (module) sẽ cung cấp xung clock cho các khối ngoại vi ứng dụng tương ứng. **Hình 18** thể hiện sơ đồ của hệ thống xung clock trên ATmega128.

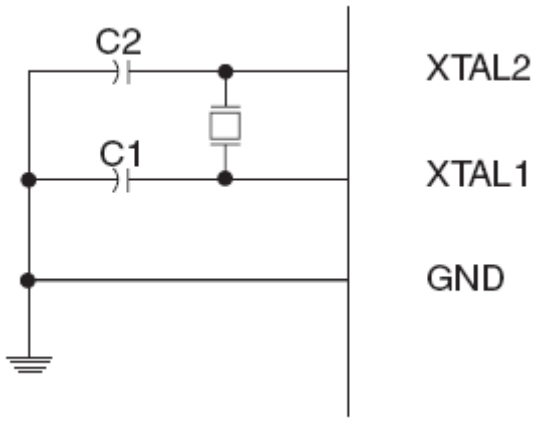


Hình 18. Hệ Thống Xung Clock

Để cấu hình cho chip hoạt động theo chế độ xung clock nào, người ta dùng các bit cầu chì (fuse bit) CKSEL 3, CKSEL2, CKSEL 1. Ngoài ra khi vi điều khiển được đánh thức từ các chế độ nghỉ sang chế độ hoạt động bình thường, bộ tạo dao động cần có một khoảng thời gian để ổn định, khoảng thời gian này gọi là thời gian khởi động (start-up time). CPU chỉ thực hiện lệnh khi hết khoảng thời gian khởi động này. Khi ta reset CPU cũng cần một khoảng thời gian trì hoãn (delay time) để nguồn nuôi đạt mức ổn định trước khi thực bắt đầu thực thi lệnh. Người ta dùng các bit cầu chì CKSEL 0, SUT1, SUT0 để thiết lập thời gian khởi động và thời gian trì hoãn. Khoảng thời gian khởi động và thời gian trì hoãn được đo được đo bằng một đồng hồ riêng, đó là bộ dao động Watchdog. Tần số của bộ dao động Watchdog phụ thuộc vào điện thế nguồn nuôi và nhiệt độ môi trường. Ở Vcc = 5V và nhiệt độ 25^{o}C thì tần số của bộ dao động Watchdog là 1 MHz. Liên quan đến việc thiết lập của hệ thống xung clock người ta còn dùng tới bit cầu chì CKOPT mà vai trò của nó khá linh hoạt tùy theo việc thiết lập xung clock cho hệ thống như thế nào. Hình 18 cho thấy ATmega128 có tới 7 bộ tạo xung clock có thể được lựa chọn. Dưới đây là mô tả cụ thể cho từng trường hợp cấu hình xung clock của hệ thống.

1. BỘ DAO ĐỘNG THẠCH ANH

Bộ dao động thạch anh được mắc theo hình 19. Trong đó chân XTAL1 và XTAL2 (tương ứng chân số 24 , 23 của vi điều khiển) lần lượt là ngõ vào và ngõ ra của bộ khuếch đại đảo được tích hợp sẵn trong chip.



Hình 19. Ghép nối bộ dao động thạch anh

Giá trị của tụ C1 và C2 phải bằng nhau và thường có giá trị vào khoảng 12pF – 22pF. Với ATmega128 thì tần số xung clock hệ thống tối đa là 16MHz và để đạt được tần số tối đa này bit cầu chì CKOPT phải được lập trình (ghi thành 0). Nếu bit CKOPT không được lập trình (ghi giá trị 1) thì tần số tối đa chỉ là 8 MHz. Các bit CKSEL3..1 được dùng để lựa

chọn dải tần số tối ưu như trong **bảng 8**. Các bit CKSEL0 và SUT1..0 được dùng để thiết lập thời gian khởi động (start-up) và thời gian trì hoãn (delay time) như trong bảng 9. Ta cũng có thể thay thế tinh thể thạch anh (Quartz crystal) bằng gốm cộng hưởng (Ceramic Resonator).

Table 8. Crystal Oscillator Operating Modes

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals
1	101(1)	0.4 - 0.9	–
1	110	0.9 - 3.0	12 pF - 22 pF
1	111	3.0 - 8.0	12 pF - 22 pF
0	101, 110, 111	1.0 -	12 pF - 22 pF

Bảng 8. Tối ưu dải tần số

Lựa chọn (1) chỉ nên dùng cho gốm cộng hưởng, không nên dùng cho thạch anh

Table 9. Start-up Times for the Crystal Oscillator Clock Selection

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	–	Ceramic resonator, BOD enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

Bảng 9. Thiết lập thời gian khởi động và trì hoãn

Ví dụ để sử dụng thạch anh 16 MHz làm xung clock hệ thống, thời gian khởi động là 16 K (16384 chu kì xung clock của bộ dao động watchdog) và thời gian trì hoãn là 65 ms thì ta cần thiết lập cho các bit cầu chì là :

{ CKOPT, CKSEL3..0, SUT1..0 } = { 0, 1, 0, 1, 1, 1, 1 }

2. BỘ DAO ĐỘNG THẠCH ANH CÓ TẦN SỐ THẤP

Thạch anh trong trường hợp này có tần số thấp 32,768 KHz được mắc vào mạch như hình 19. Tần số thấp được sử dụng để giảm công suất tiêu thụ của hệ thống và thích hợp cho các ứng dụng cần đo thời gian thực. Để cấu hình cho hệ thống xung clock theo chế độ này, cần thiết lập các bit cầu chì { CKSEL3..0 } = { 1, 0, 0, 1 }. Các tụ C1, C2 cũng có thể được bỏ đi bằng cách lập trình cho bit CKOPT để cho phép tụ bên trong chip hoạt động. Tụ bên trong chip có giá trị định danh là 36 pF. Thời gian khởi động và thời gian trì hoãn được lựa chọn nhờ vào các bit cầu chì SUT1..0 theo như **bảng 10**.

Table 10. Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	1K CK(1)	4.1 ms	Fast rising power or BOD enabled
01	1K CK(1)	65 ms	Slowly rising power
10	32K CK	65 ms	Stable frequency at start-up
11	Reserved		

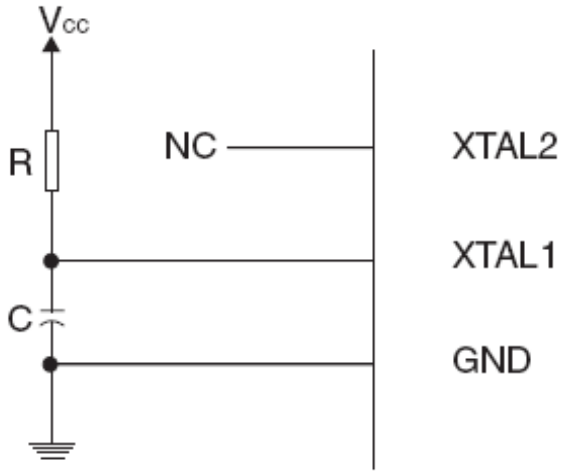
Bảng 10. Thiết lập thời gian khởi động và trì hoãn

3. BỘ DAO ĐỘNG R-C BÊN NGOÀI

Bộ dao động R-C bên ngoài thích hợp cho những ứng dụng không đòi hỏi cao về sự chính xác thời gian . Mạch R-C được mắc như hình 20. Tần số dao động vào khoảng:

$$f = \frac{1}{3 * R * C}$$

Trong đó giá trị của C phải tối thiểu là 22 pF. Tuy nhiên ta cũng có thể bỏ đi tụ C bằng cách lập trình cho bit cầu chì CKOPT để cho phép tụ bên trong chip (mắc giữa XTAL1 và GND) hoạt động. Giá trị định danh của tụ bên trong chip là 36 pF. Các bit cầu chì CKSEL3..0 sẽ cấu hình dãi tần số tối ưu như bảng 11 và các bit cầu chì SUT1..0 sẽ thiết lập thời gian khởi động và thời gian trì hoãn như **bảng 12.**



Hình 20. Mạch dao động R-C

Table 11. External RC Oscillator Operating Modes

CKSEL3..0	Frequency Range (MHz)
0101	0.1 - 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

Bảng 11. Tối ưu dải tần số

Table 12. Start-Up Times for the External RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	18 CK	–	BOD enabled
01	18 CK	4.1 ms	Fast rising power
10	18 CK	65 ms	Slowly rising power
11	6 CK(1)	4.1 ms	Fast rising power or BOD enabled

Bảng 12. Thiết lập thời gian khởi động và trì hoãn

4. BỘ DAO ĐỘNG NỘI R-C TINH CHỈNH ĐƯỢC

Bộ dao động nội RC cung cấp các tần số xung clock cố định 1 MHz, 2 MHz, 4 MHz, 8 MHz (ở Vcc = 5V và nhiệt độ 25°C). Ta có thể dùng xung clock này như là xung clock của hệ thống bằng cách cấu hình cho các bit cầu chì CKSEL 3..0 được chỉ ra ở **bảng 13**. Khi sử dụng xung clock của bộ dao động nội làm xung clock của hệ thống ta không cần phải dùng bộ dao động bên ngoài. Khi cấu hình xung clock hệ thống theo trường hợp này bit cầu chì CKOPT không được lập trình (ghi là 1). Vì bộ dao động watchdog độc lập với bộ dao động nội RC (xem hình 18) nên khi hệ thống hoạt động theo xung clock của bộ dao động nội RC thì bộ dao động watchdog vẫn được sử dụng cho bộ định thời watchdog. Ngoài ra, người dùng có thể tinh chỉnh tần số của bộ dao động nội bằng cách thay đổi giá trị của thanh ghi OSCCAL. Lí do của việc tinh chỉnh này là bởi vì trong quá trình đếm (tức phát xung clock) của bộ dao động nội, sau 1 thời gian thì sẽ có sai số, ví dụ bộ dao động nội có tần số 1 MHz sau 1000000 lần đếm thì khoảng thời gian tương ứng 1s sẽ trôi qua. Nếu thời gian đếm kéo dài sẽ có thể có sai số. Do đó người ta cần tinh chỉnh lại tốc độ của bộ dao động nội bằng cách làm cho nó đếm nhanh hơn hay chậm đi so với giá trị định danh. Để làm được điều này người ta tăng hay giảm giá trị của thanh ghi OSCCAL.

Table 13. Internal Calibrated RC Oscillator Operating Modes

CKSEL3..0	Nominal Frequency (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Bảng 13. Lựa chọn tần số dao động nội

Khoảng thời gian khởi động và thời gian trì hoãn được thiết lập bởi các bit cầu chì SUT1..0 theo **bảng 14.**

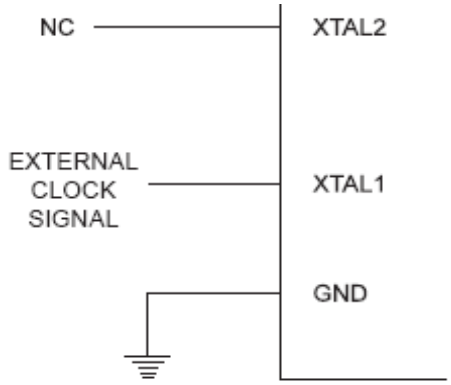
Table 14. Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10(1)	6 CK	65 ms	Slowly rising power
11	Reserved		

Bảng 14. Thiết lập thời gian khởi động và trì hoãn

5. BỘ TẠO XUNG CLOCK BÊN NGOÀI

Người dùng cũng có thể sử dụng một máy phát xung clock bên ngoài để làm xung clock cho hệ thống. Sơ đồ ghép nối với máy tạo xung clock bên ngoài được thể hiện ở **hình 21.**



Hình 21. Ghép nối với máy phát xung clock bên ngoài

Table 16. Start-up Times for the External Clock Selection

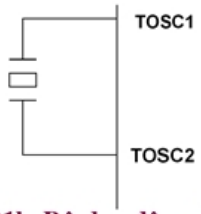
SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved		

Bảng 16. Thiết lập thời gian khởi động và trì hoãn

Trong trường hợp này các bit cầu chì CKSEL3..0 phải ghi thành "0000". Người dùng cũng có thể cho phép tụ bên trong chip (giữa XTAL1 và GND) hoạt động bằng cách lập trình cho bit CKOPT (ghi CKOPT thành 0). Giá trị định danh của tụ bên trong chip là 36 pF. Thời gian khởi động và thời gian trì hoãn được thiết lập bởi các bit SUT1..0 được cho ở **bảng 16.**

6. BỘ DAO ĐỘNG ĐỊNH THỜI

Người dùng cũng có thể mắc trực tiếp bộ dao động thạch anh vào giữa 2 chân TOSC1 và TOSC2 của vi điều khiển (không cần tụ) để tạo xung clock cho hệ thống như **hình 21b**. Bộ dao động được tối ưu cho tần số thạch anh 32,768 KHz.



Hình 21b. Bộ dao động định thời

7. Thanh Ghi XDIV – Chia Tần Số Nguồn Xung Clock

Người ta dùng thanh ghi XDIV để chia tần số của nguồn xung clock cho một số từ 2 đến 129 tùy theo giá trị được ghi vào thanh ghi này.

Bit	7	6	5	4	3	2	1	0	
	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0	XDIV
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• **Bit 7 – XDIVEN: XTAL Divide Enable:** Khi bit này là 1 sẽ cho phép chia tần số xung clock của CPU và xung clock của tất cả các ngoại vi ($clk_{I/O}$, clk_{ADC}, clk_{CPU}, clk_{FLASH}) cho một số từ 2 đến 129 . Giá trị của số chia có thể được thay đổi trong lúc chương trình đang chạy với điều kiện là bit XDIVEN đang ở logic 0. Ghi bit này là 0 thì không cho phép chia (tương đương với chia cho 1).

• **Bits 6..0 – XDIV6..XDIV0: XTAL Divide Select Bits 6 – 0: 7** bit này sẽ xác định giá trị của số chia. Nếu ghi vào 7 bit này giá trị là **d** thì tần số xung clock của CPU và các ngoại vi sẽ là :

$$f_{clk} = \frac{f_{nguồn}}{129 - d}$$

Để thay đổi hệ số **d** thì bit XDIVEN phải xóa về 0 trước khi ghi giá trị mới vào các bit XDIV6..XDIV0.

Chú ý: Khi tần số của hệ thống xung clock được chia, bộ định thời 0 chỉ hoạt động được với xung clock bất đồng bộ, tần số của xung clock bất đồng bộ phải nhỏ hơn ¼ lần tần số xung clock đã chia. Nguồn xung clock bất đồng bộ là nguồn xung clock được tạo từ bộ dao động thạch anh (tối ưu là 32,768 KHz) kết nối trực tiếp tới 2 chân TOSC1 và TOSC2 như Hình 21b. Về nguyên tắc có thể dùng máy phát xung clock kết nối trực tiếp với chân TOSC1 để dùng làm nguồn xung clock bất đồng bộ cho bộ định thời 0. Chi tiết về chế độ hoạt động bất động bộ của bộ định thời 0 được trình bày ở chương 3 " Bộ Định Thời của ATmega128 ", mục 3.

Theo mặc định của nhà sản xuất thì giá trị ban đầu của các bit cầu chì là:

{ CKSEL3..0, SUT1..0 } = {0, 0, 0, 1, 1, 0 }, tức Chip sẽ sử dụng bộ dao động nội có tần số 1 MHz với thời gian khởi động là 65 ms .

II. LẬP TRÌNH BỘ NHỚ TRÊN CHIP

(Nếu bạn đã nạp được chương trình vào Chip thì coi như xong phần này ☺)

(Phần này chưa đầy đủ, cần được bạn đọc bổ sung)

Chương VII
LẬP TRÌNH AVR BẰNG NGÔN NGỮ C

Phần này chỉ tóm tắc các vấn đề liên quan tới lập trình AVR bằng ngôn ngữ C, xem như bạn đọc đã có kiến thức về ngôn ngữ C. Trình biên dịch được sử dụng ở đây là CodeVisionAVR (http://www.hpinfotech.com/).

VII.1 Các Chú Thích Và Tiền Xử Lý (PreProcessor)

- **Các Chú Thích**.

Thông thường bắt đầu một chương trình là các chú thích về project, các chú thích phải bắt đầu bằng dấu // hay /* *các chú thích* */ và được trình biên dịch bỏ qua khi biên dịch, chẳn hạn:

```
//*****************************************
//  comments placed in there
//  File:     demo.c
//  Author:   Le Trung Thang
//  Date:     2007
//*****************************************
```

- **Các Tiền Xử Lý.**

#include : Dùng để chèn các file cần thiết vào project, các file này nên để trong thư mục **inc** của trình biên dịch CodeVisionAVR.

Ví dụ:

`#include <mega128.h>` cho phép sử dụng các thanh ghi của Atmega128. Tức báo cho trình biên dịch biết chúng ta đang sử dụng vi điều khiển Atmega128. *Đây sẽ là dòng code đầu tiên trong chương trình C.*

#define : Dùng định nghĩa một giá trị nào đó bằng các kí tự.

Ví dụ:

```
#define max 0xff
```

Định nghĩa **max** có giá trị là 0xff. Chú ý không có dấu chấm phẩy (**;**) ở cuối câu vì define chỉ là một macro chứ không phải là một lệnh. Macro cũng có thể có tham số.

ví dụ:

```
#define SUM(a,b) a+b
main( )
{
//các lệnh khác
int I = SUM(2,3);
//các lệnh khác
};
```

Thì **i** sẽ được gán thành i = 2 + 3 = 5.

VII.2 Các Kiểu Dữ Liệu (DataTypes)

Ngoài các kiểu dữ liệu của C, CodeVisionAVR còn có kiểu dữ liệu **bit** là kiểu dữ liệu 1 bit, nên dải giá trị chỉ có 0 và 1. Kiểu bit chỉ hỗ trợ đối với khai báo biến toàn cục là chính. Với biến bit cục bộ, trình biên dịch chỉ cho khai báo tối đa 8 biến **bit.**
Ví dụ :

```
bit a; //a là biến kiểu bit
```

các kiểu khác được cho trong bảng dưới.

Type	Size (Bits)	Range
bit	1	0 , 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

a. Hằng

- Các hằng số được đặt trong bộ nhớ FLASH, chứ không đặt trong RAM.
- Không được khai báo hằng trong chương trình con.
- Giá trị 100 được hiểu là số thập phân (decimal), 0b101 để chỉ giá trị nhị phân (binary) và 0xff để chỉ giá trị thập lục (hexadecimal)

Ví dụ:

```
const char a = 128 ; // hằng số a có kiểu char và có giá trị là 128.
```

b. Biến

- Biến gồm có biến **toàn cục** (global) là biến mà hàm nào cũng có thể truy xuất, và biến **cục bộ** (local) là biến mà chỉ có thể truy xuất trong hàm mà nó được khai báo.
- Biến toàn cục, nếu không có giá trị khởi tạo sẽ được mặt định là 0. Biến cục bộ, nếu không có giá trị khởi tạo sẽ có giá trị không biết trước.
- Biến toàn cục được lưu trữ trong các thanh ghi Rn, nếu đã dùng hết các thanh ghi thì sẽ chuyển sang lưu trữ trong vùng SRAM. Để ngăn cản các biến toàn cục được lưu vào các thanh ghi Rn, dù các thanh ghi này vẫn còn tự do, ta dùng từ khóa **volatile** (xem sau)**.**
- Biến toàn cục, nếu không lưu trong các thanh ghi đa chức năng thì được lưu trữ trong bộ nhớ **SRAM**, còn biến cục bộ, nếu không lưu trong các thanh ghi đa chức năng, thì được lưu trữ trong vùng **data STACK**. Khi chương trình trả về giá trị cuối cùng cho hàm thì các biến cục bộ được lưu trữ trong stack sẽ bị xóa. Để biến cục bộ không bị xóa khi thoát khỏi hàm ta dùng từ khóa **static**.
- Biến **bit** toàn cục được cấp phát ở các thanh ghi R2 tới R14 của vi điều khiển, các bit được cấp phát từ R2 tới R14 theo thứ tự khai báo, nhắc lại là ATmega128 có 32 thanh ghi đa chức năng R0 đến R31.
- Trong chương trình C, nơi bắt đầu thực thi chương trình là điểm bắt đầu của hàm **main.** Thực tế, khi biên dịch sang hợp ngữ (assembly), điểm bắt đầu của chương trình vẫn là vị trí **vector reset** (địa chỉ 0000h). Trước khi chạy tới vị trí chương trình **main**, chương trình hợp ngữ sẽ thực hiện khởi tạo các biến toàn cục, stack,... Do đó, khi chạy vào hàm main, các biến toàn cục, mà thực chất là các ô

nhớ (byte hay word), đã có giá trị khởi tạo sẵn. Với các biến cục bộ, trình hợp ngữ không khởi tạo trước giá trị.

- **ví dụ:** khai báo biến cục bộ như sau:

```
main ( )
{ unsigned char test = 9 ;
test+=1;
}
```

Sẽ dịch sang hợp ngữ là :

```
LDI     Rn,0x09 ;// n tùy theo dòng chip và chương
SUBI    Rn,0xFF ;// trình ta viết, R17 chẳn hạn
```

Như vậy, với biến cục bộ, khi nào sử dụng thì mới khởi tạo.

Ví dụ 1:

```
/* khai báo biến toàn cục */
char a;
int b;
/* có thể khởi tạo giá trị */
long c = 0b1111;
/*  chương trình con */
int increment(void)
 {
/* khai báo biến static */
static int n ;
return n++ ;
}
/* chương trình chính*/
void main(void) {
/* khai báo biến cục bộ */
char d;
int e;
/* có thể khởi tạo giá trị */
long f = 16;
d = increment() ;
/* d = 1 */
e = increment() ;
/* e = 2 , vì khi thoát khỏi hàm increment thì giá trị của
biến static n
 vẫn không bị xóa*/
     }
```

Ví dụ 2:

```
bit  bit_mot ; // bit 0 của thanh ghi R2 được cấp cho biến
bit_mot
bit  bit_hai ; // bit 1 của thanh ghi R2 được cấp cho biến
bit_hai
```

Để ý là các biến kiểu **bit** trên là biến toàn cục, đối với biến **bit cục bộ**, trình biên dịch sẽ cất trong thanh ghi R15. Các thanh ghi R2 tới R14 cũng có thể được cấp phát cho biến thanh ghi (register variable), tùy vào các tùy chọn khi cấu hình cho trình biên dịch. (có thể không đúng với các version mới của trình biên dịch).

Biến volatile:

- Để tương thích với các thiết bị ngoại vi khi ghép nối với vi điều khiển, chẳn hạn bộ ADC, ghép nối với RTC… người ta dùng các biến **volatile.**

Biến Volatile là biến mà giá trị của nó không được thay đổi bởi chương trình, nhưng có thể được thay đổi bởi phần cứng.

Ví dụ 3.

Ta muốn ghép nối MCU với một Real time clock (RTC), giá trị của thanh ghi RTC được đọc sau mỗi một khoảng thời gian.

```
unsigned int  *milliseconds = 0x8000 ;  // trỏ tới thanh
//ghi chứa giá trị giây của chip RTC
unsigned int x,y,time ;
time = *milliseconds ;    // (1) Đọc giá trị thanh ghi RTC lần 1
x = time ;
time = *milliseconds ;  // (2) Đọc giá trị thanh ghi RTC lần 2
y = time ;
```

Đoạn chương trình trên sẽ chỉ đọc giá trị thanh ghi RTC có một lần nên kết quả là thời gian thể hiện sẽ không đúng, nguyên nhân là từ dòng (1) tới (2), biến `*milliseconds` không bị thay đổi giá trị (trong chương trình ta không làm gì để thay đổi `*milliseconds` cả), do đó trình biên dịch sẽ tối ưu đoạn code trên bằng cách bỏ bớt dòng số (2), gán y = x. Nhưng thực tế là phần cứng (sự đếm của đồng hồ) làm thay đổi `*milliseconds`. Do đó, giá trị `time` ở (1) sẽ khác `time` ở (2) .

Để ngăn trình biên dịch tối ưu đoạn code trên ta dùng từ khóa **volatile** cho biến `milliseconds`. Đoạn code cần được sửa thành:

```
unsigned int  volatile  *milliseconds = 0x8000 ;  // trỏ tới
//thanh ghi chứa giá trị giây của chip RTC
unsigned int x,y,time ;
time = *milliseconds ;    // (1) Đọc giá trị thanh ghi RTC lần 1
x = time ;
time = *milliseconds ;  // (2) Đọc giá trị thanh ghi RTC lần 2
y = time ;
```

Ta cũng có thể chỉ định việc lưu trữ một biến toàn cục ở một địa chỉ cụ thể trong SRAM bằng cách dùng toán tử @.

Ví dụ 4:

```
int   a   @0x80 ; // biến nguyên a được cất ở địa chỉ 80h trong
//bộ nhớ SRAM
```

c. Chuyển Đổi Kiểu Dữ Liệu

Trong một biểu thức toán học, các toán hạng có thể có kiểu dữ liệu khác nhau, khi đó trình biên dịch sẽ tự động chuyển tất cả các toán hạng về cùng một kiểu duy nhất. Thứ tự ưu tiên chuyển đổi là :

```
Char -> unsigned char -> int -> unsigned int -> long -> unsigned
long -> float
```

Ví dụ 1.

```
int a  ;
long c, b;
c = a*b ; //a sẽ được tự động chuyển thành long
```

Ví dụ 2.
Phép nhân sau đây cho kết quả sai:

```
int  a, b = 30000;
long  c  ;
c = a*b ;
```

phép toán trên sẽ nhân a với b trước, với tích thu được là **int bị tràn**, rồi mới chuyển tích thu được sang **long**, rồi gán tích bị tràn này cho c. Để không bị tràn, ta sửa lại biểu thức trên như sau:

```
int  a, b = 30000;
long  c  ;
c = (long) a*b ;
```

lúc này a, b được chuyển thành **long** trước khi nhân, nên tích sẽ là **long không bị tràn**, rồi gán kết quả cho c.

VII.3 Mảng (Array)

Mảng là một dãy các biến xếp liên tục nhau. Kí hiệu [ ] dùng để khai báo mảng. Mảng khai báo ngoài hàm gọi là **mảng toàn cục** (global array), mảng khai báo trong hàm gọi là **mảng cục bộ** (local array).

Ví dụ:

```
int global_array[4]={1,2,3,4};
//mảng có 4 phần tử (dạng nguyên) có khởi tạo giá trị ban đầu.
global_array[0] = 9 ;
//ghi giá trị 9 vào phần tử đầu tiên của mảng
int multidim_array[2][3]={{1,2,3},{4,5,6}};
//mảng đa chiều có khởi tạo giá trị ban đầu.
```

VII.4 Hàm (Function)

- Hàm là đoạn chương trình thực hiện trọn vẹn một công việc nhất định.
- Hàm chia cắt việc lớn bằng nhiều việc nhỏ. Nó giúp cho chương trình sáng sủa, dễ sửa, nhất là đối với các chương trình lớn.
- Chương trình phục vụ ngắt (ISR) cũng có thể xem là một hàm, nhưng không có tham số truyền vào và cũng không có tham số trả về (xem sau).
- Hàm viết cho MCU cũng giống như viết trên PC, bạn đọc có thể xem lại các tài liệu về ngôn ngữ C khi cần thiết.
- Giá trị trả về của hàm được lưu trong các thanh ghi R30, R31, R22, R23.

VII.5 Con Trỏ (Pointer)

Những biến lưu trữ địa chỉ của một biến khác được gọi là **con trỏ** (pointer). Có hai toán tử liên quan tới con trỏ là : **&** và * .
& : là toán tử lấy địa chỉ, có nghĩa là "**địa chỉ của**" .
*** :** là toán tử tham chiếu, có nghĩa là "**Giá trị được trỏ bởi**".

Để sử dụng con trỏ ta phải khai báo nó. Kiểu khai báo như sau:

```
type * pointer_name;
```

Ví dụ:

```
int *con_tro ;
```

*Để ý là dấu sao (*) mà chúng ta đặt khi khai báo một con trỏ chỉ có nghĩa rằng: Đó là một con trỏ và hoàn toàn không liên quan đến toán tử tham chiếu * mà chúng ta đã nói trên. Đó đơn giản chỉ là hai tác vụ khác nhau được biểu diễn bởi cùng một dấu.*

Khi một biến con trỏ được khai báo, nó chưa chứa giá trị nào cả, giống như các kiểu biến khác. Để gán địa chỉ cho con trỏ chúng ta cần phải gán giá trị cho con trỏ đó (tức khởi tạo con trỏ).

Ví dụ.

```
int number;
int *con_tro;// khai báo biến con_tro là một con trỏ nguyên
con_tro = &number ;// biến con_tro trỏ tới biến number
```

Sau khi khởi tạo, ta có thể sử dụng con trỏ bình thường trong các biểu thức.

Ví dụ.

```
int  value1 = 5 ;
int  value2 = 15;
int * mypointer;
mypointer = &value1; // con trỏ mypointer trỏ tới biến value1
*mypointer = 10; //giá trị của biến value1 = 10
mypointer = &value2; // con trỏ mypointer trỏ tới biến value2
```

```
*mypointer = 20; // giá trị của biến value2 = 20
```

VII.5 Truy Xuất Các Thanh Ghi Vào/Ra (Accessing The I/O Registers)

Việc truy xuất các thanh ghi I/O của AVR khá đơn giản, tất cả các thanh ghi I/O của AVR đã được khai báo trong file **io.h**, ta chỉ việc include file header io.h (hoặc file header cho từng chip cụ thể, **mega128**.h) vào chương trình là có thể sử dụng các thanh ghi này. Chú ý là việc truy xuất **bit** trong các thanh ghi có địa chỉ 5Fh trở lên trong vùng nhớ SRAM là không thể thực hiện được.

Ví dụ.

```
include<io.h>
Char temp ;
temp = PIND; // Đọc giá trị ở cổng D vào biến temp
TCCR0 = 0x4F;// ghi giá trị 4Fh vào thanh ghi TCCR0
DDRD |= 0x0C; // set bit 2 và 3 của thanh ghi DDRD
```

VII.6 Truy Xuất EEPROM

Dùng từ khóa **eeprom** khi khai báo biến (toàn cục) thì biến sẽ được lưu vào EEPROM. Để ý, là biến trong eeprom không có giá trị khởi tạo ngay khi chương trình thực thi, ngay cả khi, trong khai báo biến eeprom ta có khởi tạo giá trị cho biến này. Giá trị khởi tạo chỉ được dùng để nộp trực tiếp vào eeprom bởi phần mềm. (Điều này có thể không chính xác với các phiên bản mới hơn của trình biên dịch).

Ví dụ. (xem thêm ví dụ ở mục **VII.8.h**)

```
eeprom int alfa=1;
eeprom char beta ;
```

ở đây, khi chương trình bắt đầu thực thi, biến **alfa** vẫn không chắc là có giá trị 1, nó là một giá trị không biết trước. Do đó, để chắc chắn ghi giá trị 1 vào biến **alfa**, ta nên viết lại :

```
eeprom int alfa=1;
eeprom char beta ;
alfa = 1 ;
```

Ta cũng có thể đặt một biến vào một vị trí tùy ý trong eeprom bằng cách dùng toán tử @.

Ví dụ:

```
eeprom int alfa @0x10;//biến alfa đặt ở vị trí 0x10 trong eeprom
```

VII.6 Sử Dụng Ngắt (interrupt)

Để thực thi chương trình ngắt, ta dùng từ khóa **interrupt.**
Khuôn dạng của chương trình phục vụ ngắt là :

```
interrupt [interrupt_number] void  routine_name (void)
{
//đặt chương trình phục vụ ngắt ở đây
}
```

- `interrupt_number` : là số thứ tự của vector ngắt, có thể tìm thấy ở datasheet của MCU hay từ file header của MCU trong thư mục **inc.** Ta có thể thay thế số thứ tự của vector ngắt bằng tên gợi nhớ được định nghĩa trong file header của MCU.
- `routine_name`: tên chương trình ngắt, là tùy chọn.
- Chương trình phục vụ ngắt không có tham số truyền vào và cũng không có tham số trả về.

Ví dụ.

```
// Giả định MCU đang dùng là ATmega128, ngắt tràn Timer 0
interrupt [17] void timer0_overflow(void)
{
// Đặt chương trình phục vụ ngắt ở đây
}
Hoặc là:

interrupt [TIM0_OVF] void timer0_overflow(void)
{
// Đặt chương trình phục vụ ngắt ở đây
}
```

Trình biên dịch sẽ tự động lưu giữ giá trị các thanh ghi bị tác động trong lúc đang gọi trình phục vụ ngắt và sẽ phục hồi lại giá trị các thanh ghi này khi thoát khỏi trình phục vụ ngắt. Tuy nhiên ta có thể ngăn cản sự lưu giữ giá trị các thanh ghi bị tác động bằng cách dùng chỉ thị #**pragma savereg** - . (khuyến khích là không nên dùng ☺)

VII.7 Sử Dụng Bộ ADC

Trong C, để sử dụng ADC ta chỉ cần khai báo các thông số cho bộ ADC trong hàm **main**, sau đó có thể sử dụng mẫu hàm đọc ADC do CodeVision AVR tạo ra. *Để ý là nếu sử dụng chức năng debug Jtag thì có một số chân ADC (ADC4,..,ADC7) của AVR không sử dụng được.* Bạn đọc nên xem phần chống nhiễu cho ADC trong datasheet. Ở đây chỉ tập trung vào khía cạnh lập trình.

Ví dụ sau dùng ADC 10bit, tốc độ 500.000 kHz, chạy từng bước. (xem thêm phần mô tả bộ ADC trang 61)

```
#include <mega128.h>  //Khai báo là đang dùng ATmega128
// hàm đọc kết quả ADC, tên hàm là tùy ý,theo chuẩn C
unsigned int read_adc(unsigned char adc_input) {
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
// hàm main
 Void Main( )
{
Unsigned int  adc_temp ; // khai báo biến để lưu kết quả ADC

// Khởi tạo cho bộ ADC ở đây
// Tần số biến đổi: 500.000 kHz
//Chân làm điện thế so sánh cho bộ ADC là AVCC
//Điện thế của AVCC là cố định (bằng VCC cấp cho AVR)
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x84; // xong phần khởi tạo
// ...
adc_temp = read_adc (0) ; //đọc kết quả ADC ở chân ADC0
adc_temp = read_adc (1) ; //đọc kết quả ADC ở chân ADC1
// ...
adc_temp = read_adc (7) ; //đọc kết quả ADC ở chân ADC7
                          //phải disable chức năng jtag
};
```

VII.8 Tóm Tắt Các Cấu Trúc Điều Khiển

Phần này chỉ để giúp các bạn tra cứu nhanh, nếu quên.

a. Cấu trúc điều kiện: if và else.

```
if (condition 1)
  {
   Khối lệnh 1
  }
else if (codition 2)
  {
   Khối lệnh 2
   }
else
  {
   Khối lệnh khác
  }
```

Ví dụ.

```
if(input == KEY_1)PORTD = 0x01;
else if (input == KEY_2) PORTD = 0x02;
else if (input == KEY_3)PORTD = 0x03;
else
PORTD = 0x00;
```

b. Vòng lặp while và do – While

```
while (expression) statement ; // (1)
do statement while (condition); // (2)
```

Chức năng của (1) đơn giản chỉ là lặp lại `statement` khi điều kiện `expression` còn thoả mãn.
Chức năng của (2) hoàn toàn giống vòng lặp `while` chỉ trừ có một điều là điều kiện điều khiển vòng lặp được tính toán sau khi `statement` được thực hiện, vì vậy `statement` sẽ được thực hiện ít nhất một lần ngay cả khi `condition` không bao giờ được thoả mãn.

Ví dụ.

```
int i ;
while (i < 128)
{
PORTD = i;
i = i*2 ;
```

```
}
```

Để có thể lặp vô hạn, ta dùng cấu trúc:

```
While (1)
{
Statement
}
```

c. Vòng lặp for

```
for (initialization; condition; increase) statement;
```

Chức năng chính của nó là lặp lại `statement` chừng nào `condition` còn mang giá trị đúng, như trong vòng lặp `while`. Nhưng thêm vào đó, `for` cung cấp chỗ dành cho lệnh khởi tạo và lệnh tăng. Vì vậy vòng lặp này được thiết kế đặc biệt lặp lại một hành động với một số lần xác định.

1. `initialization` được thực hiện. Nói chung nó đặt một giá khí ban đầu cho biến điều khiển. Lệnh này được thực hiện **chỉ một lần**.
2. `condition` được kiểm tra, nếu nó là đúng vòng lặp tiếp tục còn nếu không vòng lặp kết thúc và `statement` được bỏ qua.
3. `statement` được thực hiện. Nó có thể là một lệnh đơn hoặc là một khối lệnh được bao trong một cặp ngoặc nhọn.
4. Cuối cùng, `increase` được thực hiện để tăng biến điều khiển và vòng lặp quay trở lại bước 2.

Ví dụ.

```
for(int i = 1; i <= 128; i = i*2)
{
PORTD = i ;
}
```

Cấu trúc sau sẽ lặp vô hạn, giống như cấu trúc `while (1)` :

```
for(;;)
{
// Statement
}
```

d. Lệnh rẽ nhánh break và continue

- Sử dụng **break** chúng ta có thể thoát khỏi vòng lặp ngay cả khi điều kiện để nó kết thúc chưa được thoả mãn. Lệnh này có thể được dùng để kết thúc một vòng lặp không xác định hay buộc nó phải kết thúc giữa chừng thay vì kết thúc một cách bình thường.
- Lệnh **continue** làm cho chương trình bỏ qua phần còn lại của vòng lặp và nhảy sang lần lặp tiếp theo.

Ví dụ 1.

```
int n;
  for (n=10; n>0; n--)
{
 PORTD = n ;
  if (n== 7)
     {
       break;
     }
}
```

Chương trình trên sẽ cho PORTD = 10, 9, 8, 7.

Chú ý, nếu sửa lại thứ tự đoạn code trên như sau:

```
int n;
  for (n=10; n>0; n--)
{
 if (n== 7)
   {
    break;
   }
PORTD = n ;
}
```

Thì PORTD = 10, 9, 8.

Ví dụ 2.

```
for (int n=10; n>0; n--)
{
  if (n==5) continue;
  PORTD = n ;
```

```
}
```

Kết quả là PORTD = 10, 9, 8, 7, 6, 4, 3, 2, 1.

Chú ý, nếu sửa lại thứ tự đoạn code trên như sau:

```
for (int n=10; n>0; n--)
{
PORTD = n ;
if (n==5) continue;
}
```

Thì PORTD = 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

e. Lệnh nhảy goto

Lệnh **goto** cho phép nhảy vô điều kiện tới bất kì điểm nào trong chương trình.

Ví dụ.

```
  int n=10;
  loop :
  PORTD = n ;
  n-- ;
  if (n>0) goto loop;
```

PORTD = 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.
loop là nhãn của chương trình, giống cách viết trong hợp ngữ.

Để ý, lệnh `n--` *, lệnh này sẽ giảm n đi 1. Ta có thể viết gọn hai câu lệnh:*

```
PORTD = n ;
n-- ;
```

thành: **`PORTD = n--;`** *lệnh này được hiểu là thực hiện phép gán trước rồi mới giảm n đi 1. Nếu sửa lại thành* **`PORTD = --n ;`** *thì sẽ giảm n đi 1 rồi mới thực hiện phép gán. Tức tương đương với:*

```
n-- ;
PORTD = n ;
```

Lúc này PORTD = 9, 8, 7, 6, 5, 4, 3, 2, 1.
Trường hợp `++n` *và* `n++` *cũng hiểu tương tự, với dấu + chỉ sự tăng lên.*

f. Cấu trúc lựa chọn switch

```
switch (expression) {
  case constant1:
    block of instructions 1
    break;
  case constant2:
    block of instructions 2
    break;
  . . .
  . . .
  . . .
  default:
    default block of instructions
  }
```

Switch hoạt động theo cách sau: `switch` tính biểu thức và kiểm tra xem nó có bằng `constant1` hay không, nếu đúng thì nó thực hiện `block of instructions 1` cho đến khi tìm thấy từ khoá `break`, sau đó nhảy đến phần cuối của cấu trúc lựa chọn `switch`. Còn nếu không, `switch` sẽ kiểm tra xem biểu thức có bằng `constant2` hay không. Nếu đúng nó sẽ thực hiện `block of instructions 2` cho đến khi tìm thấy từ khoá `break`. Cuối cùng, nếu giá trị biểu thức không bằng bất kì hằng nào được chỉ định ở trên (bạn có thể chỉ định bao nhiêu câu lệnh `case` tuỳ thích), chương trình sẽ thực hiện các lệnh trong phần `default` nếu nó tồn tại vì phần này không bắt buộc phải có.

Có sự tương tự giữa lệnh **Switch** và cấu trúc **if – else.**

```
switch (x) {
  case 1:
    PORTD = 0x01;
    break;
  case 2:
    PORTD = 0x02;
    break;
  default:
    PORTD = 0x00;
  }
```

Tương đương với :

```
if (x == 1)
 {
  PORTD = 0x01;
 }
```

```
else if (x == 2)
 {
  PORTD = 0x02;
 }
else
 {
  PORTD = 0x00;
  }
```

g. Chằng Hợp Ngữ Vào Trong Chương Trình C

Để có thể viết hợp ngữ trong chương trình C, ta dùng chỉ thị **#asm** và **#endasm.**
Các thanh ghi R0, R1, R22, R23, R24, R25, R26, R27, R30, R31 có thể sử dụng trong đoạn chương trình hợp ngữ.

Ví dụ.

```
#asm
Sei // cho phép ngắt toàn cục
#endasm
```

```
Nếu chỉ viết trên một dòng thì có thể viết gọn là:
#asm("sei")
```

h. Tổ Chức Bộ Nhớ SRAM

Trình biên dịch phân chia và quản lí bộ nhớ SRAM của AVR như sau (xem ảnh dưới). Để truy xuất trực tiếp tới một địa chỉ nào đó trong các vùng nhớ của AVR ta dùng cách sau, cách này thích hợp khi ta muốn quản lí một khối nhớ cho một chức năng nào đó:

Truy xuất bộ nhớ RAM

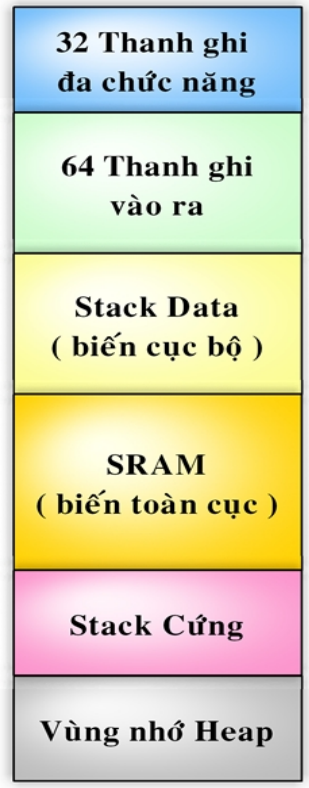
```
unsigned char *Pointer;
Pointer=(unsigned char *) 0x90h ; // truy xuất vào địa chỉ 0x90h của SRAM
```

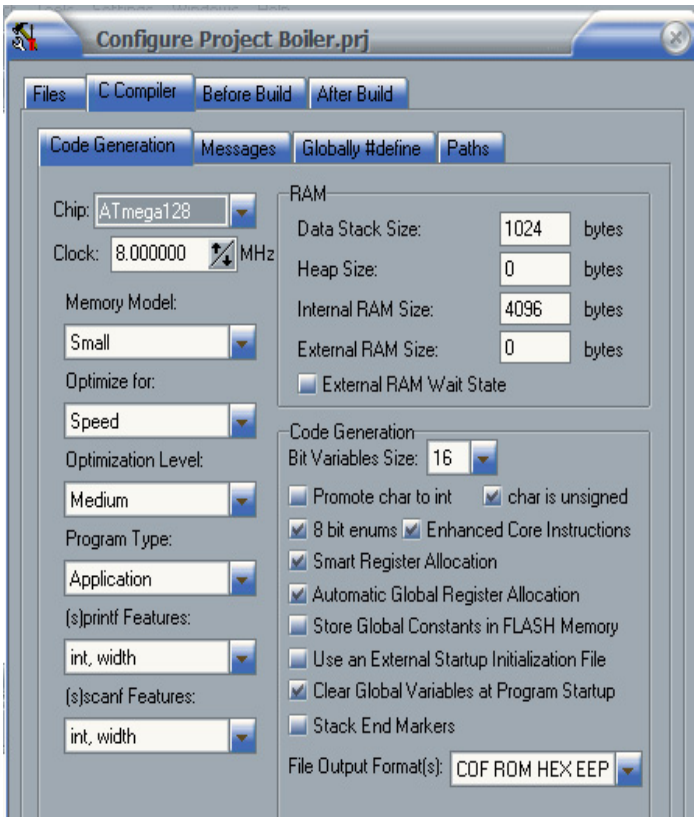
Truy xuất bộ nhớ Flash:

```
flash unsigned char *Pointer;
Pointer=(flash unsigned char *)0x90h ; // truy xuất vào địa chỉ
                                       // 0x90h của flash
```

Truy xuất bộ nhớ Eeprom:

```
eeprom unsigned char *Pointer;
Pointer=(eeprom unsigned char *)0x90h; // truy xuất vào địa chỉ
                                       // 0x90h của eeprom
```





Tùy theo số lượng biến (cục bộ và toàn cục) mà trình biên dịch sẽ phân chia kích thước các vùng nhớ một cách thích hợp. Vùng nhớ heap có thể không có nếu trong chương trình không sử dụng đến. Vùng **Stack cứng** dùng để lưu địa chỉ quay về của hàm, tức các giá trị của con trỏ **SP** (stack pointer).

Như vậy, khác với hợp ngữ chỉ có một vùng stack, là nơi lưu các giá trị quay về của chương trình khi chương trình cần nhảy tới và thực thi một đoạn chương trình con nào đó. Trong C, vùng **data stack** chỉ lưu các biến cục bộ, các tham số truyền vào của hàm,.. Còn vùng **stack cứng** mới lưu địa chỉ quay về của hàm. Vùng nhớ **heap** dùng để cấp phát biến động (dynamic variable).

Kích thước của các vùng nhớ trên có thể dễ dàng lựa chọn trong Codevision AVR. (chọn **Project -> Configure -> C complier**). Riêng phần stack cứng thì trình biên dịch sẽ khởi tạo trong start-up code. (Những người mới học có thể chưa cần quan tâm tới phần này).

PHỤ LỤC:

Giải Thích Các Từ Ngữ Trong Tài Liệu

(1) **Compare Match** [1] : Đây là một chức năng của bộ định thời, theo đó, giá trị của bộ định thời (tức giá trị thanh ghi TCNTn (n=0,..,3)) liên tục được so sánh sánh với giá trị của thanh ghi OCRn (n=0,..,3). Khi hai giá trị này bằng nhau sẽ tạo ra sự thay đổi mức logic ở chân OCn (n=0,..,3). Nhờ đó, ta có thể tạo ra xung PWM ở ngõ ra OCn (n=0,..,3) của vi điều khiển.
(2) **Input capture** [2] : là tên của một chân lối vào (ICn, n=1, 3) của AVR, chân này nối với khối input capture unit, chức năng của khối này là: Khi có sự kiện (rising, falling, level) ở chân input capture thì giá trị của thanh ghi bộ định thời (TCNTn, n=1, 3) sẽ được cập nhật (copy) vào thanh ghi input capture register (ICRn, n=1, 3). Chức năng này dùng để đo khoảng thời gian giữa các sự kiện. Để ý là khi xảy ra sự kiện input capture thì sẽ tạo ra một ngắt, nếu ngắt input capture được cho phép. Ngoài sự kiện input capture tạo ra ở chân ICn, thì ngõ ra của khối analog Comparator (ACO), do được nối với khối input capture unit nên cũng có thể tạo ra sự kiện input capture.
(3) **Falling** [3] : Cạnh xuống của tín hiệu.
(4) **Rising** [4] : Cạnh lên của tín hiệu
(5) **Toggle** [5] : Sự thay đổi mức logic từ 0 lên 1 hoặc từ 1 xuống 0.
(6) **Force Output Compare** [6] : Sự kiện **Compare Match** *không phải* do sự bằng nhau của hai thanh ghi TCNTn (n=0,..,3) và OCRn (n=0,..,3) mà xảy ra do bị ép buộc bằng cách set các bít FOCnX (n=0,1,…X=A,B,C) trong thanh ghi TCCRn.
(7) **Global interrupt** [7] **:** Ngắt toàn cục. Muốn cho phép bất kì một ngắt riêng rẽ nào hoạt động thì cũng phải cho phép ngắt toàn cục trước.
(8) Prescaler [8] : Khối chức năng giúp chia nhỏ giá trị của xung clock.
(9) **Fuse bit** [9] : Bit cầu chì, để cấu hình phần cứng cho vi điều khiển trước khi nó có thể hoạt động.
(10) **Trigger event** [10] : Bất cứ một sự thay đổi logic nào xảy ra trên chân (thường là ngõ vào) của của vi điều khiển thì gọi là một sự kiện trigger.

Tài Liệu Tham Khảo

1. Datasheet 8-bit AVR ATmega128 - © Copyright 2007 Atmel Corporation.
2. CodeVisionAVR Help - © Copyright 1998-2008 by Pavel Haiduc and HP InfoTech.