# UART

## [CẤU TRÚC VÀ CÁC THÀNH PHẦN TRONG](https://nguyenquanicd.blogspot.com/2017/06/cancontroller-area-netwwork-cau-truc-va.html) UART

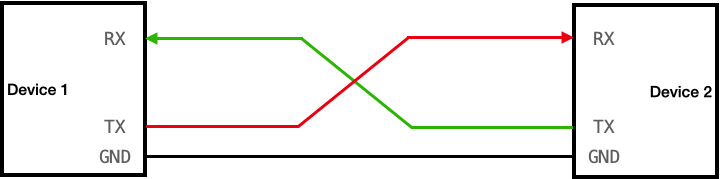
### UART là gì ?

UART (Universal Asynchronous Receiver-Transmitter – Bộ truyền nhận dữ liệu không đồng bộ) là một giao thức truyền thông phần cứng dùng giao tiếp nối tiếp không đồng bộ và có thể cấu hình được tốc độ truyền. Mục đích chính của UART là truyền và nhận dữ liệu nối tiếp.

Chuẩn giao tiếp UART sử dụng 2 dây để truyền và nhận dữ liệu giữa các thiết bị:

1. TX (Transmiter) – Dây truyền dữ liệu
2. RX (Receiver) – Dây nhận dữ liệu

### Cấu trúc phần cứng



**Hình 1.** Cấu trúc phần cứng UART

Chuẩn giao tiếp UART hoạt động theo cơ chế truyền dữ liệu bất đồng bộ, tức là không có một tín hiệu xung clock chung giữa hai thiết bị để đồng bộ việc truyền nhận dữ liệu. Cả hai UART cũng phải được cấu hình để truyền và nhận cùng một cấu trúc gói dữ liệu. Nghĩa là khi thiết lập ở vi điều khiển, các cấu hình tốc độ baud phải giống nhau.

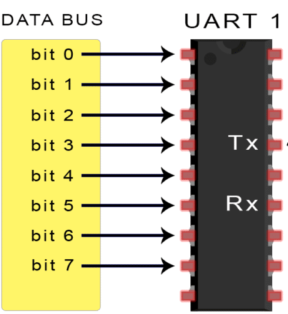
|  |
| --- |
| **Lưu ý**:  GND của hai thiết bị phải được kết nối với nhau. |

|  |  |
| --- | --- |
| Số lượng dây sử dụng | 2 |
| Tốc độ | Từ 9600 bps -> 115200 bps |
| Phương thức truyền dữ liệu | Không đồng bộ |
| Kiểu truyền dữ liệu | Nối tiếp |
| Số lượng Master (thiết bị chủ) | 1 |
| Số lượng Slave (thiết bị tớ) | 1 |

|  |
| --- |
| **Lưu ý**:  Bps: là tốc độ truyền dữ liệu được định nghĩa là khối lượng truyền dữ liệu trên mỗi đơn vị thời gian. Ký hiệu: bit / giây (bit/s) hoặc byte/ giây (B/s) |

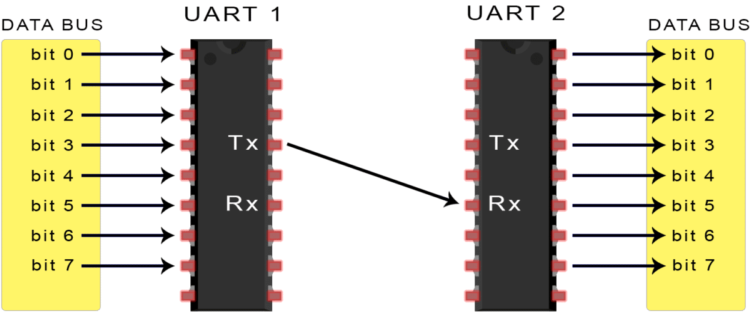
## CẤU TRÚC KHUNG TRUYỀN DỮ LIỆU VÀ KHUNG NHẬN DỮ LIỆU

### Nguyên lý hoạt động



**Hình 1.2:** Truyền dữ liệu từ DATA BUS đến UART

Dữ liệu sẽ được chuyển từ Bus dữ liệu đến UART1 truyền ở dạng song song. Sau khi nhận được dữ liệu, lúc này nó sẽ thêm một bit start, một bit chẵn lẻ và một bit stop để tạo thành gói dữ liệu (packet data).



Gói dữ liệu được truyền ra từng bit nối tiếp tại chân Tx của UART 1 và UART 2 sẽ nhận gói dữ liệu từng bit tại chân Rx của nó. Sau khi nhận xong dữ liệu, UART 2 chuyển đổi dữ liệu trở lại dạng song song và loại bỏ bit start, bit chẵn lẻ và bit stop.

Uart truyền dữ liệu nối tiếp, theo 1 trong 3 chế độ:

* Simplex: Chỉ tiến hành giao tiếp một chiều.
* Half duplex: Dữ liệu sẽ đi theo một hướng tại 1 thời điểm.
* Full duplex: Thực hiện truyền và nhận dữ liệu từ master và slave tại 1 thời điểm.

#### Full duplex

Full duplex hay còn gọi là Song Công: Là quá trình truyền ***và*** nhận dữ liệu từ master và slave tại 1 thời điểm. Giao thức UART có 2 đường truyền dữ liệu riêng biệt, cho phép dữ liệu truyền nhận theo 2 hướng cùng một thời điểm.

A diagram of a link between two lines

Description automatically generated

**Hình 1.4:** Quá trình Full duplex

#### Half duplex

Half duplex hay còn gọi là Bán Song Công: Là quá trình truyền ***hoặc*** nhận dữ liệu từ master và slave tại 1 thời điểm.

A diagram of a link between two lines

Description automatically generated

**Hình 1.4:** Quá trình Half duplex

#### Simplex:

Simplex hay còn gọi là Đơn Công: Là quá trình truyền dữ liệu theo một hướng duy nhất trên một đường truyền đã xác định.

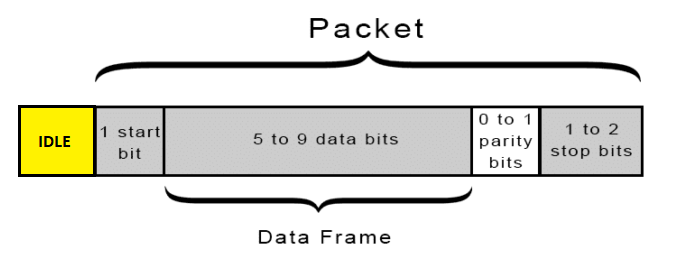
A close-up of a line

Description automatically generated

**Hình 1.5:** Quá trình Simplex

### Khung truyền dữ liệu

Dữ liệu được truyền trong giao tiếp UART được tổ chức thành các gói (Packets). Mỗi Packets chứa 1 **bit** **Start**, 5 đến 9 bit dữ liệu (tùy thuộc vào UART), 1 **bit** **Parity**và 1 hoặc 2 **bit Stop.**



**Hình 1.6:** Khung truyền của giao tiếp UART

* **IDLE** (Trạng thái bus rảnh)
* Là không có sự truyền/nhận dữ liệu trên bus.
* Giá trị của IDLE sẽ luôn luôn ở mức cao để phát hiện điều kiện bắt đầu của **Start bit**
* **Start bit** (Bit khởi đầu):
* Đường truyền dữ liệu trong giao tiếp UART thường được giữ ở mức điện áp cao(IDLE) khi nó không truyền dữ liệu.
* Để bắt đầu truyền dữ liệu, UART truyền sẽ kéo đường truyền từ mức cao xuống mức thấp trong một chu kỳ đồng hồ.
* Khi UART 2 phát hiện sự chuyển đổi điện áp cao xuống thấp, nó bắt đầu đọc các bit trong khung dữ liệu ở tần số của tốc độ truyền (Baud rate).
* **Data bits** (Bit dữ liệu) [0-8]:
* Khung dữ liệu chứa dữ liệu thực tế đang được truyền. Nó có thể dài từ 5 bit đến 8 bit nếu sử dụng **bit Parity**(bit chẵn lẻ).
* Nếu không sử dụng **bit** **Parity**, khung dữ liệu có thể dài 9 bit. Trong hầu hết các trường hợp, dữ liệu được gửi với [**bit LSB**](https://vi.wikipedia.org/wiki/Byte)(bit có trọng số thấp nhất) trước tiên.
* Nếu muốn gửi dữ liệu thì phải gửi 1 chuỗi bit. Tại 1 thời điểm chỉ gửi được 1 bit data. Nếu gửi 1 byte thì gửi 8 lần.
* **Parity bit** (Bit chẵn lẻ)
  + Dùng để kiểm tra dữ liệu truyền có đúng không (một cách tương đối). Có 2 loại parity là parity chẵn (even parity) và parity lẻ (odd parity).
  + Nếu **bit Parity** là 0 (chẵn), thì tổng các bit 1 trong khung dữ liệu phải là một số chẵn.
  + Nếu **bit Parity** là 1 (lẻ), thì tổng các bit 1 trong khung dữ liệu sẽ là một số lẻ.
  + Parity bit không phải là bit bắt buộc và vì thế nó có thể có hoặc không.

|  |
| --- |
| **Thí dụ:**  Nếu dữ liệu của bạn là 10111011 nhị phân, có tất cả 6 bit có giá trị “1” trong dữ liệu này, nếu quy định parity chẵn được dùng, bit parity sẽ mang giá trị 0 để đảm bảo tổng các số “1” là số chẵn (6 số 1). Nếu parity lẻ được yêu cầu thì giá trị của parity bit là 1. Sau khi truyền chuỗi dữ liệu kèm theo cả bit parity trên, bên nhận thu được và kiểm tra lại tổng số số “1” (bao gồm cả bit parity), nếu vi phạm quy định parity đã đặt trước thì ta khẳng định là dữ liệu nhận được là sai, còn nếu không vi phạm thì cũng không khẳng định được điều gì (mang tính tương đối). |

* (Stop bit) Bit dừng
* Để báo hiệu sự kết thúc của gói dữ liệu, UART gửi sẽ điều khiển đường truyền dữ liệu từ điện áp thấp đến điện áp cao trong ít nhất hai khoảng thời gian bit.

### Khung nhận dữ liệu của giao thức UART

A diagram of a data analysis

Description automatically generated

Bên nhận sẽ đọc liên tục đường **IDLE** của RX. Khi phát hiện sự thay đổi logic từ 1 xuống 0 thì **Start bit** được đọc và bắt đầu nhận dữ liệu. **LSB** nhận trước.

A diagram of a diagram

Description automatically generated

Delay ½ chu kì và tiếp tục delay 1 chu kì và bắt đầu lấy mẫu. Bởi vì dữ liệu ở giữa ổn định nhất. Cứ thế sau 1 chu kì lại đọc tiếp.

**Chú ý:** Truyền nhận phải cùng tốc độ Baus bởi vì truyền bất đồng bộ.

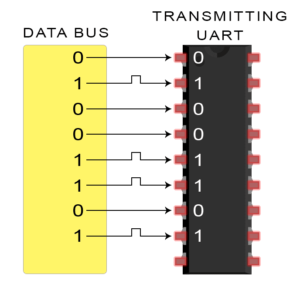
A table with numbers and a few lines

Description automatically generated with medium confidence

### ****Các bước truyền dữ liệu trong chuẩn giao tiếp UART****

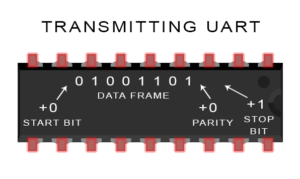
* ***Bước 1:*** UART truyền nhận dữ liệu song song từ bus dữ liệu

Dữ liệu sẽ là song song ở đầu vào. Sau đó sẽ được chuyển thành nối tiếp để truyền trên dây TX. Tốc độ baud thiết lập từ trước ở phần cấu hình.



**Hình 1.7:** Đầu vào của UART (truyền song song)

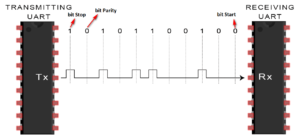
* **Bước 2:** UART truyền thêm bit thành phần như: Start bit, Parity bit và Stop bit vào khung truyền để chuẩn bị cho việc truyền dữ liệu.



**Hình 1.7:** Đưa thêm các bit điều kiện vào khung truyền

* **Bước 3:** Toàn bộ gói tin được gửi nối tiếp từ UART truyền đến UART nhận. UART nhận lấy mẫu đường dữ liệu ở tốc độ truyền được định cấu hình trước:

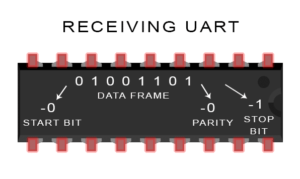
Trong quá trình này các bit trong khung truyền lần lượt được đẩy lên đường dây TX của UART1 (truyền) và đưa tới dây RX của UART2 (nhận).



**Hình 1.7:** Quá trình truyền dữ liệu trong giao tiếp UART

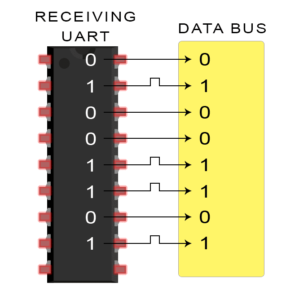
* **Bước 4**: UART nhận loại bỏ bit bắt đầu, bit chẵn lẻ và bit dừng khỏi khung dữ liệu:

Quá trình này sẽ bỏ các bit điều kiện đi, sau đó chỉ còn Data Frame (khung dữ liệu). Cuối cùng dữ liệu cần truyền sẽ tiếp tục được xử lý trong Slave.



**Hình 1.7:** Loại bỏ các bit điều kiện trong khung truyền

* **Bước 5:** UART nhận chuyển đổi dữ liệu nối tiếp trở lại thành song song và chuyển nó đến data Bus



**Hình 1.7:** Chuyển dữ liệu nối tiếp trở lại song song

Lúc này việc truyền dữ liệu qua giao tiếp UART đã hoàn thành.

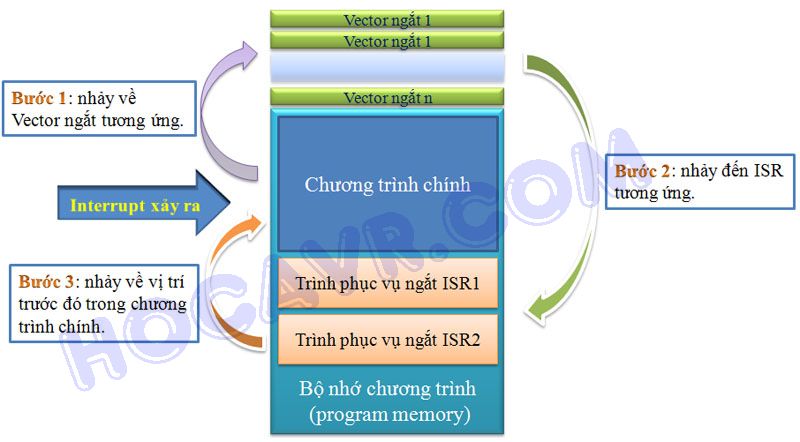
# NGẮT NGOÀI

### ****Ngắt trên AVR.****

**Interrupts**, thường được gọi là ngắt, là một tín hiệu khẩn cấp gởi đến bộ xử lí, yêu cầu bộ xử lí tạm ngừng tức khắc các hoạt động hiện tại để “nhảy” đến một nơi khác thực hiện một nhiệm vụ khẩn cấp nào đó, nhiệm vụ này gọi là trình phục vụ ngắt – isr (interrupt service routine ). Sau khi kết thúc nhiệm vụ trong isr, bộ đếm chương trình sẽ được trả về giá trị trước đó để bộ xử lí quay về thực hiện tiếp các nhiệm vụ còn dang dở. Như vậy, ngắt có mức độ ưu tiên xử lí cao nhất, ngắt thường được dùng để xử lí các sự kiện bất ngờ nhưng không tốn quá nhiều thời gian. Các tín hiệu dẫn đến ngắt có thể xuất phát từ các thiết bị bên trong chip (ngắt báo bộ đếm timer/counter tràn, ngắt báo quá trình gởi dữ liệu bằng RS232 kết thúc…) hay do các tác nhân bên ngoài (ngắt báo có 1 button được nhấn, ngắt báo có 1 gói dữ liệu đã được nhận…).

Ngắt là một trong 2 kỹ thuật “bắt” sự kiện cơ bản là hỏi vòng (Polling) và ngắt. Hãy tưởng tượng bạn cần thiết kế một mạch điều khiển hoàn chỉnh thực hiện rất nhiều nhiệm vụ bao gồm nhận thông tin từ người dùng qua các button hay keypad (hoặc keyboard), nhận tín hiệu từ cảm biến, xử lí thông tin, xuất tín hiệu điều khiển, hiển thị thông tin trạng thái lên các LCD…(bạn hoàn toàn có thể làm được với AVR), rõ ràng trong các nhiệm vụ này việc nhận thông tin người dùng (start, stop, setup, change,…) rất hiếm xảy ra (so với các nhiệm vụ khác) nhưng lại rất “khẩn cấp”, được ưu tiên hàng đầu. Nếu dùng Polling nghĩa là bạn cần viết 1 đoạn chương trình chuyên thăm dò trạng thái của các button (tôi tạm gọi đoạn chương trình đó là Input()) và bạn phải chèn đoạn chương trình Input() này vào rất nhiều vị trí trong chương trình chính để tránh trường hợp bỏ sót lệnh từ người dùng, điều này thật lãng phí thời gian thực thi. Giải pháp cho vấn đề này là sử dụng ngắt, bằng cách kết nối các button với đường ngắt của chip và sử dụng chương trình Input() làm trình phục vụ ngắt - isr của ngắt đó, bạn không cần phải chèn Input() trong lúc đang thực thi và vì thế không tốn thời gian cho nó, Input() chỉ được gọi khi người dùng nhấn các button. Đó là ý tưởng sử dụng ngắt.

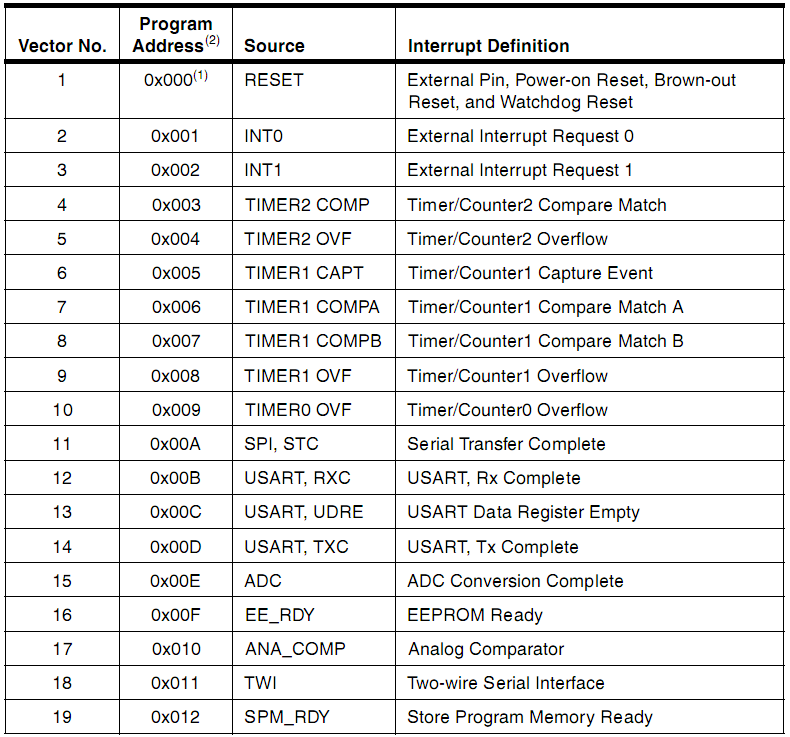
Hình 1 minh họa cách tổ chức ngắt thông thường trong các chip AVR. Số lượng ngắt trên mỗi dòng chip là khác nhau, ứng với mỗi ngắt sẽ có vector ngắt, vector ngắt là các thanh ghi có địa chỉ cố định được định nghĩa trước nằm trong phần đầu của bộ nhớ chương trình. Ví dụ vector ngắt ngoài 0 (external interrupt 0) của chip atmega8 có địa chỉ là 0x001 (theo datasheet từ Atmel). Trong lúc chương trình chính đang thực thi, nếu có một sự thay đổi dẫn đến ngắt xảy ra ở chân INT0 (chân 4), bộ đếm chương trình (Program Counter) nhảy đến địa chỉ 0x009, giả sử ngay tại địa chỉ 0x001 chúng ta có đặt 1 lệnh RJMP đến một trình phục vụ ngắt (IRS1 chẳng hạn), một lần nữa bộ đếm chương trình nhảy đến IRS1 để thực thi trình phục vụ ngắt, kết thúc ISR1, bộ đếm chương trình lại quay về vị trí trước đó trong chương trình chính, quá trình ngắt kết thúc. Không mang tính bắt buộc nhưng tôi khuyên bạn nên tổ chức chương trình ngắt theo cách này để tránh những lỗi liên quan đến địa chỉ chương trình.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjOuSP420YRtY9XSfajdOiqPzke39SBYXVnktu8zQRhEWZM-rS7xc5AUN_YvXuFtfIbSaIfhemxBnB8KjXCNRhI9DlkGpV7AkGI-g-AbgrMuuoSiLluXO7S7WBNSOfXUZ0ahzwqUgU0QqPo/s1600/isr.jpg)

Hình 1. Ngắt.

     Bảng 1 tóm tắt các vector ngắt có trên chip atmega8, cho các chip khác bạn hãy tham khảo datasheet để biết thêm.

Bảng 1 các vector ngắt và Reset trên chip Atmega8.

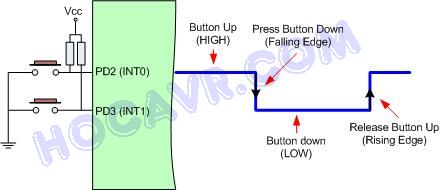


### ****II. Ngắt ngoài (External Interrupt).****

      Phần này tôi dành giới thiệu các bạn cách cài đặt và sử dụng ngắt ngoài vì đây là loại ngắt duy nhất độc lập với các thiết bị của chip, các ngắt khác thường gắn với hoạt động của 1 thiết bị nào đó như Timer/Counter, giao tiếp nối tiếp USART, chuyển đổi ADC…chúng ta sẽ khảo sát cụ thể khi tìm hiểu về hoạt động của các thiết bị này.

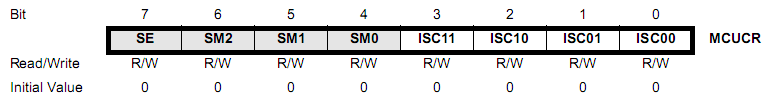
      Ngắt ngoài là cách rất hiệu quả để thực hiện giao tiếp giữa người dùng và chip. Trên chip atmega8 có 2 ngắt ngoài có tên là INT0 và INT1 tương ứng 2 chân số 4 (PD2) và số 5 (PD3). Như tôi đã đề cập trong bài AVR2, khi làm việc với các thiết bị ngoại vi của AVR, hầu như chúng ta chỉ thao tác trên các thanh ghi chức năng đặc biệt - SFR (Special Function Registers) trên vùng nhớ IO, mỗi thiết bị bao gồm một tập hợp các thanh ghi điều khiển, trạng thái, ngắt…khác nhau, điều này đồng nghĩa chúng ta phải nhớ tất cả các thanh ghi của AVR. Lúc này datasheet phát huy tác dụng, bạn phải nhanh chóng download file datasheet của chip mình đang sử dụng, có rất nhiều nơi để download như tại [www.atmel.com](http://www.atmel.com/)hay trên các trang web chuyên cung cấp IC datasheet miễn phí ([www.alldatasheet.com](http://www.alldatasheet.com/) là 1 ví dụ). Quay về với ngắt ngoài, có 3 thanh ghi liên quan đến ngắt ngoài đó là MCUCR, GICR và GIFR. Cụ thể các thanh ghi được trình bày bên dưới.

**Thanh ghi điều khiển MCU – MCUCR** (MCU Control Register) là thanh ghi xác lập chế độ ngắt cho ngắt ngoài, quan sát hình 2 trước khi tìm hiểu thanh ghi này.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj_1icLNEnkbcjid-Lw16RG6wBuEVPGc3rPI0SxRMD-_SaE1SaWA186SvJGg7aayim9YrhDz5Z-iuzAh72SlcTTdHjB2N8iV4ZapcsQzLCevuwmj9I3pW6Yd2rq4BQrUBgRLiZtnBm3Mmt8/s1600/button.jpg)

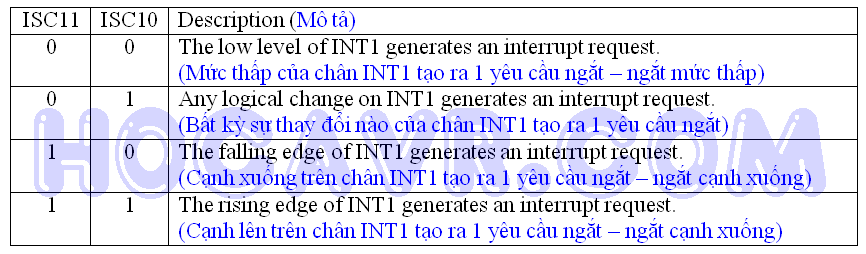
Hình 2. Kết nối ngắt ngoài cho atmega8.

      Giả sử chúng ta kết nối các ngắt ngoài trên AVR mega8 như phía trái hình 2, các button dùng tạo ra các ngắt. Có 4 khả năng (tạm gọi là các MODES) có thể xảy ra khi chúng ta nhấn và thả các button. Nếu không nhấn, trạng thái các chân INT là HIGH do điện trở kéo lên, khi vừa nhấn 1 button, sẽ có chuyển trạng thái từ HIGH sang LOW, chúng ta gọi là cạnh xuống - **Falling Edge**, khi button được nhấn và giữ, trạng thái các chân INT được xác định là LOW và cuối cùng khi thả các button, trạng thái chuyển từ LOW sang HIGH, gọi là cạnh lên – **Rising Edge**.  Trong những trường hợp cụ thể, 1 trong 4 MODES trên đều hữu ích, ví dụ trong các ứng dụng đếm xung (đếm encoder của servo motor chẳng hạn) thì 2 MODE “cạnh” phải được dùng. Thanh ghi MCUCR chứa các bits cho phép chúng ta chọn 1 trong 4 MODE trên cho các ngắt ngoài. Dưới đây là cấu trúc thanh ghi MCUCR được trích ra từ datasheet của chip atmega8.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgWWX4HUNm0ydO_VFsYgtiLd_vZDABW3LA1TFtqaZ1P0GwvDWoTJP3PTb8KITrc7ukPV58bk6mcOUTKyiYi47KB23K8FZpio5bQiVkACt0dOSmBYYpg9kRJgHGFIFqSoBXCabRk0hVNwTr7/s1600/image045.png)

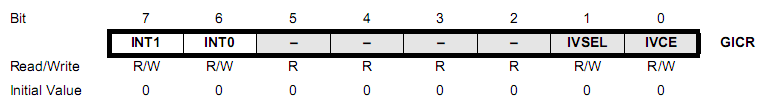
      MCUCR là một thanh ghi 8 bit nhưng đối với hoạt động ngắt ngoài, chúng ta chỉ quan tâm đến 4 bit thấp của nó (4 bit cao dùng cho Power manager và Sleep Mode). Bốn bit thấp là các bit Interrupt Sense Control (ISC) trong đó 2 bit ISC11:ISC10 dùng cho INT1 và 2 bit ISC01:ISC00 dùng cho INT0. Hãy nhìn vào bảng tóm tắt bên dưới để biết chức năng của các bit trên, đây là bảng “chân trị” của 2 bit ISC11, ISC10. Bảng chân trị cho các bit ISC01, ISC00 hoàn toàn tương tự.

Bảng 2: INT1 Sense Control

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj0ic-2qzp1TwaUnVZybAGIRIF_kQroRDlpJfc13bVP4j0DGTYIMQDfyZaOKa6OBMUuqo4Tok5o415xZsmTwU0LNSAzTAgF3A40eIkV1LAJUrjnscbm6IK61lWumEAwV1SKQf2DXRY-Pekv/s1600/image047.png)

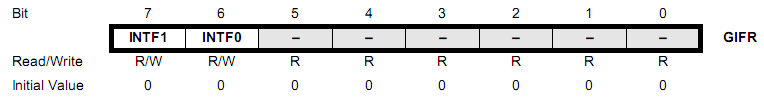
      Thật dễ dàng để hiểu chức năng của các bit Sense Control, ví dụ bạn muốn set cho INT1 là ngắt cạnh xuống (Falling Edge) trong khi INT0 là ngắt cạnh lên (Rising Edge), hãy đặt dòng lệnh MCUCR =0x0B (0x0B = 00001011 nhị phân) trong chương trình của bạn.

      Thanh ghi điều khiển ngắt chung – GICR (General Interrupt Control Register) (chú ý trên các chip AVR cũ, như các chip AT90Sxxxx, thanh ghi này có tên là thanh ghi mặt nạ ngắt thông thường GIMSK, bạn tham khảo thêm datasheet của các chip này nếu cần sử dụng đến). GICR cũng là 1 thanh ghi 8 bit nhưng chỉ có 2 bit cao (bit 6 và bit 7) là được sử dụng cho điều khiển ngắt, cấu trúc thanh ghi như bên dưới (trích datasheet).

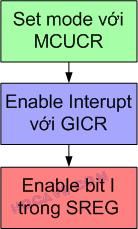
[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgmt336FoInHT4oH3_qBaFIEswKNVOBJ2_jkIOAGHhyFtzyaYSCe4GHbPM3CYWkmVPJpSnwDsC7gkqPFG1kJMNoLZPkxsljSqmZuNZ-QxPHWyngWy391w5k86EGOeqkLHqtTye0V_w6qOUN/s1600/image049.png)

      Bit 7 – INT1 gọi là bit cho phép ngắt 1(Interrupt Enable), set bit này bằng 1 nghĩa bạn cho phép ngắt INT1 hoạt động, tương tự, bit INT0 điều khiển ngắt INT0.

      Thanh ghi cờ ngắt chung – GIFR (General Interrupt Flag Register) có 2 bit INTF1 và INTF0 là các bit trạng thái (hay bit cờ - Flag) của 2 ngắt INT1 và INT0. Nếu có 1 sự kiện ngắt phù hợp xảy ra trên chân INT1, bit INTF1 được tự động set bằng 1 (tương tự cho trường hợp của INTF0), chúng ta có thể sử dụng các bit này để nhận ra các ngắt, tuy nhiên điều này là không cần thiết nếu chúng ta cho phép ngắt tự động, vì vậy thanh ghi này thường không được quan tâm khi lập trình ngắt ngoài. Cấu trúc thanh ghi GIFR được trình bày trong hình ngay bên dưới.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgUrLyyaNKAyLmgPPpPf6NOIMSh5uheLTIIL2TqvdGtoENmkPUeCr3cWeslMtz-bgiQZIk0J17mhdtdOvjrhaN3Oym9W2PCcNkVZd7HUwsVbCdZ4djAHak_jH3YS0tBMnkkslGcdBEMFYh7/s1600/image051.png)

      Sau khi đã xác lập các bit sẵn sàng cho các ngắt ngoài, **việc sau cùng chúng ta cần làm là set bit I, tức bit cho phép ngắt toàn cục**, trong thanh ghi trạng thái chung của chip (thanh ghi SREG, xem lại bài AVR2). **Một chú ý khác là vì các chân PD2, PD3 là các chân ngắt nên bạn phải set các chân này là Input** (set thanh ghi DDRD). Quá trình thiết lập ngắt ngoài được trình bày trong hình 10.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEi91WhWEeYWdUEGhlXplis_j0CcIBFucMW1OhHRQpTmkU14rUHJz_vqYTm2yLvreSQNBUaKC_7rw8qABArFA3uGxs8GPPHNLGhmAsmu6vhJyWW1aU8BCZ7JO3MiGBYW2FBAO86fcXmCfw-F/s1600/order.jpg)

Hình 3. Thiết lập ngắt ngoài.

**Ngắt ngoài với ASM:**Dưới đây tôi trình bày cách viết chương trình  sử dụng ngắt ngoài bằng ngôn ngữ ASM, đối với các ngắt khác bạn chỉ cần thêm các DIRECTIVE để định vị các vector ngắt tương ứng và viết chương trình phục vụ ngắt tương ứng.

List 1. Ngắt với ASM.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 | .CSEG .INCLUDE "M8DEF.INC" .ORG 0x000 ; Định vị vị trí đầu tiên             RJMP BATDAU  .ORG 0x001; Định vị vector ngắt ngoài 0 - INT0 (xem bảng vector)       RJMP INT0\_ISR ; Nhảy đến INT0\_ISR nếu có ngắt INT0 xảy ra .ORG 0x002 ; Định vị vector ngắt ngoài 1 – INT1 (xem bảng vector)       RJMP INT1\_ISR ; Nhảy đến INT1\_ISR nếu có ngắt INT1 xảy ra  ;Tương tự, định vị các vector ngắt khác ở đây……………….. ;………………………………………………………………..  .ORG 0x020 ; Định vị chương trình chínhBATDAU: ; khởi tạo Stack       LDI R16, HIGH(RAMEND)       LDI R17, LOW(RAMEND)       OUT SPH, R16       OUT SPL, R17  ; set chân PD2 và PD3 như các chân input             LDI R16, 0Bxxxx00xx      ; x là trạng thái do bạn tự chọn, 0 hoặc 1             OUT DDRD, R16            ; PD2 và PD3 là input             LDI R16, 0Bxxxx11xx      ; x là trạng thái do bạn tự chọn, 0 hoặc 1             OUT PORTD, R16          ; mắc điện trở kéo lên cho PD2, PD3  ; khởi động  ngắt       LDI R16, $0B  ; $0B=00001011,  INT1: ngắt xuống, INT0: ngắt cạnh lên             OUT MCUCR, R16  ; xuất giá trị điều khiển  ra thanh ghi MCUCR             LDI R16, $C0      ;$C0=11000000: Enable INT1 và INT0       OUT GICR, R16 ;xuất giá trị điều khiển  ra thanh ghi  GICR       SEI ;set bit cho phép ngắt toàn cục ; Chương trình chính MAIN: ;các công việc mà chương trình chính cần thực hiện……………… ;…………………………………………………………………….       RJMP MAIN  ;và đây là định nghĩa trình phục vụ ngắt INT0\_ISR………………… INT0\_ISR: ; các công việc cần thực hiện khi có ngắt …………………… ;……………………………………………………………….       RETI ; phải dùng lệnh RETI để quay về chương trình chính  ;và đây là định nghĩa trình phục vụ ngắt INT1\_ISR………………… INT1\_ISR: ; các công việc cần thực hiện khi có ngắt …………………… ;……………………………………………………………….       RETI ; phải dùng lệnh RETI để quay về chương trình chính |

      Bạn thấy các các ngắt được định vị nằm giữa vị trí 0x0000, khi mới khởi động, tại ví trí 0x000 là lệnh “RJMP BATDAU”, như thế các lệnh RJMP tại các vector ngắt và các ISR đều không được thực hiện, chúng chỉ được thực hiện một cách tự động khi có ngắt.

**Ngắt ngoài với C:** Avr-libc hỗ trợ một thư viện hàm cho ngắt khá hoàn hảo, để sử dụng ngắt trong chương trình viết bằng C (avr-gcc) bạn chỉ cần include file “interrupt.h” nằm trong thư mục con “avr” là xong. file header interrupt.h chứa định nghĩa các hàm và phương thức phục vụ cho viết trình phục vụ ngắt, các vector ngắt không được định nghĩa trong file này mà trong file iom8.h (cho atmega8). Nếu bạn vô tình tìm thấy 1 chương trình ngắt nào đó không include file interrupt.h mà include file signal.h thì bạn đừng ngạc nhiên, đó là cách viết cũ trong avr-gcc, thật ra bạn hoàn toàn có thể sử dụng cách viết cũ vì các phiên bản mới của avr-libc (đi cùng với các bản WinAVR mới) vẫn hỗ trợ cách viết này nhưng không khuyên khích bạn dùng.

       Trong C, các trình phục vụ ngắt có dạng là **ISR(vector\_name).** Trong các phiên bản cũ trình phục vụ ngắt có tên SIGNAL(vector\_name), nhưng cũng như file header signal.h, cách viết này vẫn được hỗ trợ trong phiên bản mới nhưng không được khuyến khích.

List 2. Ngắt với C.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | #include <avr/interrupt.h>  ISR (vector\_name) { //user code here } |

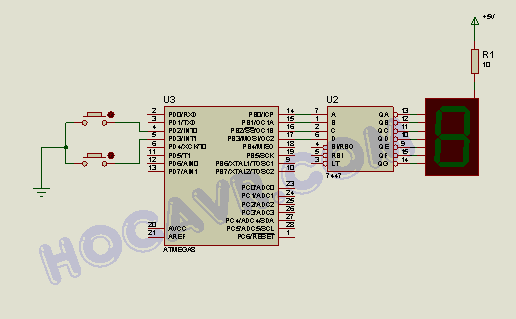
      Trong đó vector\_name là tên của các vector ngắt định nghĩa sẵn avr-libc, ISR là tên bắt buộc, bạn không được dùng các tên khác tùy ỳ (nhưng có thể dùng SIGNAL như đã trình bày ở trên). Đặc biệt, bạn có thể đặt ISR ở trước hoặc sau chương trình chính đều không ảnh hưởng vì thật ra, đã có khá nhiều “công đoạn” được thực hiện khi bạn gọi ISR (nhưng bạn không thấy và cũng không cần quan tâm). ISR luôn được trình biên dịch đặt ở ngoài vùng vector ngắt như cách chúng ta thực hiện trong ASM, như thế một chương trình sử dụng nhiều loại ngắt sẽ phải có số lượng trình ISR tương ứng nhưng với vector\_name khác nhau, mỗi khi có ngắt xảy ra, tùy thuộc vào giá trị của vector\_name mà 1 trong các trình ISR được thực thi. Đối với các vector\_name, để biết được vector\_name cho mỗi loại ngắt, bạn cần tham khảo tài liệu “avr-libc manual”. Bảng 10 tóm tắt các vector\_name của một số ngắt thông dụng trên atmega8, bạn chú ý rằng các vector\_name trong avr-libc được định nghĩa rất khác nhau cho từng loại chip, bạn nhất thiết  phải sử dụng tài liệu “avr-libc manual” để biết chính xác các vector\_name cho loại chip mà bạn đang dùng.

Bảng 3: vector\_name cho atmega8.

|  |  |  |
| --- | --- | --- |
| **Vector name** | **Old vector name** | **Description** |
| ADC\_vect | SIG\_ADC | ADC Conversion Complete |
| ANA\_COMP\_vect | SIG\_COMPARATOR | Analog Comparator |
| EE\_RDY\_vect | SIG\_EEPROM\_READY | EEPROM Ready |
| INT0\_vect | SIG\_INTERRUPT0 | External Interrupt 0 |
| INT1\_vect | SIG\_INTERRUPT1 | External Interrupt Request 1 |
| SPI\_STC\_vect | SIG\_SPI | Serial Transfer Complete |
| SPM\_RDY\_vect | SIG\_SPM\_READY | Store Program Memory Ready |
| TIMER0\_OVF\_vect | SIG\_OVERFLOW0 | Timer/Counter0 Overflow |
| TIMER1\_CAPT\_vect | SIG\_INPUT\_CAPTURE1 | Timer/Counter Capture Event |
| TIMER1\_COMPA\_vect | SIG\_OUTPUT\_COMPARE1A | Timer/Counter1 Compare Match A |
| TIMER1\_COMPB\_vect | SIG\_OUTPUT\_COMPARE1B | Timer/Counter1 Compare MatchB |
| TIMER1\_OVF\_vect | SIG\_OVERFLOW1 | Timer/Counter1 Overflow |
| TIMER2\_COMP\_vect | SIG\_OUTPUT\_COMPARE2 | Timer/Counter2 Compare Match |
| TIMER2\_OVF\_vect | SIG\_OVERFLOW2 | Timer/Counter2 Overflow |
| TWI\_vect | SIG\_2WIRE\_SERIAL | 2-wire Serial Interface |
| USART3\_UDRE\_vect | SIG\_USART3\_DATA | USART3 Data register Empty |

**III. Ví dụ ngắt ngoài với C.**

      Để thực hiện ví dụ sử dụng ngắt ngoài bằng C, tôi sẽ viết lại chương trình ví dụ của bài "[cấu trúc AVR](http://www.hocavr.com/index.php/lectures/cautrucavr" \t "_blank)" nhưng bằng ngôn ngữ C và sử dụng ngắt. Trong chương trình ví dụ của bài AVR2, chúng ta thực hiện việc đếm lên và đếm xuống dùng 2 button, chúng ta sẽ vẫn thực hiện trên ý tưởng này nhưng có chút thay đổi trong kết nối, trước hết bạn vẽ 1 mạch điện mô phỏng trong Proteus như hình 4.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjw4VN6wYFfey6Fz0ki1CrCEZ6vqRdabvri-r8687je_sl0YN1MZadA4aOrtO4U7_4gzA6uJsBb3e5TZbvOYbAfjgc08ZLJGNzrFEjoLFmt7HBPWP52tGAK71gs5wdLFlv29TWeGrk5Fa-m/s1600/image056.gif)

Hình 4. Mạch điện mô phỏng ngắt.

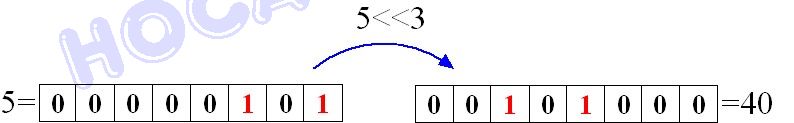
      Kết nối button đếm lên với ngắt INT0, button đếm xuống với INT1, PORTB được chọn làm PORT xuất. Hãy chạy [Programmer Notepad](http://www.hocavr.com/index.php/software/winavr),  tạo 1 Project mới tên AVR2-INT, type đoạn code bên dưới vào 1 file new và lưu với tên main.c, add file này vào Project của bạn, sau đó tạo một Makefile cho Project.

List 3. ví dụ ngắt ngoài bằng C.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 | #include <avr/io.h> #include <avr/interrupt.h> #include <avr/delay.h>  volatile int8\_t  val=0;    //khai báo 1 biến val 8 bit, có dấu và giá trị khởi tạo bằng 0. int main(void){      DDRD=0x00;     //khai báo PORTD là Input để sử dụng 2 chân ngắt.     PORTD=0xFF;  //sử dụng điện trở nội kéo lên.     DDRB=0xFF;    //PORTB là Output để xuất LED 7 đoạn        MCUCR|=(1<<ISC11)|(1<<ISC01); //cả 2 ngắt là ngắt cạnh xuống      GICR    |=(1<<INT1)|(1<<INT0);    //cho phép 2 ngắt hoạt động     sei();                                            //set bit I cho phép ngắt toàn cục        DDRC=0xFF;                                   //PORTC là Output      while (1){                                         //vòng lặp vô tận            PORTC++;                                //quét PORTC         \_delay\_loop\_2(60000);     }     return 0; }  //Trình phục vụ ngắt của  INT0 ISR(INT0\_vect){     val++;                                       //nếu có ngắt INT0 xảy ra, tăng val thêm 1     if (val>9) val=0;                        //giới hạn không vượt quá 9     PORTB=val; }  //Trình phục vụ ngắt của  INT1 ISR(INT1\_vect){     val--;                                        //nếu có ngắt INT1 xảy ra, giảm val đi 1     if (val<0) val=9;                       //giới hạn không nhỏ hơn 0     PORTB=val; } |

      Có lẽ đoạn code này khá dễ hiểu nếu các bạn theo dõi từ đầu bài học, tôi chỉ giải thích những nét cơ bản và “mới”. Ý tưởng là chúng ta sử dụng 1 biến tạm 8 bit, có dấu để lưu giá trị đếm, tên biến val,  mỗi khi có ngắt trên chân INT0, tăng val 1 đơn vị và ngược lại khi có ngắt trên INT1, giảm val đi 1, đó là nội dung của 2 trình phục vụ ngắt. Trong chương trình chính, trước hết chúng ta thực hiện việc xác lập hoạt động cho 2 ngắt, sau đó đưa chương trình vào 1 vòng lặp vô tận while(1), PORTC được dùng để kiểm tra rằng chương trình trong vòng lặp vô tận vẫn đang hoạt động. Có lẽ phần khó hiểu nhất trong đoạn code là cách mà tôi dùng để khai báo cho 2 thanh ghi điều khiển ngắt MCUCR và GICR.

      Nếu xem lại bảng tóm tắt các toán tử của C, toán tử “<<” được gọi là toán tử “dịch trái” dùng trên dạng nhị phân của các con số, nếu bạn thấy x=5<<3 nghĩa là dịch các bit nhị phân của 5 sang trái 3 vị trí và gán cho x, như mô tả như sau:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhVS1N-J8enMA2Nz5dtcz3WiPVo3oSQyRnkMPvyKuex-U7_4bwU5Jt0npe5na-ZKwSk4-T14Ah8UnIjHtGehtHKpTdp6TDhkQ-7hKGWAfu2A-y9lJ-BX8pOb-VRkTUH7epOvqcYtzY6g_Th/s1600/dichtrai.jpg)

     Bạn thấy toàn bộ các bit của 5 đã dịch sang trái 3 vị trí và giá trị của số mới thu được là x=40, chú ý 40=5x8=5x2^3 . Hãy nhìn câu lệnh MCUCR|=(1<<ISC11)|(1<<ISC01), giờ thì bạn đã hiểu (1<<ISC11) nghĩa là dịch số 1 sang trái ISC11 vị trí, và (1<<ISC01) là dịch số 1 sang trái ISC01 vị trí, nhưng ISC11 và ISC01 ở đâu ra và giá trị của chúng là bao nhiêu? Bạn chú ý, khi bạn include file “io.h” thì file “iom8.h” được chèn vào, và trong file này chứa khai báo địa chỉ các thanh ghi của chip atmega8, các tên bit cũng được khai báo sẵn trong file này, nếu bạn mở file iom8.h (thường nằm trong thư mục ~\WinAVR\avr\include\avr) bằng 1 chương trình text editor như notepad, dùng chức năng find bạn sẽ thấy các dòng định nghĩa như sau:

/\* MCUCR \*/  
#define SE         7  
#define SM2      6  
#define SM1      5  
#define SM0      4  
#define ISC11   3  
#define ISC10   2  
#define ISC01   1  
#define ISC00   0

        Đây là định nghĩa vị trí các bit trong thanh ghi MCUCR, vậy là đã rõ, ISC11=3, ISC01=1, do đó: (1<<ISC11) tương đương (1<<3) = 00001000 (Binary) và (1<<ISC01) = 00000010,  bạn hãy tưởng tượng rằng bạn đã mang số 1 đến các vị trí của ISC11 và ISC01 trong thanh ghi MCUCR. Bây giờ đến lượt toán tử OR bitwise “|”.

(1<<ISC11)                        = 00001000  
(1<<ISC01)                        = 00000010  
--------------------------------------------------  
(1<<ISC11)|(1<<ISC01)     = 00001010

       Gán giá trị này cho MCUCR, đối chiếu với bảng các giá trị của các bit ISC (bảng 9) bạn sẽ thấy chúng ta đang set cho 2 ngắt là falling edge. Điều cuối cùng của câu lệnh set MCUCR là cách rút gọn câu lệnh MCUCR|=(1<<ISC11)|(1<<ISC01) thực chất là MCUCR= MCUCR|  ((1<<ISC11)|(1<<ISC01)), đây là cách set một số bit trong một thanh ghi mà không muốn làm ảnh hưởng đến các bit khác (nhưng bạn phải thật cẩn thận với cách làm này vì có thể sẽ phản tác dụng nếu bạn không nắm rõ), bạn có thể gán trực tiếp MCUCR=(1<<ISC11)|(1<<ISC01), hay nhanh hơn MCUCR=0x0A (0x0A=00001010). Vậy lí do nào khiến tôi biến 1 câu lệnh gán đơn giản thành một “bài toán” khó hiểu, câu trả lời chính là tính tổng quát. Trong các chip AVR khác nhau, vị trí các bit trong các thanh ghi là rất khác nhau, câu lệnh MCUCR=0x0A đúng cho atmega8 nhưng không áp dụng được cho các chip khác trong khi câu lệnh MCUCR=(1<<ISC11)|(1<<ISC01) thì hoạt động tốt, một lí do khác là cách viết gián tiếp này giúp người khác (hay chính bạn sau này) khi đọc code có thể dễ dàng hiểu được ý đồ người viết…

# TIMER - COUNTER

### Giới thiệu.

Trong bài 3 tôi đã giới thiệu khái quát phương pháp lập trình bằng ngôn ngữ C cho AVR với WinAVR và cách sử dụng ngắt trong AVR. Bài 4 này chúng ta sẽ khảo sát các chế độ hoạt động của  phương pháp điều khiển các bộ định thời, đếm (Timer/Counter) trong AVR. Công cụ phục vụ cho bài này vẫn là bộ công cụ WinAVR và phần mềm mô phỏng Proteus. Tôi vẫn dùng chip Atmega8 để làm ví dụ. Một điều không may mắn là không phải tất cả các bộ Timer/Counter trên tất cả các dòng chip AVR là như nhau, vì thế những gì tôi trình bày trong bài này có thể sẽ không đúng với các dòng AVR khác như AT90S…Tuy nhiên tôi cũng sẽ cố gắng chỉ ra một số điểm khác biệt cơ bản để các bạn có thể tự mình điều khiển các chip khác. Nội dung bài học này bao gồm:

* Nắm bắt cơ bản các bộ Timer/Counter có trên AVR.
* Sử dụng các Timer/Counter như các bộ định thời.
* Sử dụng các Timer/Counter như các bộ đếm.
* Sử dụng các Timer/Counter như các bộ tạo xung điều rộng PWM.
* Viết một ví dụ điều khiển động cơ RC servo bằng PWM.

### Tổng quan các bộ Timer/Counter trên chip Atmega8.

Timer/Counter là các module độc lập với CPU. Chức năng chính của các bộ Timer/Counter, như tên gọi của chúng, là định thì (tạo ra một khoảng thời gian, đếm thời gian…) và đếm sự kiện.  Trên các chip AVR, các bộ Timer/Counter còn có thêm chức năng tạo ra các xung điều rộng PWM (Pulse Width Modulation), ở một số dòng AVR, một số Timer/Counter còn được dùng như các bộ canh chỉnh thời gian (calibration) trong các ứng dụng thời gian thực. Các bộ Timer/Counter được chia theo độ rộng thanh ghi chứa giá trị định thời hay giá trị đếm của chúng, cụ thể trên chip Atmega8 có 2 bộ Timer 8 bit (Timer/Counter0 và Timer/Counter2) và 1 bộ 16 bit (Timer/Counter1). Chế độ hoạt động và phương pháp điều khiển của từng Timer/Counter cũng không hoàn toàn giống nhau, ví dụ ở chip Atmega8:

**Timer/Counter0**: là một bộ định thời, đếm đơn giản với 8 bit. Gọi là đơn giản vì bộ này chỉ có 1 chế độ hoạt động (mode) so với 5 chế độ của bộ Timer/Counter1. Chế độ hoat động của Timer/Counter0 thực chất có thể coi như 2 chế độ nhỏ (và cũng là 2 chức năng cơ bản) đó là tạo ra một khoảng thời gian và đếm sự kiện. Chú ý là trên các chip AVR dòng mega sau này như Atmega16,32,64…chức năng của Timer/Counter0 được nâng lên như các bộ Timer/Counter1…

**Timer/Counter1**: là bộ định thời, đếm đa năng 16 bit. Bộ Timer/Counter này có 5 chế độ hoạt động chính. Ngoài các chức năng thông thường, Timer/Counter1 còn được dùng để tạo ra xung điều rộng PWM dùng cho các mục đích điều khiển. Có thể tạo 2 tín hiệu PWM  độc lập trên các chân OC1A (chân 15) và OC1B (chân 16) bằng Timer/Counter1. Các bộ Timer/Counter kiểu này được tích hợp thêm khá nhiều trong các chip AVR sau này, ví dụ Atmega128 có 2 bộ, Atmega2561 có 4 bộ…

**Timer/Counter2**: tuy là một module 8 bit như Timer/Counter0 nhưng Timer/Counter2 có đến 4 chế độ hoạt động như Timer/Counter1, ngoài ra nó nó còn được sử dụng như một module canh chỉnh thời gian cho các ứng dụng thời gian thực (chế độ asynchronous).

       Trong phạm vi bài 4 này, tôi chủ yếu hướng dẫn cách sử dụng 4 chế độ hoạt động của các Timer/Counter. Chế độ asynchronous của Timer/Counter2 sẽ được bỏ qua vì có thể chế độ này không được sử dụng phổ biến.

Trước khi khảo sát hoạt động của các Timer/Counter, chúng ta thống nhất cách gọi tắt tên gọi của các Timer/Counter là T/C, ví dụ T/C0 để chỉ Timer/Counter0…

**II. Sử dụng Timer/Counter.**

       Có một số định nghĩa quan trọng mà chúng ta cần nắm bắt trước khi sử dụng các T/C trong AVR:

* BOTTOM: là giá trị thấp nhất mà một T/C có thể đạt được, giá trị này luôn là 0.
* MAX: là giá trị lớn nhất mà một T/C có thể đạt được, giá trị này được quy định bởi bởi giá trị lớn nhất mà thanh ghi đếm của T/C có thể chứa được. Ví dụ với một bộ T/C 8 bit thì giá trị MAX luôn là 0xFF (tức 255 trong hệ thập phân), với bộ T/C 16 bit thì MAX bằng 0xFFFF (65535). Như thế MAX là giá trị không đổi trong mỗi T/C.
* TOP: là giá trị mà khi T/C đạt đến nó sẽ thay đổi trạng thái, giá trị này không nhất thiết là số lơn nhất 8 bit hay 16 bit như MAX, giá trị của TOP có thể thanh đổi bằng cách điều khiển các bit điều khiển tương ứng hoặc có thể nhập trừ tiếp thông qua một số thanh ghi. Chúng ta sẽ hiểu rõ về giá trị TOP trong lúc khảo sát T/C1.

**1. Timer/Counter0:**

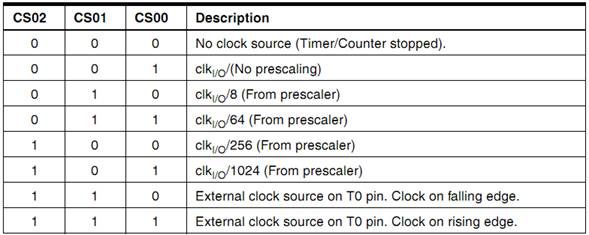
**Thanh ghi**: có 4 thanh ghi được thiết kế riêng cho hoạt động và điều khiển T/C0, đó là:

* TCNT0 (Timer/Counter Register): là 1 thanh ghi 8 bit chứa giá trị vận hành của T/C0. Thanh ghi này cho phép bạn đọc và ghi giá trị một cách trực tiếp.
* TCCR0 (Timer/Counter Control Register): là thanh ghi điều khiển hoạt động của T/C0. Tuy là thanh ghi 8 bit nhưng thực chất chỉ có 3 bit có tác dụng đó là CS00, CS01 và CS02.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhWQy7CxliGaGsZd-O_dqvhKTkgoeAIjDwFUH18Hp8kpvzG7Lh-LZL3yEzQrRRetG_doXcVEGnl3WrMYyyelOZg3EfQCZGcfEtxwb3OqkpZmm7Ro-Q8-9-eDStwJsSiQvSvXcSDCaR_la1l/s1600/image002.jpg)

       Các bit CS00, CS01 và CS02 gọi là các chip chọn nguồn xung nhịp cho T/C0 (Clock Select). Chức năng các bit này được mô tả trong bảng 1.

Bảng 1: chức năng các bit CS0X

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhL8K46u9yofHPoCQV5F6wNiiX7Nm2lcf38dQIECM0_UEaUkDc9E_L0q6A1cFHca98HOKF-SG5dSFhyCGDjIt8qIB9J4Msg08OtKwpApu7kGI-ve0ve4KOTOj1QGzm5HC0JfJ0tnD1rukCO/s1600/image004.jpg)

* TIMSK (Timer/Counter Interrupt Mask Register): là thanh ghi mặt nạ cho ngắt của tất cả các T/C trong Atmega8, trong đó chỉ có bit TOIE0 tức bit số 0 (bit đầu tiên) trong thanh ghi này là liên quan đến T/C0, bit này có tên là bit cho phép ngắt khi có tràn ở T/C0. Tràn (Overflow) là hiện tượng xảy ra khi bộ giá trị trong thanh ghi TCNT0 đã đạt đến MAX (255) và lại đếm thêm 1 lần nữa.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiKZAzn_Avcw54ZDvblWhOD1fVTW7AGaxSlVl0o_jIBLcvzFQMzw3CVwkA_55s9PvwGcuTZjCYs61jpRDrFmaasytFquX5x8iifgsjD5MtU41hJ7trFUoozCp0DiwdG2IGZMki4imSo_nax/s1600/image006.jpg)

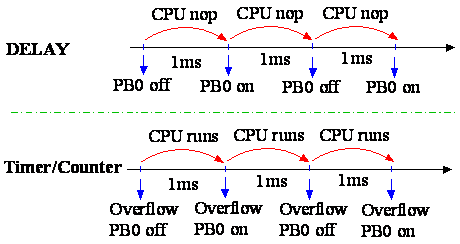
       Khi bit TOIE0=1, và bit I trong thanh ghi trạng thái được set (xem lại bài 3 về điều khiển ngắt), nếu một “tràn” xảy ra sẽ dẫn đến ngắt tràn.

* TIFR (Timer/Counter Interrupt Flag Register): là thanh ghi cờ nhớ cho tất cả các bộ T/C. Trong thanh ghi này bit số 0, TOV0 là cờ chỉ thị ngắt tràn của T/C0. Khi có ngắt tràn xảy ra, bit này tự động được set lên 1. Thông thường trong điều khiển các T/C vai trò của thanh ghi TIFR không quá quan trọng.

**Hoạt động**: T/C0 hoạt động rất đơn giản, hoạt động của T/C được “kích” bởi một tín hiệu (signal), cứ mỗi lần xuất hiện tín hiệu “kích” giá trị của thanh ghi TCNT0 lại tăng thêm 1 đơn vị, thanh ghi này tăng cho đến khi nó đạt mức MAX là 255, tín hiệu kích tiếp theo sẽ làm thanh ghi TCNT0 trở về 0 (tràn), lúc này bit cờ tràn TOV0 sẽ tự động được set bằng 1. Với cách thức hoạt động như thế có vẻ T/C0 “vô dụng” vì cứ tăng từ 0 đến 255 rồi lại quay về không và quá trình lặp lại. Tuy nhiên, yếu tố tạo sự khác biệt chính là “tính hiệu kích” và ngắt tràn, kết hợp 2 yếu tố này chúng ta có thể tạo ra 1 bộ định thời gian hoặc 1 bộ đếm sự kiện. Trước hết bạn hãy nhìn lại bảng 1 về các bit chọn xung nhịp cho T/C0. Xung nhịp cho T/C0 chính là tín hiệu kích cho T/C0, xung nhịp này có thể tạo bằng nguồn tạo dao động của chip (thạch anh, dao động nội trong chip…). Bằng cách đặt giá trị cho các bit CS00, CS01 và CS02 của thanh ghi điều khiển TCCR0 chúng ta sẽ quyết định “bao lâu” thì sẽ kích T/C0 một lần. Ví dụ mạch ứng dụng của bạn có nguồn dao động clkI/O = 1MHz tức  chu kỳ 1 nhịp là 1us ( 1 micro giây), bạn đặt TCCR0=5 (SC02=1, CS01=0, CS00=1), căn cứ theo bảng 1 tín hiệu kích cho T/C0 sẽ bằng clkI/O/1024 nghĩa là sau 1024us thì T/C0 mới được “kích” 1 lần, nói cách khác giá trị của TCNT0 tăng thêm 1 sau 1024us (chú ý là tần số được chia cho 1024 thì chu kỳ sẽ tăng 1024 lần). Quan sát 2 dòng cuối cùng trong bảng 1 bạn sẽ thấy rằng tín hiệu kích cho T/C0 có thể lấy từ bên ngoài (External clock source), đây chính là ý tưởng cho hoạt động của chức năng đếm sự kiện trên T/C0. Bằng cách thay đổi trạng thái chân T0 (chân 6 trên chip Atmega8) chúng ta sẽ làm tăng giá trị thanh ghi TCNT0 hay nói cách khác T/C0 có thể dùng để đếm sự kiện xảy ra trên chân T0. Dưới đây chúng ta sẽ xem xét cụ thể cách điều khiển T/C0 theo 1 chế độ định thời gian và đếm.

***1.1 Bộ định thời gian.***

       Chúng ta có thể tạo ra 1 bộ định thì để cài đặt một khoảng thời gian nào đó. Ví dụ bạn muốn rằng cứ sau chính xác 1ms thì chân PB0 thay đổi trạng thái 1 lần (nhấp nháy), bạn lại không muốn dùng các lệnh delay như trước nay vẫn dùng vì nhược điểm của delay là “CPU không làm gì cả” trong lúc delay, vì thế trong nhiều trường hợp các lệnh delay rất hạn chế được sử dụng. Bây giờ chúng ta dùng T/C0 để làm việc này, ý tưởng là chúng ta cho bộ đếm T/C0 hoạt động, khi nó đếm đủ 1ms thì nó sẽ tự kích hoạt ngắt tràn, trong trình phục vụ ngắt tràn chúng tat hay đổi trạng thái chân PB0. Tôi minh họa ý tưởng như trong hình 1.

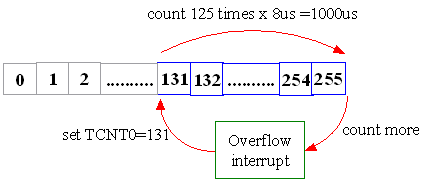
[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiWwkSpRoiQc_EsVqA7Z2qBc7lJ8rPU6TXt4wtbCiDOdXFqdGcXZaYUMQnklhudQ2N9Wqo0j9bwJAIrWe1ZZeItnhxw8-1Psgzh5gL_Adb_zscpLzdb6hBuPpVc65HVBqqvlU4bGMZOnYQt/s1600/image008.gif)

Hình 1. So sánh 2 cách làm việc.

       (CPU nop: trong khoảng thời gian này CPU không làm gì cả)

       Một vấn đề nảy sinh lúc này, như tôi trình bày trong phần trước, T/C0 chỉ đếm từ 0 đến 255  rồi lại quay về 0 (xảy ra 1 ngắt tràn), như thế dường như chúng ta không thể cài đặt giá trị mong muốn bất kỳ cho T/C0? Câu trả lời là chúng ta có thể bằng cách gán trước một giá trị cho thanh ghi TCNT0, khi ấy T/C0 sẽ đếm từ giá trị mà chúng ta gán trước và kết thúc ở 255. Tuy nhiên do khi tràn xảy ra, TCNT0 lại được tự động trả về 0, do đó việc gán giá trị khởi tạo cho TCNT0 phải được thực hiện liên tục sau mỗi lần xảy ra tràn, vị trí tốt nhất là đặt trong trình phục vụ ngắt tràn.

       Việc còn lại và cũng là việc quan trọng nhất là việc tính toán giá trị chia (prescaler) cho xung nhịp của T/C0 và việc xác định giá trị khởi đầu cần gán cho thanh ghi TCNT0 để có được 1 khoảng thời gian định thì chính xác như mong muốn. Trước hết chúng ta sẽ chọn prescaler  sao cho hợp lí nhất (chọn giá trị chia bằng cách set 3 bit CS02,CS01,CS00). Giả sử nguồn xung clock  “nuôi” chip của chúng ta là clkI/O=1MHz tức là 1 nhịp mất 1us, nếu chúng ta để prescaler=1, tức là tần số của T/C0 (tạm gọi là fT/C0) cũng bằng clkI/O=1MHz, cứ 1us T/C0 được kích và TCNT0 sẽ tăng 1 đơn vị. Khi đó giá trị lớn nhất mà T/C0 có thể đạt được là 256 x 1us=256us, giá trị này nhỏ hơn 1ms mà ta mong muốn. Nếu chọn prescaler=8 (xem bảng 1) nghĩa là cứ sau 8 nhịp (8us) thì TCNT0 mới tăng 1 đơn vị, khả năng lớn nhất mà T/C0 đếm được là 256 x 8us=2048us, lớn hơn 1ms, vậy ta hoàn toàn có thể sử dụng prescaler=8 để tạo ra một khoảng định thì 1ms. Bước tiếp theo là xác định giá trị khởi đầu của TCNT0 để T/C0 đếm đúng 1ms (1000us). Ứng với prescaler=8 chúng ta đã biết là cứ 8us thì TCNT0 tăng 1 đơn vị, dễ dàng tính được bộ đếm cần đếm 1000/8=125 lần để hết 1ms, do đó  giá trị ban đầu của TCNT0 phải là 256-125=131. Bạn có thể quan sát hình 2 để hiểu thấu đáo hơn.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEguhPTcMOEWdIP0SruT4wNCsyFY17fRwCojoomEsU89rqwSt8I3A1tnnB6wRKMp2fAWMOrpYG7vRiopA9LSa8y_bDqcq8yNUhQ0Kqv5YsGhDBuvUvCcyQ3fx0pgXWd00wG7tmYG_Vnam_ym/s1600/image010.gif)

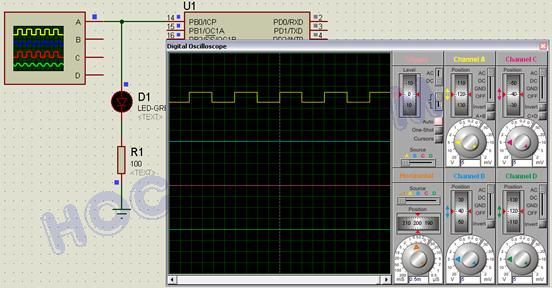
Hình 2. Quá trình thực hiện.

       Hãy tạo 1 Project bằng Programmer Notepad với tên gọi TIMER0 và viết đoạn code cho Project này như trong list 1.

List 1. Định thì 1ms với T/C0.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | #include <avr/io.h> #include <avr/interrupt.h> #include <util/delay.h>  int main(void){        DDRB=0xFF;                //PORTB la output PORT        PORTB=0x00;         TCCR0=(1<<CS01);// CS02=0, CS01=1, CS00=0: chon Prescaler = 8        TCNT0=131;              //gan gia tri khoi tao cho T/C0        TIMSK=(1<<TOIE0);//cho phep ngat khi co tran o T/C0        sei();                       //set bit I cho phep ngat toan cuc         while (1){           //vòng lặp vô tận                //do nothing        }        return 0; }  //trinh phuc vu ngat tran T/C0 ISR (TIMER0\_OVF\_vect ){         TCNT0=131; //gan gia tri khoi tao cho T/C0        PORTB^=1; //doi trang thai Bit PB0 } |

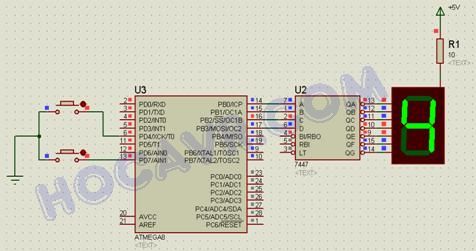
       Đoạn code rất đơn giản, bạn chỉ cần chú ý đến 3 dòng khai báo cho T/C0 (dòng  9, 10, 11). Với dòng 9: TCCR0=(1<<CS01) là 1 cách set bit CS01 trong thanh ghi điều khiển TCCR0 lên 1, 2 bit CS02 và CS00 được để giá trị 0 (bạn xem lại bài 3 về cách set các bit đặc biệt trong các thanh ghi), tóm lại dòng này tương đương TCCR0=2, giá trị Prescaler được chọn bằng 8 (tham khảo bảng 1). Dòng 10 chúng ta gán giá trị khởi tạo cho thanh ghi TCNT0. Và dòng 11 set bit TIOE0 lên 1 để cho phép ngắt xảy ra khi có tràn ở T/C0. Trong trình phục vụ ngắt tràn T/C0, chúng ta sẽ thực hiện đổi trạng thái chân PB0 bằng toán từ XOR (^), chú ý đến ý nghĩa của toán tử XOR: nếu XOR một bit với số 1 thì bit này sẽ chuyển trạng thái (từ 0 sang 1 và ngược lại). Cuối cùng và quan trọng là chúng ta cần gán lại giá trị khởi tạo cho T/C0.  
     Bạn có thể vẽ môt mạch điện mô phỏng đơn giản dùng 1 Oscilloscope như trong hình 3 để kiểm tra hoạt động của  đoạn code.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjAA5sB52Wty_uV6pstwAOH_9SfMVmNKlqI8a8Z2KtAuiUXMN9ByVN4mV_jHn2MFq3-vS9WDJw4gVKgaSJMFgx_eEX9k_VN67Wpt7ZKW7TqYhdwSTMO9jlcn3X8nh-bxSCnvIsdtVzmyJ0E/s1600/image012.jpg)

Hình 3. Mô phỏng định thì của T/C0.

***1.2 Bộ đếm sự kiện.***

       Như tôi trình bày trong phần hoạt động của T/C0, chúng ta có thể dùng T/C0 như một bộ đếm (counter) để đếm các sự kiện (sự thay đổi trạng thái) xảy ra trên chân T0. Bằng cách đặt giá trị cho thanh ghi TCCR0 = 6 (CS02=1, CS01=1, CS00=0) cho phép đếm “cạnh xuống” trên chân T0, nếu TCCR0 = 7 (CS02=1, CS01=1, CS00=1) thì “cạnh lên” trên chân T0 sẽ được đếm. Có sử dụng ngắt hay không phụ thuộc vào mục đích sử dụng. Khảo sát 1 ví dụ đơn giản gần giống với ví dụ đếm trong bài AVR2 nhưng sử dụng T/C0 và chỉ đếm 1 chiều tăng. Kết nối mạch điện như trong hình 4, mỗi lần Button 1 được nhấn, giá trị đếm tăng thêm 1. Button 2 dùng reset giá trị đếm về 0. Đoạn code cho ví dụ thứ 2 này được trình bày trong List 2.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjzFZaXa6srsIOqrGDa9ZkvlnVN49KEaUMGSaT100OkScU6rrVC7eEg2hxMmbLzdKlWuXd-5eIOUxQ1ANAKHBOwaGS-ct6D_d_TTHCSMUUY_7G_Dal8_ilR9UiUlB4wdUp37YbVz98MsEmk/s1600/image014.jpg)

Hình 4. Đếm 1 chiều bằng T/C0.

List 2. Đếm sự kiện với T/C0

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | #include <avr/io.h> #include <avr/interrupt.h>  int main(void){        DDRB=0xFF;                //PORTB la output PORT        PORTB=0x00;        DDRD=0x00; //khai bao PORTD la input de ket noi Button kich vao chan T0        PORTD=0xFF; //su dung dien tro keo len cho PORTD         TCCR0=(1<<CS02)|(1<<CS01);// CS02=1,CS01=1, CS00=0: xung nhip tu  chan T0, down        TCNT0=0;         while (1){           //vòng lặp vô tận               if (TCNT0==10) TCNT0=0;              PORTB=TCNT0;   //xuat gia tri dem ra led 7 doan              if (bit\_is\_clear(PIND,7)) TCNT0=0;  //Reset bo dem neu chan PD7=0         }        return 0; } |

       Nội dung trong chương trình chính là khai báo các hướng giao tiếp cho các PORT, PORTB là ouput để xuất kết quả đếm ra led 7 đoạn, PORTD được khái báo input vì các button được nối với PORT này. T/C0 được khai báo sử dụng nguồn kích ngoài từ T0, dạng cạnh xuống thông qua dòng TCCR0=(1<<CS02)|(1<<CS01), bạn cũng có thể khai báo tương đương là TCCR0=6 (tham khảo bảng 1). Giá trị của bộ đếm sẽ được xuất ra PORTB để kiểm tra. Điểm chú ý trong đoạn chương trình này là macro “bit\_is\_clear”, đây là một macro được định nghĩa trong file “sfr\_defs.h” dùng để kiểm tra 1 bit trong một thanh ghi đặc biệt có được xóa (bằng 0) hay không, trong trường hợp của đoạn code trên: “if(bit\_is\_clear(PIND,7)) TCNT0=0;” nghĩa là kiểm tra xem nếu chân PD7 được kéo xuống 0 (button 2 được nhấn) thì sẽ reset bộ đếm về 0.

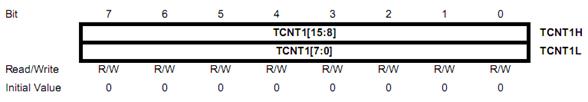
       Như vậy việc sử dụng T/C0 là tương đối đơn giản, bạn chỉ cần khai báo các giá trị thích hợp cho thanh ghi điều khiển TCCR0 bằng cách tham khảo bảng 1, sau đó khởi tạo giá trị cho TCNT0 (nếu cần thiết), khai báo có sử dụng ngắt hay không bằng cách set hay không set bit TOIE0 trong thanh ghi TIMSK là hoàn tất.

**2. Timer/Counter1:**

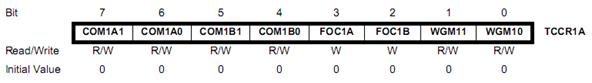
       Timer/Counter1 là bộ T/C 16 bits, đa chức năng. Đây là bộ T/C rất lý tưởng cho lập trình đo lường và điều khiển vì có độ phân giải cao (16 bits) và có khả năng tạo xung điều rộng PWM (Pulse Width Modulation – thường dùng để điều khiển động cơ).

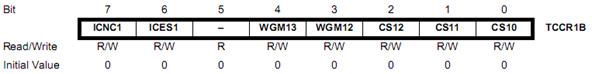
**Thanh ghi**: có khá nhiều thanh ghi liên quan đến T/C1. Vì là T/C 16 bits trong khi độ rộng bộ nhớ dữ liệu của AVR là 8 bit (xem lại bài 2) nên đôi khi cần dùng những cặp thanh ghi 8 bits tạo thành 1 thanh ghi 16 bit, 2 thanh ghi 8 bits sẽ có tên kết thúc bằng các ký tự L và H trong đó L là thanh ghi chứa 8 bits thấp (LOW) và H là thanh ghi chứa 8 bits cao (High) của giá trị 16 bits mà chúng tạo thành.

* TCNT1H và TCNT1L (Timer/Counter Register): là 2 thanh ghi 8 bit tạo thành thanh ghi 16 bits (TCNT1) chứa giá trị vận hành của T/C1. Cả 2 thanh ghi này cho phép bạn đọc và ghi giá trị một cách trực tiếp. 2 thanh ghi được kết hợp như sau:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjtW_-J4EY3d0YtbT6JZ8DFyveSuWbRkfA9gC4XCJdfnYWbHFg4HwtP-7xZp2D7-X1Op7cSalzqTt0iHeeKggVk-OqVu2BDSFaBuXXzTBLWagy6awEHG8hGwwFgbwPValWZO2E6M2BpOx_w/s1600/image016.jpg)

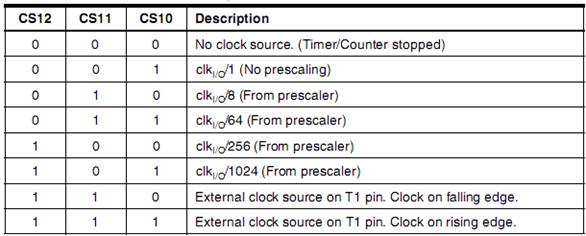
* TCCR1A và TCCR1B (Timer/Counter Control Register): là 2 thanh ghi điều khiển hoạt động của T/C1. Tất cả các mode hoạt động của T/C1 đều được xác định thông qua các bit trong 2 thanh ghi này. Tuy nhiên, đây không phải là 2 byte cao và thấp của một thanh ghi mà là 2 thanh ghi hoàn toàn độc lập. Các bit trong 2 thanh ghi này bao gồm các bit chọn mode hay chọn dạng sóng (Waveform Generating Mode – WGM), các bit quy định dạng ngõ ra (Compare Output Match – COM), các bit chọn giá trị chia prescaler cho xung nhịp (Clock Select – CS)…Cấu trúc của 2 thanh ghi được trình bày như bên dưới.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhiXCJHDnJa_04rUddze2b_1cj_fepIYJYvfYOA0tI_j3uu7EbgGhVgMpXD5DpZEfKH9ypmw3o88UmAIeGNbTiY8DRqGajhnVXjq6W7YwRvTXGM73Vz6-2S4IE24dgneeKuki5GQT8-kt04/s1600/image018.jpg)

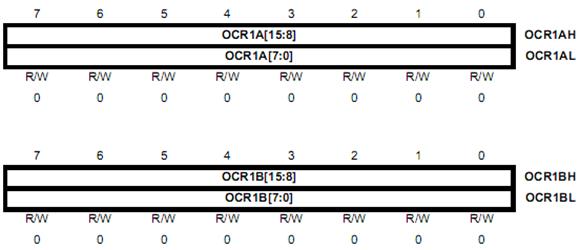
[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgHgZUrScFKye-QHsDGamzyxzTHKTrfV4NMHLfN7h-Mn5hZi33rOXFph7Jcnu-DFQFxqFwjEebhOgdlR7UuTyjwVjunkgXS_nFepXsNofm5X7IPO6fuHU1XPaKg8AoE4OQ3eOPgNfSyflp8/s1600/image020.jpg)

       Nhìn chung để “thuộc” hết cách phối hợp các bit trong 2 thanh ghi TCCR1A và TCCR1B là tương đối phức tạp vì T/C1 có rất nhiều mode hoạt động, chúng ta sẽ khảo sát chúng trong phần các chế độ hoạt động của T/C1 bên dưới. Ở đây, trong thanh ghi TCCR1B có 3 bit khá quen thuộc là CS10, CS11 và CS12. Đây là các bit chọn xung nhịp cho  T/C1 như truong T/C0. Bảng 2 sẽ tóm tắt các chế độ xung nhịp trong T/C1.

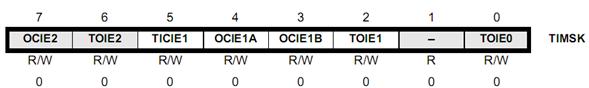
Bảng 2: chức năng các bit CS12, CS11 và CS10.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEga_ShaIsQsZxn_9rAG1uv0wYbLuy26uG5xIZtUeiRqLaIuScy6iiiuJSkq9w1d785oU4Xv29waRokL4psPKKUxFcdfCBa0x0Ceb7e_BWIPne5CPbHfq5psT07JFYqJgvb8MqTQE1Z8i9si/s1600/image022.jpg)

* OCR1A và OCR1B (Ouput Compare Register A và B): có một số khái niệm mới mà chúng ta cần biết khi làm việc với T/C1, một trong số đó là Ouput Compare (sorry, I don’t wanna translate it to Vietnamese). Trong lúc T/C hoạt động, giá trị thanh ghi TCNT1 tăng, giá trị này được liên tục so sánh với các thanh ghi OCR1A và OCR1B (so sánh độc lập với từng thanh ghi), việc so sánh này trên AVR gọi là gọi là Ouput Compare. Khi giá trị so sánh bằng nhau thì 1 “Match” xảy ra, khi đó một ngắt hoặc 1 sự thay đổi trên chân OC1A (hoặc/và chân OC1B) xảy ra (đây là cách tạo PWM bởi T/C1). Tại sao lại có A và B? Đó là vì người thiết kế AVR muốn mở rộng khả năng ứng dụng T/C1 cho bạn. A và B đại diện cho 2 kênh (channel)  và B. Cũng vì điều này mà chúng ta có thể tạo 2 kênh PWM bằng T/C1. Tóm  lại, cơ bản 2 thanh ghi này chứa các giá trị để so sánh, chức năng và các chế độ hoạt động cụ thể của chúng sẽ được khảo sát trong các phần sau.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjkROe3U-uk-JCtP8shRQs4efx_Tr7iLACoAdS83mNEgt6HXJqbDhYzgTL4fuKx3tHAtQDgv_NBHOyDHtR3U-CNZa_YGsP_BiMFm_Llu7gmVxodX3PSCdrkv4-nCmV73kriGBJK1377MLJr/s1600/image024.jpg)

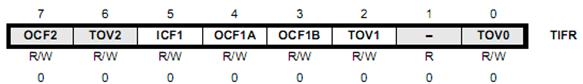
* ICR1 (InputCapture Register 1): khái niệm mới thứ 2 của T/C1 là Input Capture. Khi có 1 sự kiện trên chân ICP1 (chân 14 trên Atmega8), thanh ghi ICR1sẽ “capture” giá trị của thanh ghi đếm TCNT1. Một ngắt có thể xảy ra trong trường hợp này, vì thế Input Capture có thể được dùng để cập nhật giá trị “TOP” của T/C1.
* TIMSK (Timer/Counter Interrupt Mask Register): các bộ T/C trên AVR dùng chung thanh ghi mặt nạ ngắt, vì thế TIMSK cũng được dùng để quy định ngắt cho T/C1. Có điều lúc này chúng ta chỉ quan tâm đến các bit từ 2 đến 5 của TIMSK. Có tất cả 4 loại ngắt trên T/C1 (nhớ lại T/C0 chỉ có 1 loại ngắt tràn)

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj25LHo1yP7PexlIQ1UqRujpTQ7zFVPQBp5Nwi3U1Vbg39zyJ_S907U3ZS0aIGA4cFaEj1AUsbgsc5rDWqKQAxAaZmIrthjgL_JFx4b1o9jrA1UUi-1WW6omywTvLLsGBNA8qrE6HVBP0Ab/s1600/image026.jpg)

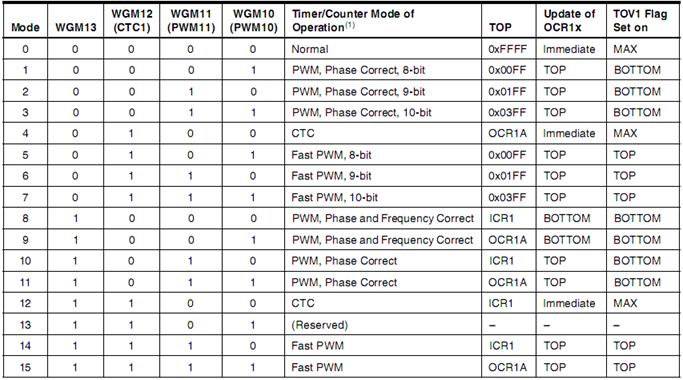
Bit 2 trong TIMSK là TOIE1, bit quy định ngắt tràn cho thanh T/C1 (tương tự trường hợp của T/C0).  
Bit 3, OCIE1B là bit cho phép ngắt khi có 1 “Match” xảy ra trong việc so sánh TCNT1 với OCR1B.  
Bit 4, OCIE1A là bit cho phép ngắt khi có 1 “Match” xảy ra trong việc so sánh TCNT1 với OCR1A.  
Bit 5, TICIE1 là bit cho phép ngắt trong trường hợp Input Capture được dùng.

       Cùng với việc set các bit trên, bit I trong thanh ghi trạng thái phải được set nếu muốn sử dụng ngắt (xem lại [bài 3](http://www.hocavr.com/index.php/lectures/ngatngoai" \t "_blank) về điều khiển ngắt).

* TIFR (Timer/Counter Interrupt Flag Register): là thanh ghi cờ nhớ cho tất cả các bộ T/C. Các bit từ 2 đến 5 trong thanh ghi này là các cờ trạng thái của T/C1.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgFX9wwdqy0ggYUXdbMVOQZ5PlS9zN2W3RTA1xzJWZ3UEgaSJEeBsHz3aSwJOXDDVZ2t8yab2pNLCDUO0I_FdLn9ltLEAFc6bfeSi2zeFef0ja0ioiCCZACguZLgsxjUwaulCvX9LjiHUVh/s1600/image028.jpg)

       Các mode hoạt động: có tất cả 5 chế độ hoạt động chính trên T/C1. Các chế độ hoạt động cơ bản được quy định bởi 4 bit Waveform Generation Mode (WGM13, WGM12, WGM11 WGM10) và một số bit phụ khác. 4 bit Waveform Generation Mode lại được bố trí nằm trong 2 thanh ghi TCCR1A và TCCR1B (WGM13 là bit 4, WGM12 là bit 3 trong TCCR1B trong khi WGM11 là bit 1 và WGM10 là bit 0 trong thanh ghi TCCR1A) vì thế cần phối hợp 2 thanh ghi TCCR1 trong lúc điều khiển T/C1. Các chế độ hoạt động của T/C1 được tóm tắt trong bảng sau 3:  
Bảng 3: các bit WGM và các chế độ hoạt động của T/C1.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjMk6igRUpF0_RpIx2mGVrDeu_rkK2MBd7rPP5cdPMjS9RNLqc0M8soCjFyx-SI_itq-paUEzR8rRt7KODZScGI9HoX6rAeUYYd9u0kptU_4b6d6O30u6vOCmPBEM5xk5vjt2WDi7Z6ssVD/s1600/image030.jpg)

***2.1 Normal mode (Chế độ thường).***        Đây là chế độ hoạt động đơn giản nhất của T/C1. Trong chế độ này, thanh ghi đếm TCNT1 được tăng giá trị từ 0 (BOTTOM) đến 65535 hay 0xFFFF (TOP) và quay về 0. Chế độ này hoàn toàn giống cách mà Timer0 hoạt động chỉ có khác là giá trị đếm cao nhất là 65535 thay vì 255 như trong timer0. Nhìn vào bảng 3, để set T/C1 ở Normal mode chúng ta cần set 4 bit WGM về 0, vì 0 là giá trị mặc định của các thanh ghi nên thực tế chúng ta không cần tác động đến các bit WGM. Duy nhất một việc quan trọng cần làm là set các bit Clock Select (CS12, SC11, CS10) trong thanh ghi TCCR1B (xem thêm bảng 2). Bạn có thể tham khảo ví dụ của Timer0. Đoạn code trong  list 3 là 1 ví dụ tạo 1 khoảng thời gian 10ms bằng T/C1, normal mode:

List 3. Định thì 10ms với T/C1.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | #include <avr/io.h> #include <avr/interrupt.h> #include <util/delay.h>  int main(void){        DDRB=0xFF;                //PORTB la output PORT        PORTB=0x00;         TCCR1B=(1<CS10);// CS12=0, CS11=0, CS10=1: chon Prescaler =1        // thanh ghi TCCR1B duoc dung thay vi TCCR0 cua Timer0        TCNT1=55535;              //gan gia tri khoi tao cho T/C1        TIMSK=(1<<TOIE1);//cho phep ngat khi co tran o T/C1         sei();                       //set bit I cho phep ngat toan cuc         while (1){           //vòng lặp vô tận               //do nothing        }        return 0; } //trinh phuc vu ngat tran T/C1 ISR (TIMER1\_OVF\_vect ){         TCNT1=55535; //gan gia tri khoi tao cho T/C1        PORTB ^=1;  //doi trang thai Bit PB0 } |

***2.2 Clear Timer on Compare Match (xóa timer nếu xảy ra bằng trong so sánh)-CTC.***        Một cách gọi tắt của chế độ hoạt động này là CTC, một chế độ hoạt động mới trên T/C1. Nhìn vào bảng 3 bạn sẽ thấy có 2 mode CTC (mode 4 và mode 12). Tôi lấy ví dụ mode 4 để giải thích hoạt động của CTC. Khi bạn set các bit Waveform Generation Mode tuong ứng: WGM13=0, WGM12=1, WGM11=0, WGM10=0 thì mode 4 được chọn. Trong mode này, thanh ghi OCR1A chứa giá trị TOP (giá trị so sánh do người dùng đặt), thanh ghi đếm TCNT1 tăng từ 0, khi TCNT1 bằng giá trị chứa trong OCR1A thì một “Compare Match” xảy ra. Khi đó, một ngắt có thể xảy ra nếu chúng ta cho phép ngắt Compare Match (set bit OCF1A trong thanh ghi TIMSK lên 1). Mode này cũng tương đối đơn giản, một ứng dụng cơ bản của mode này là đơn giản hóa việc đếm các sự kiện bên ngoài. Ví dụ bạn kết nối 1 sensor  đếm số người đi vào 1 căn phòng với chấn T1 (chân counter source  của T/C1), bạn muốn rằng cứ sau khi đếm 5 người thì sẽ thông báo 1 lần. List 4 là đoạn code mô tả ví dụ này:

List 4. Phối hợp CTC với đếm sự kiện.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | #include <avr/io.h> #include <avr/interrupt.h> #include <util/delay.h> volatile  usigned char val=0;  //khai bao 1 bien tam val va khoi tao =0 int main(void){        DDRB=0xFF;                //PORTB la output PORT        PORTB=0x00;        TCCR1B=(1<<WGM12)|(1<<CS12)|(1<<CS11); //xung nhip tu chan T1, canh xuong        OCR1A=4;             //gan gia tri can so sanh        TIMSK=(1<OCIE1A);//cho phep ngat khi gia tri dem bang 4         sei();                       //set bit I cho phep ngat toan cuc         while (1){           //vòng lặp vô tận                //do nothing        }        return 0; } //trinh phuc vu ngat compare match ISR (TIMER1\_COMPA\_vect){         val++;        if (val==10) val=0;  //gioi han bien val tu 0 den 9        PORTB =val;        //xuat gia tri ra PORTB } |

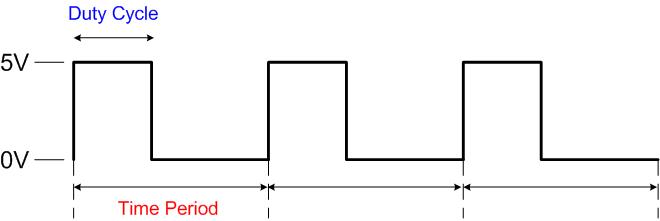
    Tôi chỉ giải thích những điểm mới trong List 4. Thứ nhất là “attribute” volatile dùng trước khai báo biến val, biến val được khai báo là unsigned char (8 bit, không dấu) dùng chứa giá trị tạm thời để xuất ra PORTB khi có ngắt xảy ra. Điều đặc biệt là từ khóa volatile đặt trước nó,  volatile  là một thuộc tính (attribute) của bộ biên dịch gcc-avr, nó nói với trình dịch rằng biến val sẽ được dùng trong chương trình chính và cả trong các trình phục vụ ngắt. Nếu bạn muốn cập nhập giá trị 1 biến toàn cục trong các trình phục vụ ngắt mà biến đó không được chỉ định thuộc tính volatile trước thì quá trình cập nhật thất bại. Một cách dễ hiểu hơn, bạn xem trình ISR trong ví dụ trên, cứ mỗi lần có ngắt Compare Match xảy ra, biến val được tăng thêm 1 (dòng 21) sau đó kiểm tra điều kiện bằng 10 hay không và cuối cùng là gán cho PORTB. Nếu trong khai báo của val (dòng 4) chúng ta không chỉ định volatile thì giá trị xuất ra PORTB sẽ luôn là 1 khi có ngắt. Chú ý là điều này chỉ đúng it nhất là với phiên bản WinAVR tháng 12 năm 2007, các phiên bản sau có thể không cần dùng volatile (tôi sẽ cập nhật sau).

       Dòng 8 set các bit điều khiển: TCCR1B=(1<<WGM12)|(1<<CS12)|(1<<CS11); bạn thấy tôi chỉ set bit WGM12 trong 4 bit WGM vì tôi muốn chọn mode CTC 4 (xem bảng 3). Hai bit CS12 và CS11 được set bằng 1 trong khi CS10 được giữ ở 0 để chọn  xung clock là từ bên ngoài, chân T1 (xem bảng 2). Trong dòng 10, OCR1A=4; là giá trị cần so sánh, chúng ta biết rằng TCNT1 tăng lên từ 0, vì thế để đếm 5 sự kiện thì cần đặt giá trị so sánh là 4 (0, 1, 2, 3, 4). Dòng 11 set bit cho phép ngắt khi có Compare match xảy ra (dùng cho channel A).

       Mode 12 của CTC (WGM13=1, WGM12=1, WGM11=0, WGM10=0) cũng tương tự mode 4 nhưng cái khác là giá trị cần so sánh được chứa trong thanh ghi ICR1 (không phải OCR1A hay OCR1B). Khi đó nếu muốn dùng ngắt thì bạn phải dùng ngắt Input capture. Cụ thể dòng 8 trong list 4 đổi thành:  TCCR1B=(1<<WGM13)|( (1<<WGM12)|(1<<CS12)|(1<<CS11);  dòng 10: ICR1=4 và dòng  20: ISR (TIMER1\_CAPT\_vect ){

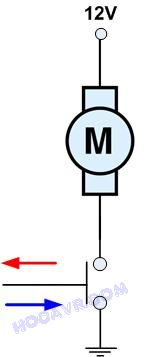
       Một khả năng khác của CTC là xuất tín hiệu xung vuông trên chân OC1A (chân 15 trên Atmega8) bằng cách set các bit Compare Output Mode trong thanh ghi TCCR1A. Tuy nhiên việc tạo các tín hiệu output trong mode CTC không thật sự thú vị. Vì vậy chúng ta sẽ khảo sát cách tạo tín hiệu output trong 1 chế độ chuyên nghiệp và thú vị hơn, chế độ PWM.

       Trước khi bắt đầu làm việc với các chế độ PWM tôi nghĩ cần thiết giới thiệu thế nào là PWM và nhắc lại các khái niệm giá trị đếm của Timer1 (hay bất kỳ timer nào khác) trên AVR. Trước hết, PWM hay Pulse Width Modulation được hiểu theo nghĩa tiếng Việt là “xung điều rộng” là khái niệm chỉ tín hiệu xung mà thường thì chu kỳ (Time period) của nó được cố định, duty cycle (thời thời gian tín hiệu ở mức HIGH) của nó có thể được thay đổi. Bạn xem 1 ví dụ về PWM  trong hình  5.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgc-QOA7XcmpnDjm_AbKG-HMgklRF5azzye7Lh2Kn4AbyH-WRJioKbd5YwjjU2xDTmp_DzApyedrHtOnyp0FVEzEpvbi6dxeqC20Ka4Ha4uRTCQxMm6juxaYWBOR_-BgewvrU_2LPCsBDIP/s1600/pwm.jpg)

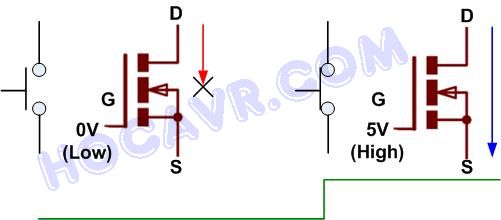
Hình 5. Ví dụ về tín hiệu PWM.

       Tạo ra PWM tức là tạo ra những tín hiệu xung mà ta có thể điều khiển duty cycle (và cả tần số ~ Time period nếu cần thiết). Timer 1 trêsn Atmega8 là 1 module lý tưởng để tạo ra các tín hiệu dạng này. Nhưng PWM dùng để làm gì và cách mà nó được sử dụng như thế nào? Tôi lấy một ví dụ như trong hình 6: một động cơ DC và một switch button.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjlbIVaDabDJrmVgTHBd1gs6wmXpo7nuTE4v_BhULY_V6enN-RGIwc9obSWw_4c7krATTV0bYeSEcloXyiIrcARPnC0ZWpVv0qUQG51pKouzeiaGi1CYP_TyXSDyXt_n2LRzo7vgZXQxk5F/s1600/dcm.jpg)

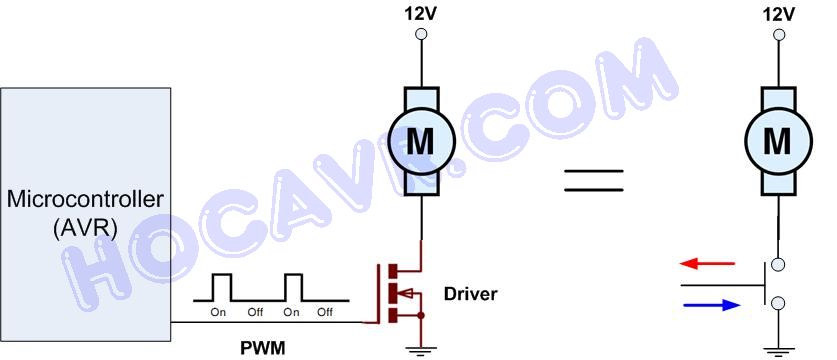
Hình 6. Motor và switch.

       Nếu nhấn button thì động cơ hoạt động, thả button thì động cơ dừng. Tuy nhiên  do tốc độ nhấn và thả của con người có hạn, bạn sẽ thấy động cơ hoạt động hơi “sượng” (ripple). Điều gì xảy ra nếu bạn nhấn và thả button với vận tốc 5000 lần/giây. Câu trả lời là tay bạn sẽ bị gãy và button sẽ bị hỏng (^^). 5000 lần/s là điều không tưởng, tuy nhiên nếu bạn làm được như thế thì tổng thời gian cho 1 lần nhấn+thả là 1:5000=0.0002s = 200us. Có sự khác biệt nào không giữa trường hợp thời gian nhấn = 150us, thời gian thả 50us và trường hợp thời gian nhấn là 50us còn thời gian thả là 150us. Bạn sẽ dễ dàng tìm câu trả lời, trong trường hợp 1 động cơ sẽ quay với vận tốc nhanh hơn trường hợp 2. Đó là ý tưởng cơ bản để sử dụng PWM điều khiển vận tốc động cơ (và điều khiển nhiều thứ khác nữa). để biến cái không tưởng trên (5000 lần/s) thành hiện thực, chúng ta sẽ thay thế cái button cơ khí kia bằng 1 công tắc điện tử (electronics switch). Thường thì các chip MOSFET được dùng làm các khóa điện tử. MOSFET thường có 3 chân G (gate), D (drain) và S (source). Ví dụ 1 MOSFET kênh N ở trạng thái thông thường 2 chân D và S ko có dòng điện chạy qua, nếu điện áp chân G lớn hơn chân S khoảng 3V trở lên thì dòng điện có thể chạy từ D sang S. hãy xem cách mô tả tương đương 1 MOSFET với 1 button trong hình 7.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhGkRjMt61JFKX75pjuuyzuuGxlXmf22TemNqv9gAvvAT-j7Pyu4WvG-OziHLAUnqyOzD7S7cwoBWY2YLX0zg5eKhUTnjvH3anfsFZcR4puLnIJDjJvB67XRWJqV8zM1b1Q9Sdz7AMS69JL/s1600/fet.jpg)

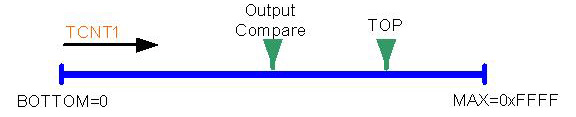
Hình 7. MOSFET và button.

       Việc “kích” các MOSFET có thể thực hiện bằng các tín hiệu PWM. Vì thế ý tưởng điều khiền động cơ trong hình 6 có thể được thực hiện lại thông qua PWM như trong hình 8.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgxxrW0Zp59n5jED8KSrTpgGr-BuqJXGhpY9izNtp4nt8s6cYPfJ17-CLGwaQLHxh8oeHJR1VpOP1g1clJ1aVfxZ3uVtqSvB-CImCX1Yf1c7hSUi3cpQL2vUSD6gp1ousw3Z2BxpWoJxCaz/s1600/pwmavr.jpg)

Hình 8. Mô hình điều khiển tốc độ động  cơ bằng PWM đơn giản.

       Như vậy là xong phần giới thiệu về PWM, bây giờ chúng ta sang các khái niệm số đếm trong Timer. Hình 9 minh họa cách bố trí các số đếm trong Timer1 trên hệ trục đếm.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEj44o5q6Y5J0tdQSjvc-BoQb2Suys7jNd3Y-5J0uBg3wGhmKFmefpEL6Ys2SIHSZtaIK5ZkP-Y7bs7fxqeTVk7MtD4uYgb1ZcdZzfxA9z81HjmXYToSfnAmRG7s0ftoZOAZjooYyFZ6Hm-x/s1600/moc.jpg)

Hình 9: các mốc giá trị của T/C1.

       BOTTOM luôn được cố định là 0 (giá trị nhỏ nhất), MAX luôn là 0xFFFF (65535). TOP là giá trị đỉnh do người dùng định nghĩa, giá trị của TOP có thể được cố định là 0xFF (255), 0x1FF (511), 0x3FF 91023) hoặc định nghĩa bởi các thanh ghi ICR1 hoặc OCR1A. thực chất đối với ứng dụng PWM thì TOP chính là Time period của PWM. Do mục đích sử dụng mà có thể chọn TOP là các giá trị cố định hay các thanh ghi, riêng với tôi, cho mục đích tạo tín hiệu PWM tôi chọn TOP định nghĩa bởi thanh ghi ICR1. Ouput Compare là giá trị so sánh của bộ Timer. Trong chế độ PWM thì Output Compare quy định Duty cycle. Với T/C1, Output Comapre là giá trị trong các thanh ghi OCR1A và OCR1B. Do có 2 thanh ghi độc lập A và B, tương ứng chúng ta có thể tạo ra 2 tín hiệu PWM trên 2 chân OC1A và OC1B bằng T/C1. Đã đến lúc chúng ta tìm hiểu cách tạo PWM trên AVR.

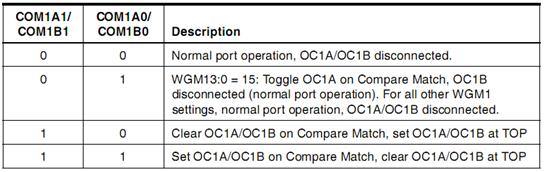
***2.3 Fast PWM (PWM tần số cao).***        Trong chế độ Fast PWM, 1 chu kỳ được tính trong 1 lần đếm từ BOTTOM lên TOP (single-slope), vì thế mà chế độ này gọi là Fast PWM (PWM nhanh). Có tất cả 5 mode trong Fast PWM tương ứng với 5 cách chọn giá trị TOP khác nhau (tham khảo bảng 3).  Việc xác lập chế độ hoạt động cho Fast PWM thực hiện thông qua 4 bit WGM và các bit chọn dạng xung ngõ ra, Compare Output Mode trong thanh ghi TCCR1A, nhìn lại 2 thanh ghi TCCR1A và TCCR1B.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEg40ejVIR6MemyuTFan9thONAe1hkobwdf_AsA2W7KW55MevWdTwtvSpxsrzV2MfVgB8jF2g8sNbxVR6NmZgn4wcAtKzgHp-nfDdylUrckFvayGiWQSMEyE-ePudNQUygBFJtLtguDQoifL/s1600/image042.jpg)

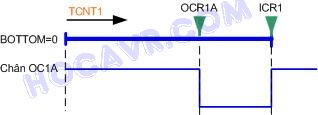
[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgu4MhEL7vNqPjAicn_h7hYzHbjS6h4bBzF1nbc-EsS9Tc9Ex8Avt17BMBr1C-RpZMBmXsBy7d9QtS0SxenRg-1uaNca2m53P5WRJ8d5_bVZCOIiG_25_SAXmMx8AVQrgdtzvJMPdxYA5Nd/s1600/image044.jpg)

       Chú ý các bit COM1A1, COM1A0 và COM1B1, COM1B0 là các bit chọn dạng tín hiệu ra của PWM (Compare Output Mode bits). COM1A1, COM1A0 dùng cho kênh A và  COM1B1, COM1B0 dùng cho kênh B. Hãy đối chiếu bảng 4.

Bảng 4: mô tả các bit COM trong chế độ fast PWM.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhBDkBG57LRKru7KvgQt0ranmF3XnNJwERY_XDPQqtgxOeWHnD8iD0bBPx-JWCKJ5veUdxWxsqVhzKs0OZfKpeMsn1jYsc8DSVNEbvvNXkMJXpGi76wuDbvh-h4cNSYg7FOBhs7d3Yi3YyX/s1600/image046.jpg)

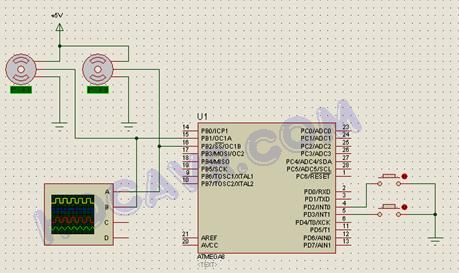
       Tôi sẽ giải thích hoạt động của Fast PWM kênh A thông qua 1 trường hợp cụ thể, mode 14 (WGM13=1, WGM12=1, WGM11=1, WGM10=0). Trong mode 14, giá trị TOP (cũng là chu kỳ của PWM) được chứa trong thanh ghi ICR1, khi hoạt động thanh ghi TCNT1 tăng giá trị từ 0, giả sử các bit phụ  COM1A=1, COM1A0=0, lúc này trạng thái của chân OC1A (chân 15) là HIGH (5V), khi TCNT1 tăng đến bằng giá trị của thanh ghi OCR1A thì chân OC1A được xóa về mức LOW (0V), thanh ghi đếm TCNT1 vẫn tiếp tục tăng đến khi nào nó bằng giá trị TOP chứa trong thanh ghi ICR1 thì TCNT1 tự động  reset về 0 và chân OC1A trở về trạng thái HIGH, cái này gọi là “Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at TOP” mà bạn thấy trong hàng 4 bảng 4. Hình 10 mô tả cách tạo xung PWM trên chân OC1A ở mode 14.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhseUxQAaYOcThCpDpHuekkCdhIG_tklkbvx0e-eL8XC8pXqc0xLZg3OJXsnl0PplCbdndnUYSg-_p2DyLAa5bxy6bAPT3F0xTkgWlfxsR5PQq44k36lgKpnvFM4tBbFBTTdODnrn34RPtZ/s1600/fastpwm.jpg)

Hình 10: Fast PMW mode 14.

       Rõ ràng chúng ta có thể điều khiển cả time period và duty cycle của PWM bằng 2 thanh ghi ICR1 và OCR1A. Thông thường giá trị của ICR1 được tính toán và gán cố định, giá trị của OCR1A được thay đổi để thực hiện mục đích điều khiển (như thay đổi vận tốc động cơ). Chú ý là nếu chúng ta set các bit phụ ngược lại: COM1A=0, COM1A0=1, thì tín hiệu PWM trên chân OC1A sẽ có phần “LOW” từ 0 đến OCR1A và “HIGH” từ OCR1A đến ICR1, đây gọi là “set OC1A/OC1B on Compare Match, clear OC1A/OC1B at TOP” (ngược với tín hiệu trên hình 10). Hoạt động của fast PWM kênh B hoàn toàn tương tự, trong đó thanh ghi ICR1 cũng chứa TOP của PWM kênh B và thanh ghi ICR1B chứa duty cycle. Như vậy 2 kênh A và B có cùng tần số hay Time period và duty cycle được điều khiển độc lập. Chân xuất tín hiệu PWM của kênh B là chân OC1B (chân 16 trên Atmega8).

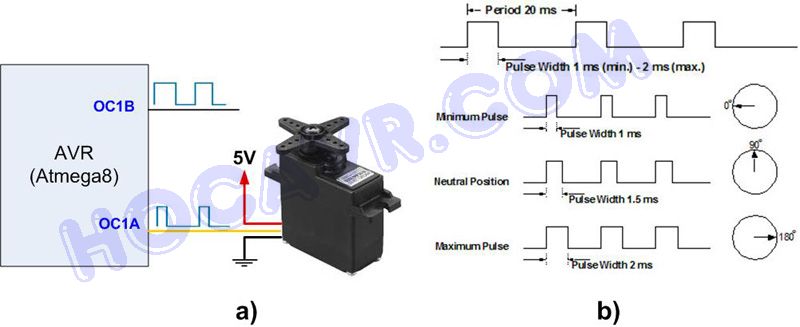
      Các mode 5, 6 và 7 của Fast PWM hoạt động hoàn toàn tương tự mode 14. Điểm khác nhau cơ bản là giá trị TOP(Time period). Trong các mode này giá trị TOP không do thanh thi ICR1 định nghĩa mà là các hằng số không đổi. Với mode 5, tức mode 8 bits, (WGM13=0, WGM12=1, WGM11=0, WGM10=1) giá trị TOP là 1 hằng số, TOP = 255 (số 8 bits lớn nhất). Với mode 6, tức mode 9 bits, (WGM13=0, WGM12=1, WGM11=1, WGM10=0) giá trị TOP là 1 hằng số, TOP = 511 (số 9 bits lớn nhất). Và với mode 7, tức mode 10 bits, (WGM13=0, WGM12=1, WGM11=1, WGM10=1) TOP =1023 (số 10 bits lớn nhất). Mode 15 cũng là Fast PWM trong đó TOP do OCR1A quy định, vì thế mà tín hiệu ra ở kênh A hầu như không phải là 1 xung, nó chỉ thay đổi trạng thái trong 1 clock. Theo tôi, để sử dụng Fast PWM bạn nên dùng mode 14 đã được giải thích trên. Các mode 5, 6, 7 cũng có thể dùng nhưng không nên dùng mode 15.  
       Chúng ta tiến hành viết 1 ví dụ minh họa dùng 2 kênh ở chế độ fast PWM điều khiển 2 động cơ RC servo (gọi tắt là Servo). Mạch điện minh họa như trong hình 11.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiFpR_JMsRrWDNNch1VuRD3mL-OxPPWmeUxNoJAmzFUwSJ9GZm1x3d0pKGxM9OvNJHmkIasb3PNYJlt5dDW9VTo8jdDkqFtXWxxPqBkDnxbgrwCiqPwgzA6c7Ipqc3iPukJaA7STAXGZi4-/s1600/image050.jpg)

Hình 11: Điều khiển 2 RC servo bằng PWM.

       Hai button được nối với 2 ngõ ngắt ngoài INT0 và INT1 để điều khiển góc xoay của 2 Servo. Tên của Servo trong phần mềm Proteus là “MOTOR-PWMSERVO”. Trước khi viết code điều khiển các Servo, bạn cần biết cách điều khiển chúng, tôi giới thiệu ngắn gọn như sau:

       RC servo là một tổ hợp gồm 1 động cơ DC công suất nhỏ, hộp giảm tốc và bộ điều khiển góc quay. Có 2 loại chính là Servo thường và digital Servo, trong ví dụ này tôi giới thiệu Servo thường (phổ biến). Servo thường có 3 dây, dây màu đen là dây GND, dây đỏ là dây nguồn (thường là 5V) và 1 dây trắng hoặc vàng và dây điều khiển (có một số loại Servo có màu dây khác, bạn cần tham khảo datasheet của chúng). Vì các Servo đã có sẵn mạch điều khiển góc quay bên trong nên chúng ta không cần bất cứ giải thuật gì mà chỉ cần cấp tín hiệu PWM cho dây điều khiển là Servo có thể xoay đến 1 vị trí nào đó (chú ý là Servo thường chỉ xoay nữa vòng, điều khiển servo là điều khiển góc xoay chứ không phải điều khiển cận tốc xoay). Hình 12 là hình ảnh servo và cách điều khiển servo.

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEjyslfoo-QgKEyLUL6IO3GRzK61Dk0Jj16sfpqirUe5gNXYER8wpANBV94MAxnw_UT2rFNxHlVyx-N9k8789FXQ6ql80lvWEyykHvJhPmdJYdSHpPn06cvz7Rv-KUKQOB6J26cSjEzvKXzN/s1600/servo.jpg)

Hình 12. Servo và cách điều khiển.

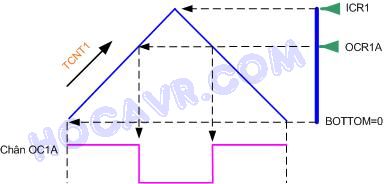
       Bạn xem hình 12b, để điều khiển servo bạn cần cấp cho dây điều khiển một tín hiệu PWM có Time Period khoảng 20ms, duty cycle của PWM sẽ quyết định góc xoay của servo. Với Duty cycle là 1ms, servo xoay về vị trí 0o, khi duty cycle =2ms, góc xoay sẽ là 180o, từ đó bạn có thể tính được duty cycle cần thiết khi bạn muốn servo xoay đến 1 vị trí bất kỳ giữa 0o và 180o. Sau khi hiểu cách điều khiển servo, chúng ta có thể dễ dàng viết code điều khiển chúng, chỉ cần tạo các xung PWM bằng T/C1. Đoạn code cho ví dụ này được trình bày trong list 5.

List 5. Điều khiển Servo bằng PWM.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 | #include <avr/io.h> #include <avr/interrupt.h>  int main(void){        DDRB=0xFF;                //PORTB la output PORT        PORTB=0x00;         MCUCR|=(1<<ISC11)|(1<<ISC01); //ngat canh xuong        GICR |=((1<<INT1)|(1<<INT0); //cho phép 2 ngat hoat dong          TCCR1A=(1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);        TCCR1B=(1<<WGM13)|(1<<WGM12)|(1<<CS10);        OCR1A=1000;              //Duty cycle servo1=1000us=1ms (0 degree)        OCR1B=1500;              //Duty cycle servo2=1500us=1.5ms (90 degree)        ICR1=20000;                  //Time period = 20000us=20ms         sei();                       //set bit I cho phep ngat toan cuc        while (1){           //vòng lặp vô tận                //do nothing        }        return 0; }  //trinh phuc vu ngat ngoai ISR (INT0\_vect ){           if (OCR1A==1000) OCR1A=1500;  //thay doi goc xoay servo1 den 90 do        else OCR1A = 1000;  // thay doi goc xoay servo1 den 0 do } ISR (INT1\_vect ){           if (OCR1B==1000) OCR1B=1500;  //thay doi goc xoay servo1 den 90 do        else OCR1B = 1000;  // thay doi goc xoay servo1 den 0 do } |

       Với ví dụ này tôi chỉ cần giải thích các dòng từ 11 đến 15 liên quan đến việc xác lập chế độ hoạt động Fast PWM mode 14 inverse, phần còn lại bạn đọc tự đối chiếu với các bài trước. Dòng 11 và 12 thực hiện set các bit điều khiển Timer1, trước hết là các bit COM. Bạn thấy tôi chỉ set 2 bit COM1A1 và COM1B1: (1<<COM1A1)|(1<<COM1B1). Hai bit COM1A0 và COM1B0 không set tức mặc định bằng 0. Đối chiếu với bảng 4 bạn thấy chúng ta sẽ dùng “Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at TOP” cho tất cả 2 kênh A và B. Chúng ta set 3 bit  WGM13, WGM12 (thanh ghi TCCR1B, dòng 12) và WGM11 (thanh ghi TCCR1A, dòng 11) như thế thu được tổ hợp (WGM13=1, WGM12=1, WGM11=1, WGM10=0) tức là mode 14 được chọn (bảng 3). Còn lại chúng ta set bit CS10 để khai báo rằng nguồn xung clock cho Timer1 bằng clock cho vi điều khiển (prescaler=1) tức là 1us trong tường hợp f=1Mhz. (nếu bạn dùng các trình biên dịch khác không hỗ trợ định nghĩa tên các bit thì 2 dòng 11 và 12 tương đương: TCCR1A=0xA2; TCCR1B=0x19).  
       Dòng 15 chúng ta khai nhập giá trị cho ICR1 cũng là Time period cho PWM, ICR1=20000 chúng ta thu được Time period =20000 us = 20ms thỏa yêu cầu của servo. Hai dòng 13 và 14 khai báo giá trị ban đầu của các duty cycle của 2 kênh PWM, các giá trị này định vị trí góc xoay của các servo. Trong 2 trình phục vụ ngắt, các giá trị này được thay đổi khi các button được nhấn.

***2.3 Phase correct PWM (PWM với pha chính xác).***        Phase correct PWM cung cấp một chế độ tạo xung PWM có độ phân giải cao (high resolution) nên được gọi là Phase correct PWM. Tương tự Fast PWM, cũng có 5 mode hoạt động thuộc Phase correct PWM đó là các mode 1, 2, 3, 10 và 11 (xem bảng 3). Năm mode này tương ứng các mode 5, 6, 7, 14 và 15 của fast PWM. Về cách điều khiển, Phase correct hầu như giống fast PWM, nghĩa là nếu bạn đã biết cách sử dụng các mode của fast PWM thì bạn sẽ hoàn toàn điều khiển được Phase correct PWM. Khác nhau cơ bản của 2 chế độ này là trong cách hoạt động, nếu Fast PWM có chu kỳ hoạt động trong 1 single-slope (một sườn) thì Phase correct PWM lại dual-slope (hai sườn). Lấy ví dụ mode 10 của Phase correct PWM tương ứng với mode 14 của Fast PWM, trong mode này thanh ghi ICR1 chứa TOP và OCR1A (hoặc OCR1B đối với kênh B) chứa giá trị so sánh. Khi hoạt động, thanh ghi TCNT1 tăng từ 0, khi TCNT1 bằng với OCR1A thì chân OC1A được xóa xuống mức LOW (tôi đang nói trường hợp COM1A1=1, COM1A0=0), TCNT1 tiếp tục tăng đến TOP, khi TCNT1=TOP thì TCNT1 KHÔNG được tự động reset về 0 như trường hợp Fast PWM mà TCNT1 bắt đầu đếm ngược, tức giảm từng giá trị từ TOP về 0. Trong lúc TCNT1 giảm, đến 1 lúc nó sẽ bằng giá trị của OCR1A lần thứ 2, và lần này, chân OC1A được set lên mức HIGH, TCNT1 tiếp tục giảm đến 0 thì 1 chu kỳ hoàn tất. Rõ ràng 1 chu kỳ là quá trình đếm trong 2 “sườn” nên ta gọi Phase correct PWM là dual-slope. Cũng vì tính chất dual-slope mà tín hiệu PWM trong chế độ này có tính đối xứng, thích hợp cho các ứng dụng điều khiển động cơ. Hình 13 mô tả cách mà Phase correct PWM hoạt động tron mode 10 với ngõ ra đảo (COM1A1=1, COM1A0=0).

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEiosGhQkTcm_Bi0UJIqfbS-lj1flW1wOeEdJxaq_-SAp6PglhRIPDftN9YZ98mMC3-EYmT2Abq7YdaFQ-oGwZkwsSdz6KXlhXuu0cIF-RSajG0-JUMKPrw_xSidCwO_GxL2cEkyy6vN3lEN/s1600/correctPWM.jpg)

Hình 13. Phase correct PWM mode 10.

       Việc viết code cho chế độ Phase correct PWM gần như tương tự fast PWM, bạn chỉ cần thay đổi tổ hợp các bit WGM dựa theo bảng 3 và sau đó nhâp các giá trị phù hợp cho ICR1 và ORC1A, OCR1B là được.

***2.3 Phase correct and frequency correct PWM.***  
        Chế độ này có 2 mode là 8 và 9. Về hầu hết các phương diện, 2 mode này giống với 2 mode 10 và 11 của Phase correct PWM. Cái khác nhau duy nhất là thời điểm mà thanh ghi OCR1A và OCR1B được cập nhật dữ liệu nếu có sự thay đổi. Việc này, nhìn chung không ảnh hưởng đến hầu hết người dùng PWM để điều khiển. Bạn sẽ rất khó để thấy sự khác biệt nếu bạn không phải đang viết 1 ứng dụng mà sai số trong 1 micro giây là điều tệ hại. Vì thế tôi không đề cập chi tiết chế độ này, bạn đọc có thể tham khảo datasheet của chip để hiểu rõ hơn nếu cần thiết.

       Ngoài ra trên chip atmega8 còn có bộ timer2 8 bits có PWM và asynchronous operation. Về mặt chức năng timer2 giống như phiên bản 8 bit của timer1 (độ phân giải thấp hơn nhưng có cùng chế độ và phương thức hoạt động). Điểm khác biệt và cũng là điểm đặc biệt của Timer2 là khả năng hoạt động không đồng bộ với chip, nó giống như việc bạn tách timer2 ra thành 1 chip timer riêng, vì thế cần cung cấp 1 nguồn xung clock khác cho timer này (1 thạch anh khác). Chế độ này có thể được dùng để calip (calibrate), canh chỉnh sai số  và bù cho nguồn xung clock chính trên chip.