LỜI MỞ ĐẦU

Theo xu hướng phát triển của xã hội ngày nay, ngành tự động hóa là một trong những ngành không thể thiếu, kĩ thuật ngày càng phát triển con người lại mong muốn tìm đến những thiết bị hoạt động theo hướng tự động hóa với mục đích nâng cao chất lượng cuộc sống. Hiện nay nhiệt độ trái đất đang tăng cao do đó người ta sử dụng nhiều phương pháp chống nóng khác nhau. Biện pháp thường hay được sử dụng là quạt điện chỉ sử dụng được cho một diện tích nhỏ, khi nhiệt độ trong phòng tăng cao dễ khiến cho cơ thể mất nước và mệt mỏi. Sử dụng điều hòa thì cần chi phí lớn. Trong khi phương pháp ít được mọi người sử dụng đến là phun sương làm mát cho tòa nhà, trong chăn nuôi hoặc nhà xưởng có diện tích lớn…, với giá thành rẻ, chi phí hoạt động thấp.

Xuất phát từ những thực tế đó, em đã mạnh dạn nêu ra ý tưởng của mình và được thầy, cô chấp nhận đề tài "**Thiết kế, xây dựng hệ thống phun sương làm mát tự động".** Trong đề tài này em đã sử dụng vi điều khiển AVR với những tính năng mạnh mẽ và giá thành rẻ. Một cảm biến nhiệt độ để đo nhiệt độ môi trường.

Đồ án gồm các nội dung sau:

Chương 1: Tổng quan về vi điều khiển ATmega8

Chương 2: Ngôn ngữ lập trình C và phần mềm lập trình CodevisionAVR

Chương 3: Thiết kế và xây dựng hệ thống phun sương làm mát tự động

Sinh viên thực hiện

Trịnh Minh Đồng

CHƯƠNG 1:

TỔNG QUAN VỀ VI ĐIỀU KHIỂN ATMEGA8

1.1. GIỚI THIỆU VỀ AVR

AVR là các vi điều khiển 8 bits với cấu trúc tập lệnh đơn giản hóa RISC (Reduced Instruction Set Computer) có cấu trúc Harvard được phát triển bởi Atmel năm 1996. AVR là một trong những họ vi điều khiển đầu tiên dùng bộ nhớ flash tích hợp trên chip để chứa chương trình, khác với ROM (chỉ có thể lập trình một lần), EPROM, hay EEPROM được dùng cho các họ vi điều khiển khác cùng thời điểm đó.

Hình 1.1: Atmel AVR ATmega8.

Lịch sử phát triển AVR:

Mọi người vẫn tin rằng kiến trúc cơ bản của AVR được hình thành từ hai sinh viên của trường đại học Norwegian Institure of Technology tên là Alf-BgilBogen và Vegard Wollan.

Ban đầu AVR MCU (Micro Controller Unit) được phát triển tại một phòng ASIC (Application Specific IC) ở Trondheim Nauy, đó là nơi mà 2 người sáng lập của Atmel Nauy làm việc như là sinh viên. Và nó được biết đến với tên µRISC (Micro RISC). Khi công nghệ này được bán cho Atmel,

cấu trúc bên trong AVR được phát triển xa hơn bởi Alf và Vegard tại Atmel Nauy, một công ty con của Atmel được thành lập bởi 2 thành viên trên.

Một sản phẩm đầu tiên của AVR là AT90S8515, cũng có đóng gói DIP 40 chân giống như 8051, nó bao gồm phức hợp địa chỉ các thành phần bên ngoài và data bus. Điều khác biệt là chân RESET (8051 RESET tích cực mức cao, AVR lại tích cực mức thấp), ngoại trừ điểm này, các ngõ ra đều giống nhau.

AVRs thường được chia thành 6 nhóm lớn:

- **TinyAVRs :**

- 1-8 kB bộ nhớ chương trình.
- 8-32 chân.
- Hạn chế các thiết bị ngoại vi.

- **MegaAVRs:**

- 4-256 kB bộ nhớ chương trình.
- 28-100 chân.
- Mở rộng tập lệnh.
- Nhiều thiết bị ngoại vi.

- **XmegaAVRs:**

- 16-256 kB bộ nhớ chương trình.
- 44-64-100 chân.
- Mở rộng các thiết bị như DMA, "Event System", và hỗ trợ mã hóa .
- Mở rộng thiết bị ngoại vi với DACs.

- **Ứng dụng cụ thể của AVR:**

MegaAVRs với các tính năng đặc biệt không tìm thấy trên các thành viên khác của họ AVR, chẳng hạn như màn hình LCD controller, USB controller, advanced PWM, CAN…

- **FPSLIC (AVR với FPGA):**

- FPGA 5K đến 40K cổng.

- SRAM cho mã chương trình AVR, không giống như tất cả các AVRs khác.
- AVR cốt lõi có thể chạy lên đến 50 MHz.

- **32-bit AVRs:**

Năm 2006, Atmel phát hành vi điều khiển dựa trên, kiến trúc 32-bit, AVR. Chúng bao gồm các hướng dẫn SIND và DSP , cùng với âm thanh và các tính năng xử lý video. Họ 32-bit của các thiết bị này được thiết kế để cạnh tranh với các bộ vi xử lý dựa trên ARM. Các tập lệnh tương tự như các lõi RISC khác, nhưng không tương thích với AVR ban đầu hoặc bất kỳ các lõi ARM khác nhau.

1.2. CHI TIẾT VỀ CHIP ATMEGA8

1.2.1. Tổng quan

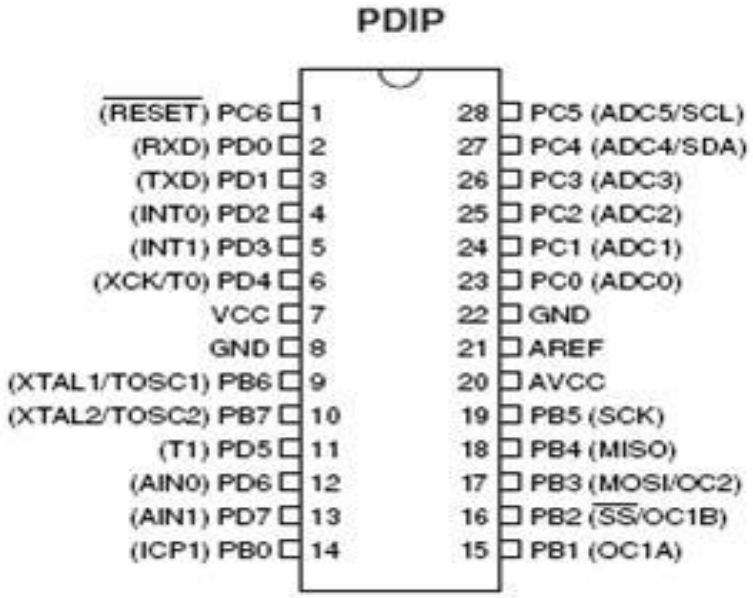
Những tính năng chính của Atmega8:

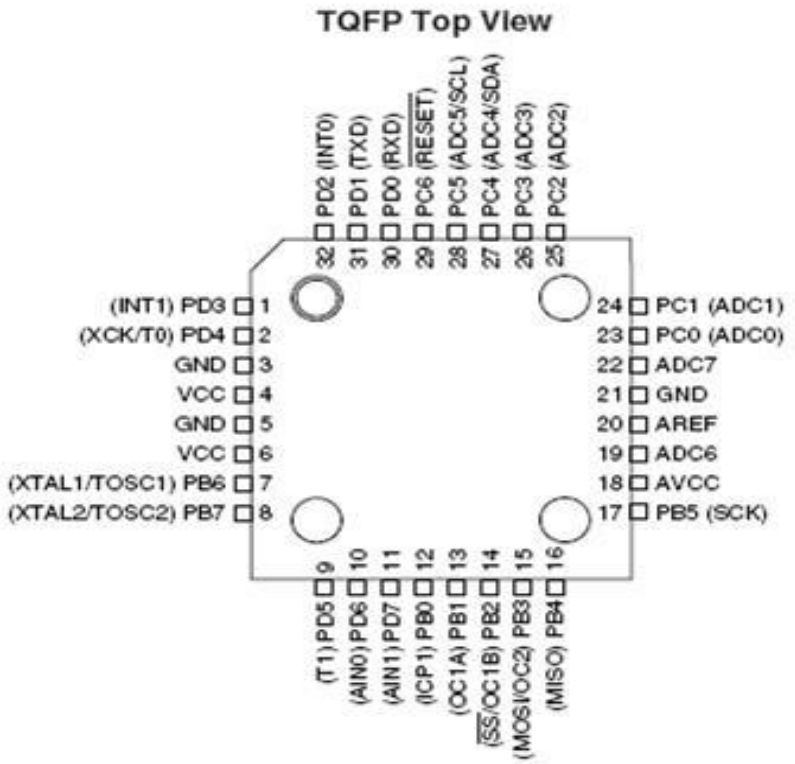
- Rom: 8 Kbyte bộ nhớ flash
- Sram: 1 Kbytes nội
- EEPROM: 512 bytes
- Có thể xóa lập trình được và có thể chịu được 10000 lần ghi xóa
- Có 28 chân, trong đó có 23 cổng vào/ra
- 160 thanh ghi vào ra mở rộng
- 32 thanh ghi đa mục đích 8 bit
- 2 bộ định thời 8 bit
- 1 bộ định thời 16 bit
- Bộ định thời watchdog
- Bộ dao động nội RC tần số 1 MHz, 2 MHz, 4 MHz, 8 MHz
- ADC 6 kênh với độ phân giải 10 bit (Ở dòng Xmega lên tới 12 bit)
- 3 kênh PWM 8 bit
- Bộ so sánh tương tự có thể lựa chọn ngõ vào
- Khối USART lập trình được

- Khối truyền nhận nối tiếp SPI
- Hỗ trợ Boot loader
- 5 chế độ tiết kiệm năng lượng
- Lựa chọn tần số hoạt động bằng phần mềm
- Tần số tối đa 16MHz
- Nguồn nuôi từ 2,7 – 5,5 đối với ATmega8L và từ 4,5V – 5,5V đối với ATmega8
- Dòng làm việc 3,6 mA

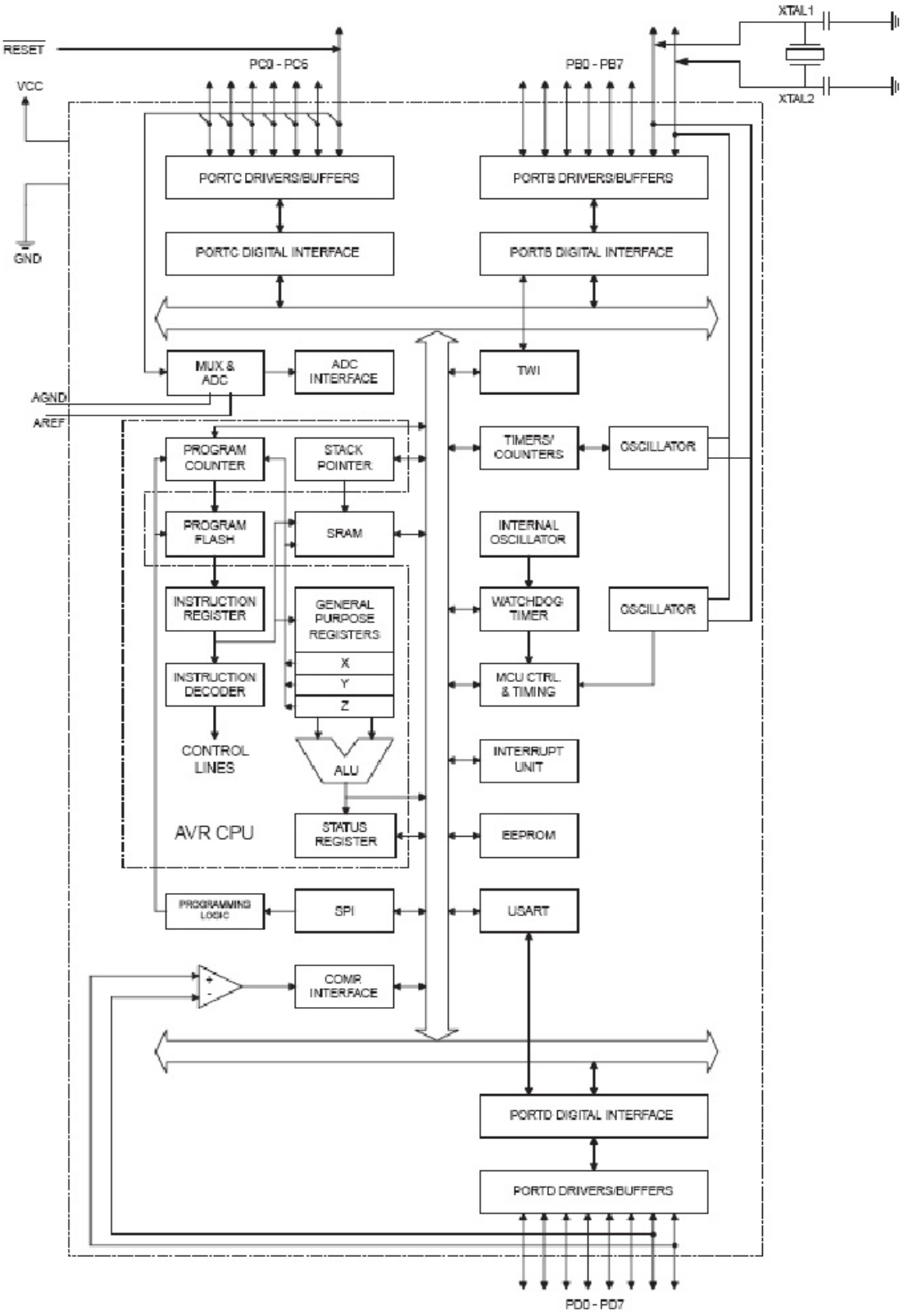
Hình 1.2: Hình ảnh các loại AVR.

Sơ đồ bố trí chân của ATmega8 gồm 2 kiểu:



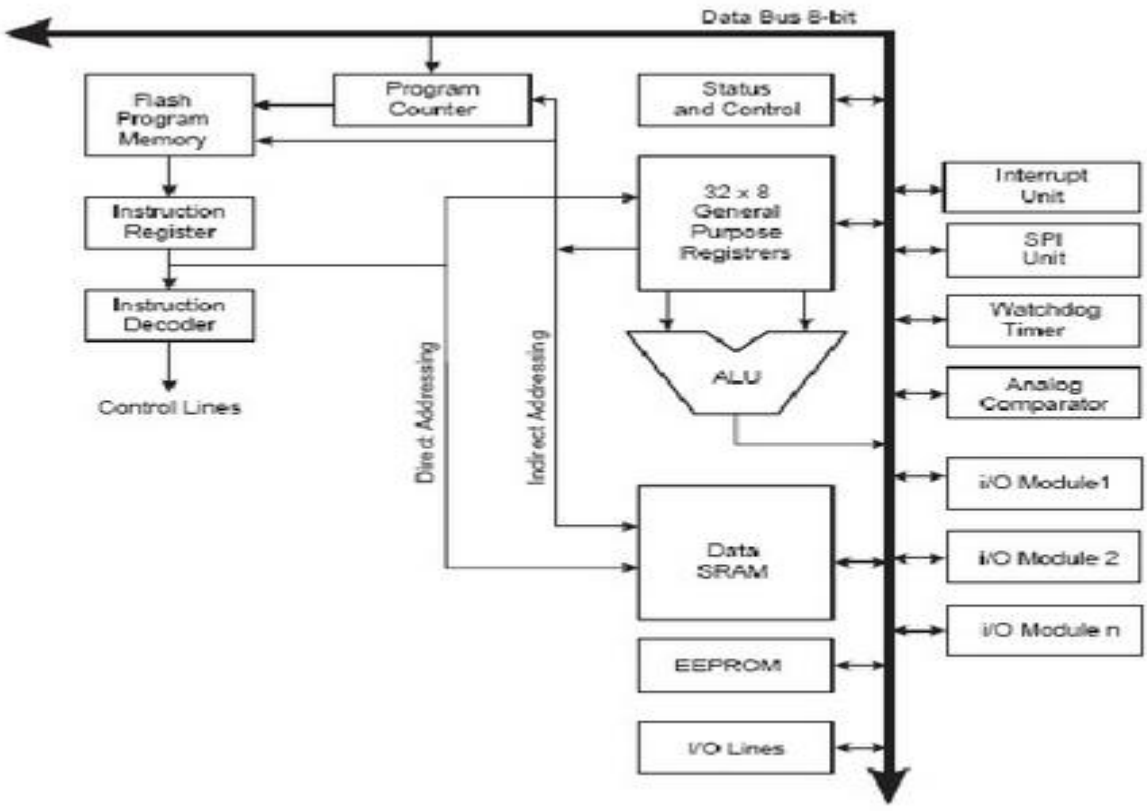


Hình 1.3: Sơ đồ bố trí chân của các dạng ATmega8.



Hình 1.4: Cấu tạo bên trong của ATmega8.

1.2.2. Cấu trúc cơ bản của ATmega8



Hình 1.5: Sơ đồ khối cấu trúc vi điều khiển AVR.

1.2.2.1. Cấu trúc bộ nhớ

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 128K bytes. Bộ nhớ chương trình có độ rộng Bus là 16 bit. Những địa chỉ đầu tiên của bộ nhớ chương trình được dùng trong bảng vecto ngắt. Đối với Atmega 8 bộ nhớ chương trình có thể chia làm 2 phần: phần boot loader (Boot loader program section) và phần ứng dụng (Application program section).

- ***Bộ nhớ dữ liệu:***

Bộ nhớ dữ liệu của AVR được chia làm hai 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 dung lượng 1 Kbytes, bộ nhớ SRAM có hai chế độ hoạt động là chế độ thông thường và chế độ tương thích với ATmega8.

- ***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 cung cấp bị mất. Với vi điều khiển AT mega8, bộ nhớ EEPROM có kích thước là 512 byte. EEPROM được xem như là một bộ nhớ vào ra được đánh địa chỉ độc lập với SRAM. Để điều khiển vào ra dữ liệu với EEPROM ta sử dụng ba thanh ghi:

+ Thanh ghi EEAR (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	
ĐỌC / VIẾT	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	
GIÁ TRỊ BẮT ĐẦU	0	0	0	0	X	X	X	X	
	X	X	X	X	X	X	X	X	

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

+ Thanh ghi EEDR:

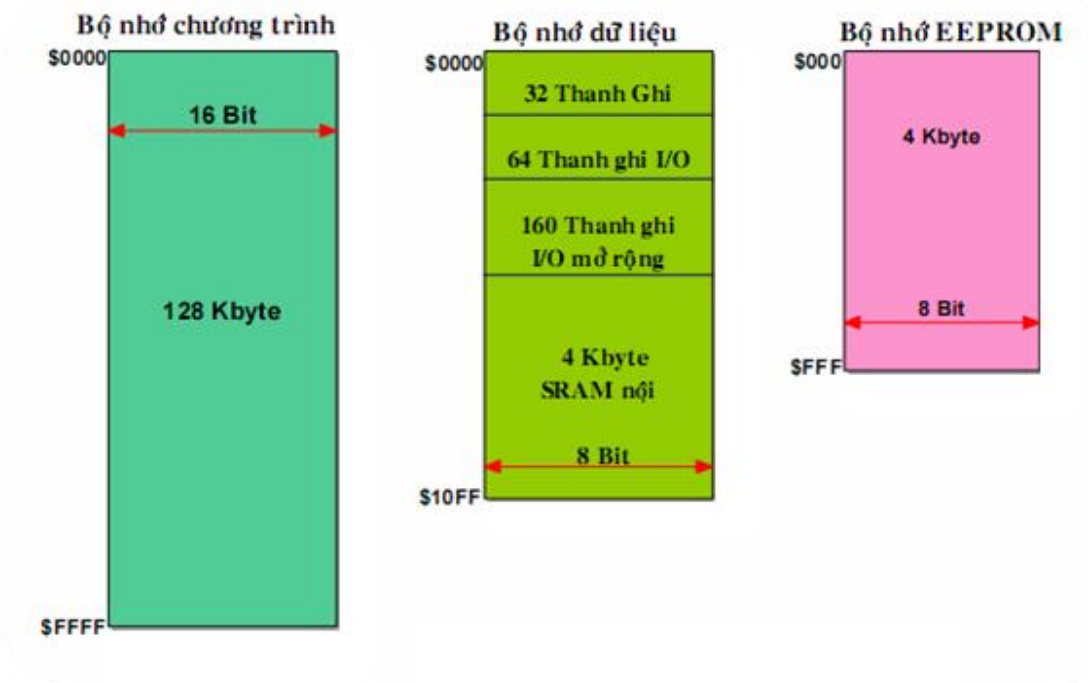
BIT	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
ĐỌC / VIẾT	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
GIÁ TRỊ BẮT ĐẦU	0	0	0	0	0	0	0	0	

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

+ Thanh ghi EECR:

BIT	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
ĐỌC/ VIẾT	R	R	R	R	R/W	R/W	R/W	R/W	
GIÁ TRỊ BẮT ĐẦU	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, bốn bit cuối là dự trữ.



Hình 1.6: Tóm tắt bản đồ bộ nhớ bên trong ATmega8.

1.2.2.2. Cổng vào/ra (I/O)

Có thể thấy chip này gồm 28 chân , bao gồm 4 PORT vào ra. Cổng vào ra là một trong số các phương tiện để vi điều khiển giao tiếp với các thiết bị ngoại vi. Ở ATmega8 có tất cả các cổng vào ra 8 bit là: PORTB, PORTC, PORTD Các cổng vào ra của AVR là cổng vào hai chiều có thể định hướng, tức có thể chọn hướng của cổng là hướng vào (input) hay hướng ra (output). Tất cả các cổng vào ra của AVR đều có chức 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 một chân nào

đó thì nó không làm ảnh hướng tới hướng của các chân khác. Tất cả các chân của các PORT đều 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 (VCC – Vdd) và đầu còn lại được nối với tín hiệu lối vào/ra của một mạch logic chức năng.

- **Thanh ghi DDRx:**

Đây là thanh ghi 8 bit (có thể đọc/ghi) có khả năng điều khiển hướng của cổng (lối vào hay lối ra). 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.

- **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.

- **Thanh ghi PINx:**

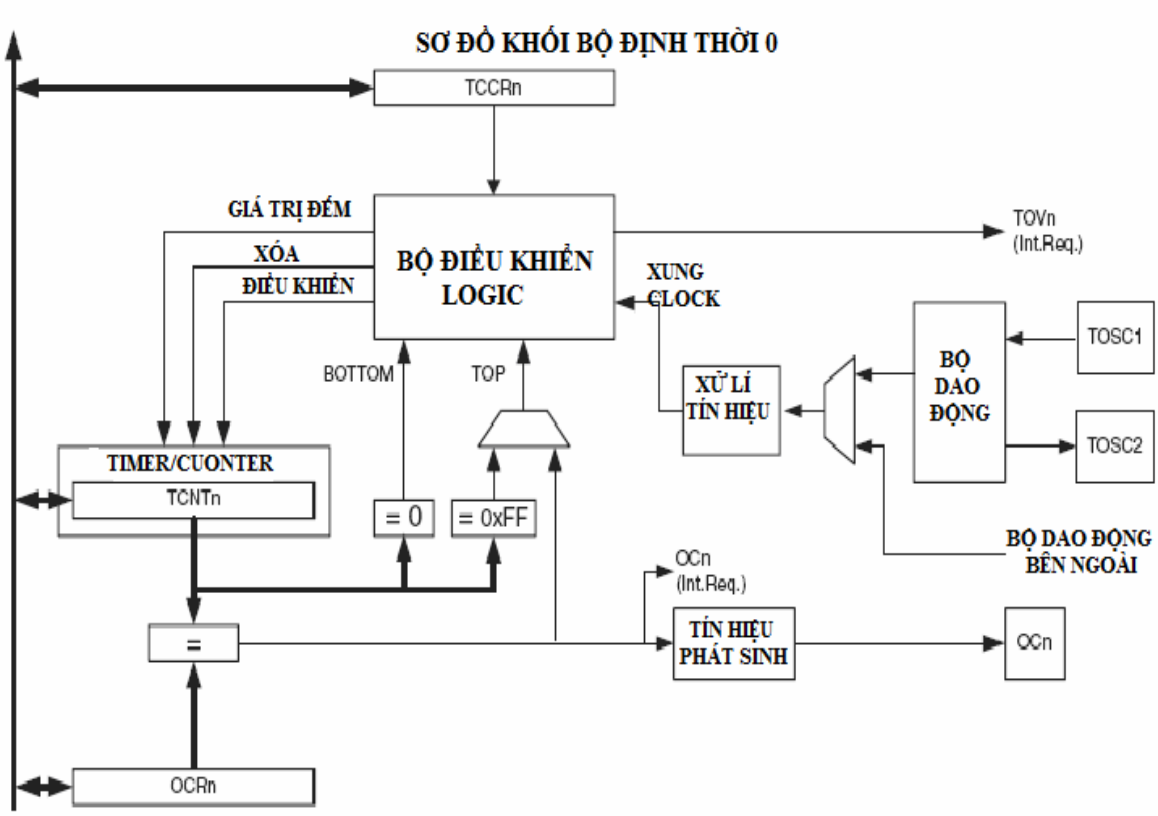
PINx không phải là một thanh ghi thật sự, đây là địa chỉ trong bộ nhớ I/O kết nối trực tiếp với các chân của cổng. Khi ta đọc PORTx tức ta đọc dữ liệu được chốt trong PORTx, còn khi đọc PINx thì giá trị logic hiện thời ở chân của cổng tương ứng được đọc. Vì thế đối với thanh ghi PINx ta có thể đọc mà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

1.2.3. Bộ định thời của ATmega8

Timer/Counter (T/C) là các module độc lập với CPU. Chức năng chính của các bộ T/C, như tên gọi của chúng, là định thời (tạo ra một khoảng thời

gian, đếm thời gian…) và đếm sự kiện. Trên các chip AVR, các bộ T/C còn có thêm chức năng tạo ra các xung điều rộng PWM (Pulse Width Modulation), ở một số dòng AVR, một số T/C còn được dùng như các bộ canh chỉnh thời gian (calibration) trong các ứng dụng thời gian thực. Các bộ T/C 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 T/C cũng không hoàn toàn giống nhau.

1.2.4. Bộ định thời 0:



Hình 1.7: Sơ đồ khối bộ định thời 0.

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

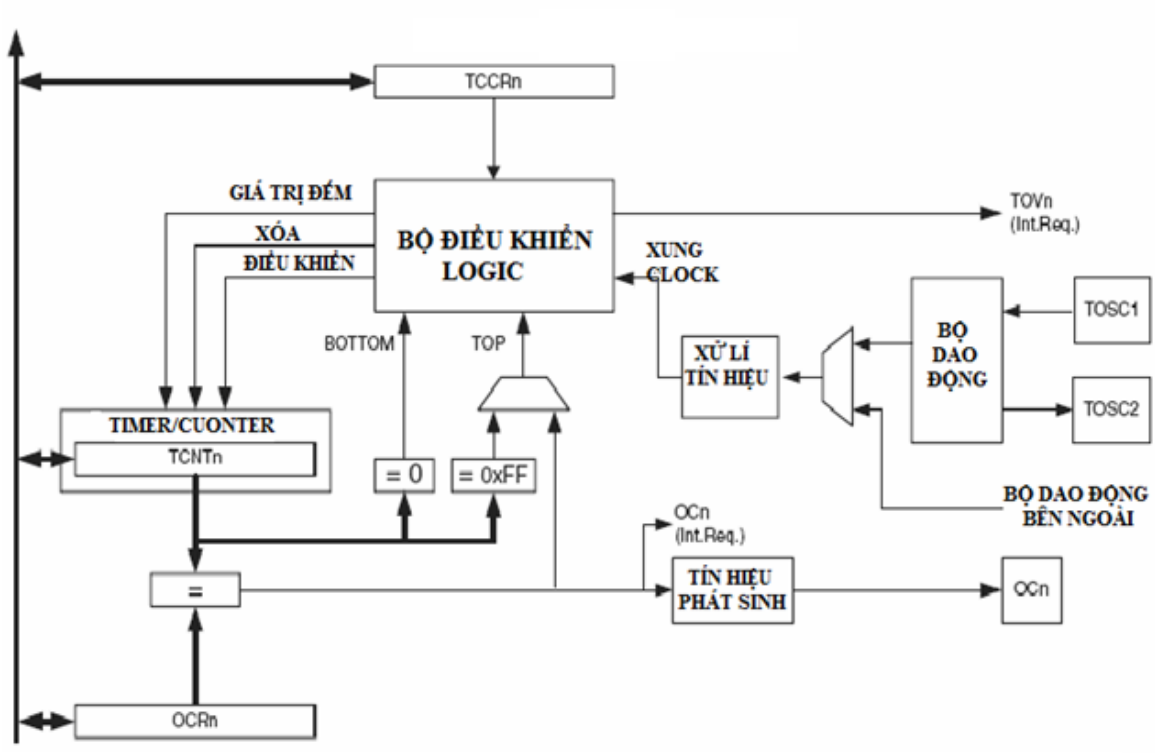
- **BOTTOM:** Bộ đếm đạt tới giá trị BOTTOM khi nó có giá trị 00h.
- **MAX:** Bộ đếm đạt tới giá trị Max khi nó bằng FFh.

- **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 kỳ giá trị nào được qui định trong thanh ghi OCRn (n=0,2), tùy theo chế độ thực thi.

Các thanh ghi trong bộ định thời 0 bao gồm:

- Thanh ghi Timer/Counter Control Register – TCCR0
- Thanh ghi Timer/Counter Register - TCNT0
- Thanh ghi Output Compare Register – OCR0
- Thanh ghi Timer/Counter Interrupt Mask Register – TIMSK
- Thanh ghi Timer/Counter Interrupt Flag Register – TIFR
- Thanh ghi Special Function IO Register – SFIOR
- Thanh ghi Asynchronous Status Register – ASSR

1.2.5. Bộ định thời 2



Hình 1.8: Sơ đồ khối bộ định thời 2.

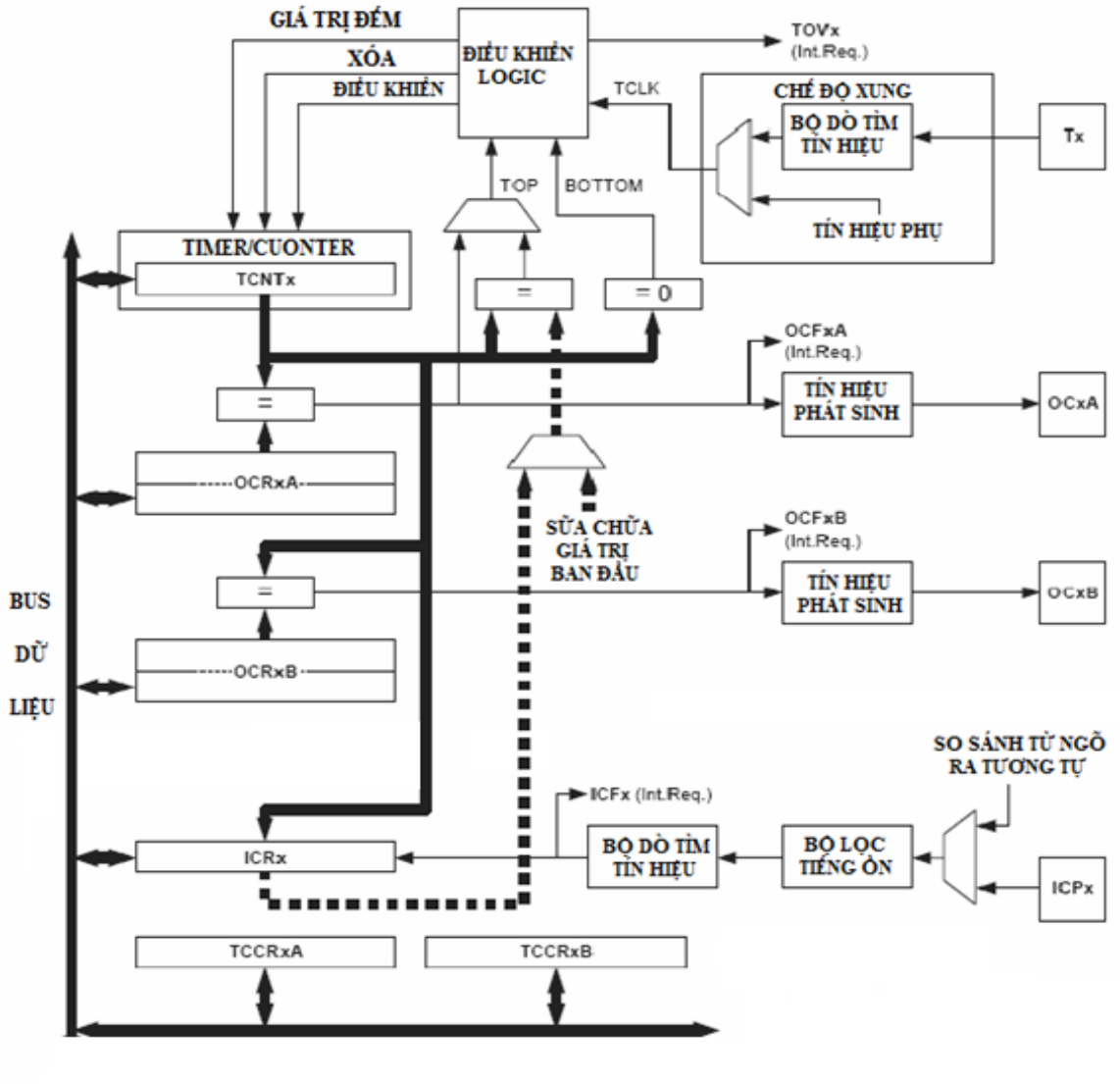
Bộ định thời 2 là bộ định thời 8 bit, bộ định thời 2 liên quan tới 5 thanh ghi với nhiều chế độ thực thi khác nhau. Thuộc tính chính của bộ định

thời 2 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 trong bộ định thời 2:

- Thanh ghi Timer/Counter Control Register – TCCR2
- Thanh ghi Timer/Counter Register – TCNT2
- Thanh ghi Output Compare Register – OCR2
- Thanh ghi Timer/Counter Interrupt Mask Register – TIMSK
- Thanh ghi Timer/Counter Interrupt Flag Register – TIFR

1.2.6. Bộ định thời 1:

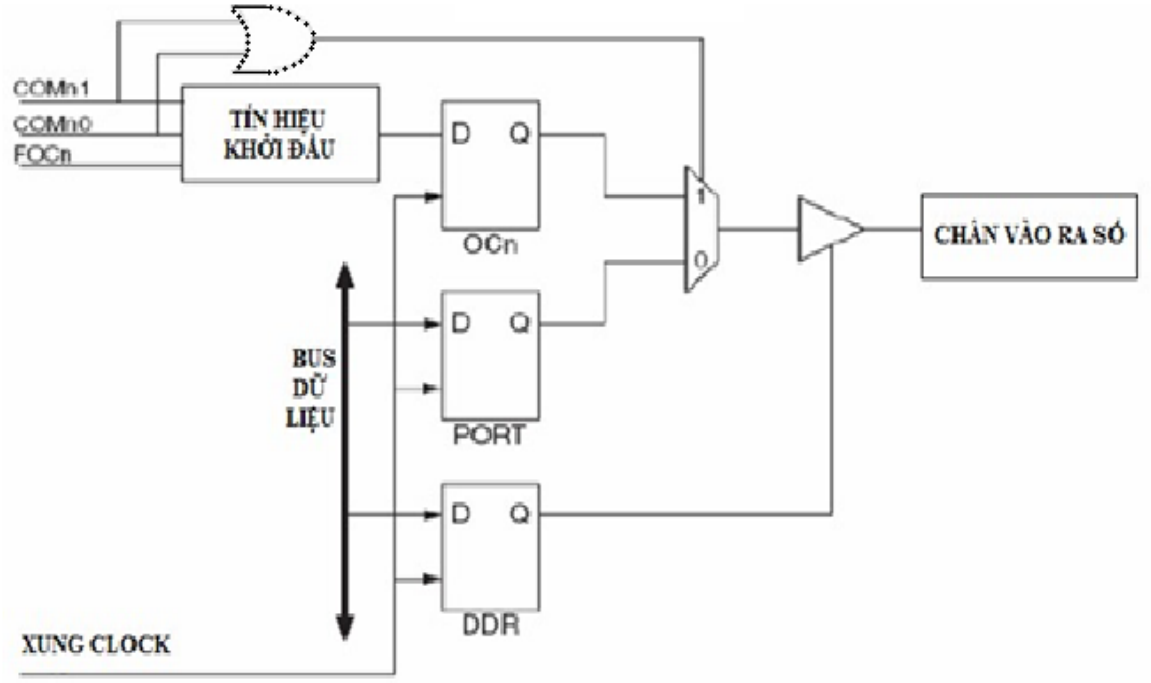


Hình 1.9: Sơ đồ khối bộ định thời 1.

Các định nghĩa sau sẽ được sử dụng trong bộ định thời 1:

- **BOTTOM:** Bộ đếm đạt đến giá trị BOTTOM khi nó có giá trị 0000h.
- **MAX:** Bộ đếm có 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 cứ giá trị nào được qui định trong thanh ghi OCRnX (X=A,B) hay ICRn, tùy theo chế độ thực thi.

Ngõ ra khối Compare Match Output Unit:

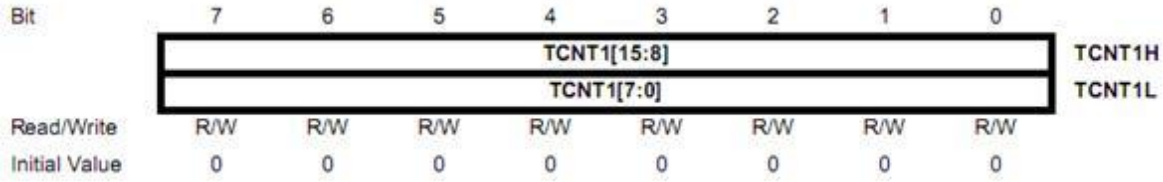


Hình 1.10: Sơ đồ ngõ ra khối.

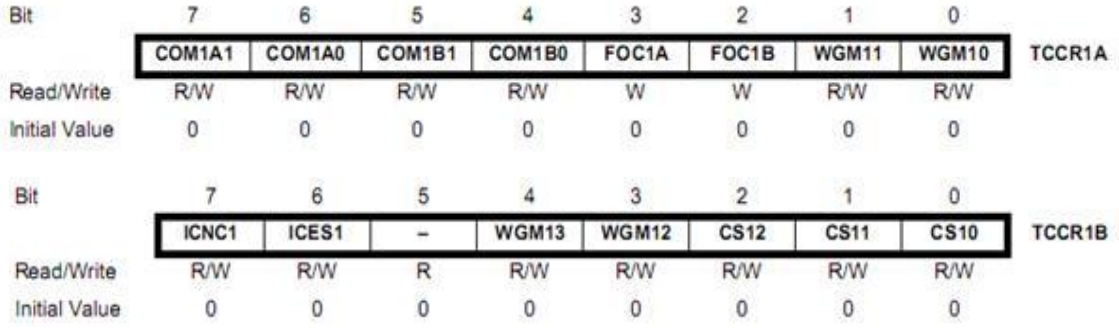
Nhìn hình ta thấy Pin OCnX (chẳn hạn pin 15 của IC tương ứng với OC1A), là ngõ ra của khối Compare Match Output Unit, có thể được nối với 3 thanh ghi là OCnX, PortX và DDRX. Thanh ghi nào được nối với OCn là phụ thuộc vào các bit COMn1:0 (tức là tùy theo chế độ hoạt động của bộ định thời). Nếu ta thiết lập bộ định thời hoạt động ở chế độ thường (tức không sử dụng chức năng so sánh khớp) thì chân OCn trở thành chân vào ra số thông thường.

Bộ định thời 1 bao gồm các thanh ghi:

- 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 đọc và ghi giá trị một cách trực tiếp.



- TCCR1A và TCCR1B (Timer/Counter Control Register): là 2 thanh ghi điều khiển hoạt động của T/C1. Tất cả các chế độ 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:

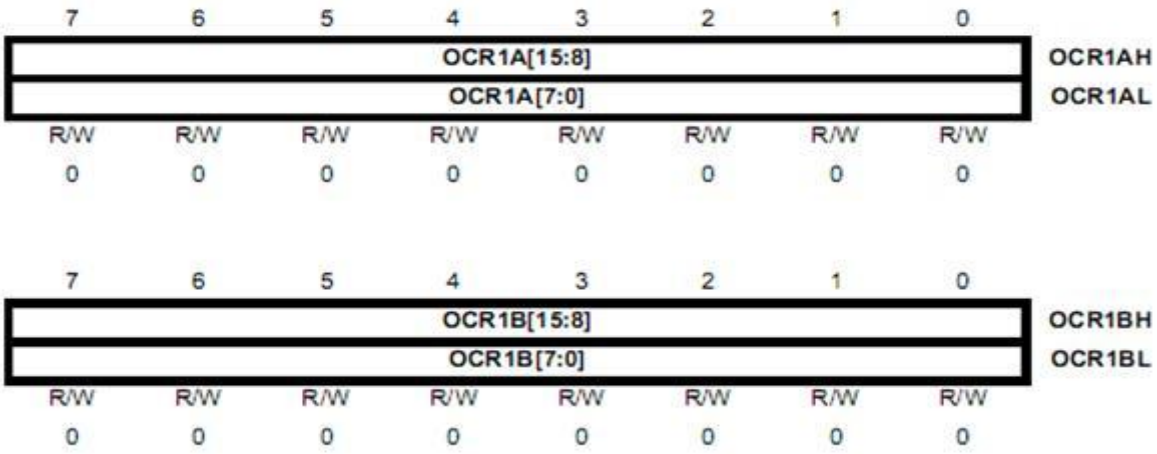


Nhìn chung để "thuộc" hết cách phối hợp các bit trong 2 thanh ghi TCCR1A và TCCR1B là tương đối phức tạp vì T/C1 có rất nhiều mode hoạt động. Ở đây, trong thanh ghi TCCR1B có 3 bit khá quen thuộc là CS10, CS11 và CS12. Đây là các bit chọn xung nhịp cho T/C1 như truong T/C0.

Bảng 1.1: Tóm tắt các chế độ chọn xung nhịp trong T/C1.

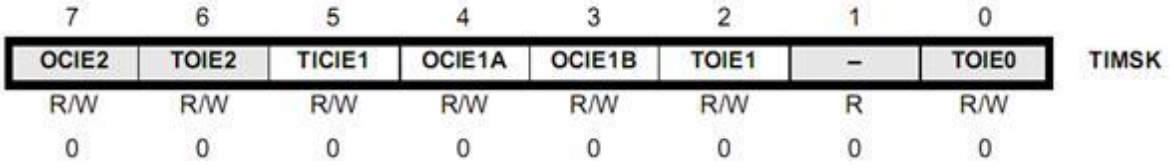
CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

- 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 (ngõ so sánh ra). 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). A và B đại diện cho 2 kênh (channel). 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.



- 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), thanh ghi ICR1 sẽ "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 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).



7	6	5	4	3	2	1	0	
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
0	0	0	0	0	0	0	0	

- 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.

- 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.

7	6	5	4	3	2	1	0	
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0	TIFR
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
0	0	0	0	0	0	0	0	

Các chế độ hoạt động: có tất cả 5 chế độ hoạt động chính trên T/C1. Các chế độ (mode) 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 WGM 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:

Bảng 1.2: Các bit WGM và các chế độ hoạt động của T/C1.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation[1]	TOP	Update of OCR1x	TOV1 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	OCR1A	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	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

- ***Tìm hiểu chế độ Fast PWM (PWM tần số cao) - chế độ 14:***

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 (ở bảng trên). 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.

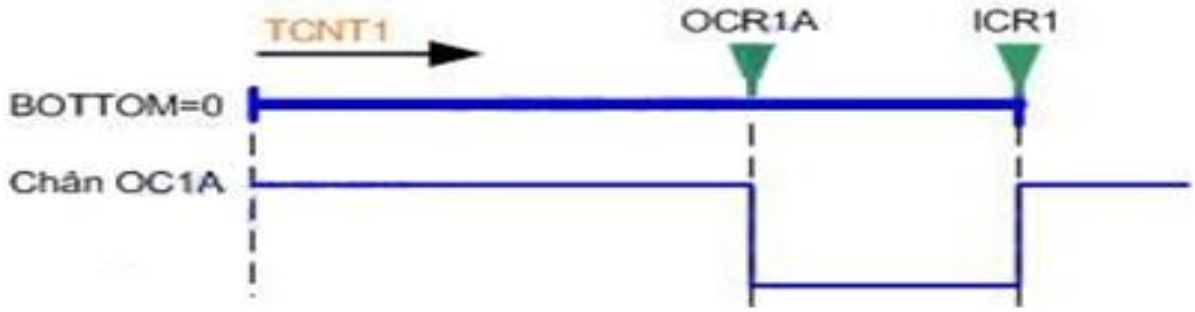
Chú ý các bit COM1A1, COM1A0 và COM1B1, COM1B0 là các bit chọn dạng tín hiệu ra của PWM (Compare Output Mode bits). COM1A1, COM1A0 dùng cho kênh A và COM1B1, COM1B0 dùng cho kênh B. Hãy đối chiếu bảng dưới:

Bảng 1.3: Mô tả các bit COM trong chế độ fast PWM .

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at TOP
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at TOP

Ví dụ cụ thể về chế độ (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, COM1A=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 ta thấy trong hàng 4 bảng trên. Hình dưới mô tả cách tạo xung PWM trên chân OC1A ở mode 14.



Hình 1.11: Fast PMW mode 14.

1.3. CẤU TRÚC NGẮT CỦA ATMEGA8

1.3.1. 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 dịch vụ ngắt ISR (Interrupt Service Routine) hay còn gọi là trình quản lí 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 thỏa mãn thì nó phục vụ thiết bị. Sau đó chuyển sang hiển thị trạng thái của thiết bị kế tiếp cho đến khi tất cả đều được phục vụ.

Mặc dù phương pháp thăm dò có thể hiển thị tình trạng của một vài thiết bị và phục vụ mỗi thiết bị khi các điều kiện nhất định được thỏa mãn nhưng nó không tận dụng hết công dụng của bộ vi điều khiển. Điểm mạnh của phương pháp ngắt là bộ vi điều khiển có thể phục vụ 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 cho các thiết bị vì nó kiểm tra tất cả các thiết bị theo kiểu xoay vòng. Quan trọng hơn là trong phương pháp ngắt thì bộ vi điều khiển cũng 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 ưa 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.

1.3.2. Trình phục vụ ngắt của bảng Vector ngắt

Đối với mỗi ngắt thì phải có một trình phục vụ ngắt ISR (Interrupt Service Ruotine) 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 lại 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 thì 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.

1.3.3. Bảng Vector ngắt của ATmega8

Đây là bảng véc tơ ngắt của Atmega8, cùng với địa chỉ của nó trong bộ nhớ chương trình.

Bảng 1.4: Bảng vector ngắt của Atmega8.

Số vector	Địa chỉ	Nguồn (điểm gốc)	Ý nghĩa
1	$0000	RESET	Reset AVR
2	$0002	INT0	Ngắt ngoài 0
3	$0004	INT1	Ngắt ngoài 1
4	$0006	INT2	Ngắt ngoài 2
5	$0008	INT3	Ngắt ngoài 3
6	$000A	INT4	Ngắt ngoài 4
7	$000C	INT5	Ngắt ngoài 5
8	$000E	INT6	Ngắt ngoài 6
9	$0010	INT7	Ngắt ngoài 7
1	$0012	TIMER2 COMP	So sánh Timer/Counter 2
1	$0014	TIMER2 OVF	Báo tràn Timer/ Counter 2
1	$0016	TIMER1 COMPA	Sử dụng Timer/ Counter 1
1	$0018	TIMER1 COMPA	So sánh Timer/Counter1 (A)
1	$001A	TIMER1 COMPB	So sánh Timer/Counter1 (B)
1	$001C	TIMER1 OVF	Báo tràn Timer/Counter 1
1	$001E	TIMER0 COMP	So sánh Timer/Counter0
1	$0020	TIMER0 OVF	Báo tràn Timer/Counter0
1	$0022	SPI.STC	Khối truyền nhận nối tiếp
1	$0024	USART0. RX	Bộ truyền dữ liệu nối tiếp 0
2	$0026	USART0.UDRE	Bộ dữ liệu trống USART0
2	$0028	USART0.TX	Bộ truyền dữ liệu nối tiếp TX
2	$002A	ADC	Bộ chuyển đổi ADC
2	$002C	EE READY	Bộ nhớ EEPROM
2	$002E	ANALOG COMP	So sánh tín hiệu tương tự
2	$0030	TIMER1 COMPC	So sánh Timer/Cuonter1 (C)
2	$0032	TIMER3 CAPT	Sử dụng Timer/Cuonter3
2	$0034	TIMER3 COMPA	So sánh Timer/Cuonter3 (A)
2	$0036	TIMER3 COMPB	So sánh Timer/Cuonter3 (B)
2	$0038	TIMER3 COMPC	So sánh Timer/Counter3 (C)
3	$003A	TIMER3 OVF	Báo tràn Timer 3
3	$003C	USART1.RX	Bộ truyền dữ liệu nối tiếp 1
3	$003E	USART1.UDRE	Bộ dữ liệu rỗng USART1
3	$0040	USART1.TX	Bộ truyền dữ liệu nối tiếp 1

3	$0042	TWI	Hai giá trị bên ngoài
3	$0044	SPM READY	Bộ nhớ chương trình

1.3.4. Thứ tự ưu tiên ngắt

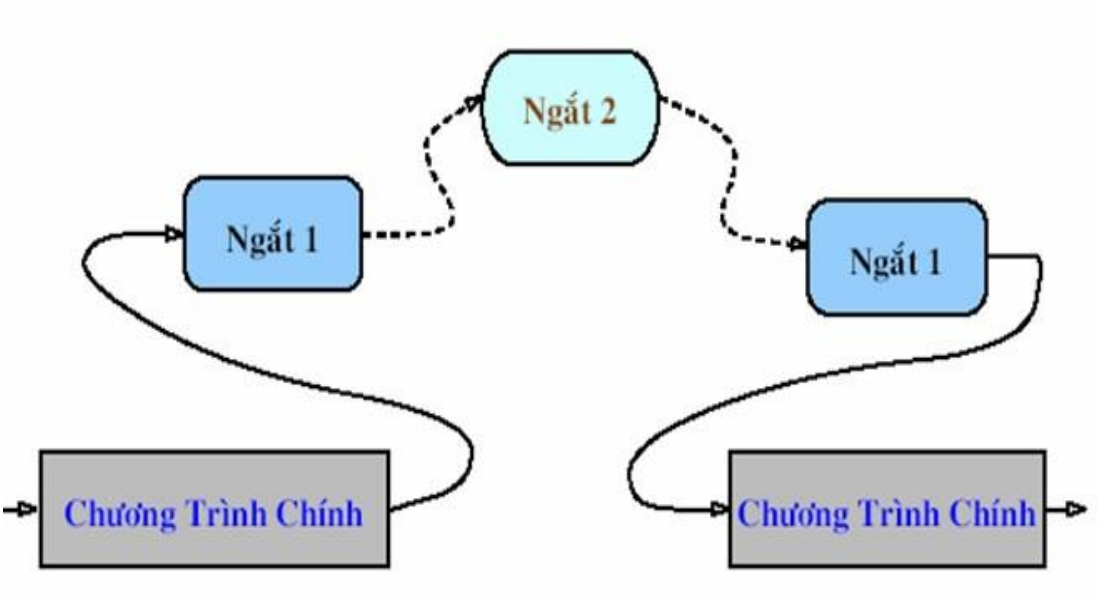
Thứ tự ưu tiên các ngắt là không thể thay đổi và theo qui tắc: "Một véc tơ ngắt có địa chỉ thấp hơn trong bộ nhớ chương trình có mức độ ưu tiên cao hơn". Chẳng hạn ngắt ngoài 0 (INT0) có mức độ ưu tiên cao hơn ngắt ngoài 1 (INT1).

Để cho phép một ngắt người dùng cần cho phép ngắt toàn cục (set bit 1 trong thanh SREG) và các bit điều khiển 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 thì nó sẽ không được phục vụ, do đó để cho phép các ngắt trong một ISR (interrupt service routine) khác đang thực thi, thì trong chương trình ISR phải có lệnh SEI đề set lại bit I trong SREG.

1.3.5. 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 1.12: 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.

1.3.6. Các ngắt ngoài

Trên chip ATmega8 có 2 ngắt ngoài có tên là INT0 và INT1 tương ứng 2 chân số 4 (PD2) và số 5 (PD3) . Có 3 thanh ghi liên quan đến ngắt ngoài đó là MCUCR, GICR và GIFR.

- Thanh ghi điều khiển MCU – MCUCR (MCU Control Register) là thanh ghi xác lập chế độ ngắt cho ngắt ngoài. Thanh ghi MCUCR chứa các bits cho phép chúng ta chọn 1 trong 4 MODE trên cho các ngắt ngoài.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	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	

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.

Bảng 1.5: Bảng điều khiển kiểu ngắt.

ISC11	ISC10	Mô tả
0	0	Mức thấp của chân INTx tạo ra 1 yêu cầu ngắt – ngắt mức thấp
0	1	Bất kỳ sự thay đổi nào của chân INTz tạo ra 1 yêu cầu ngắt
1	0	Cạnh xuống trên chân INTx tạo ra 1 yêu cầu ngắt – ngắt cạnh xuống
1	1	Cạnh lên trên chân INT1 tạo ra 1 yêu cầu ngắt – ngắt cạnh lên

- Thanh ghi điều khiển ngắt chung – GICR (General Interrupt Control Register) (trên các chip AVR cũ, như các chip AT90Sxxxx, thanh ghi này có tên là thanh ghi mặt nạ ngắt thông thường GIMSK). 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:

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	–	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – INT1 gọi là bit cho phép ngắt 1(Interrupt Enable), set bit này bằng 1 nghĩa là 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:

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	–	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

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). Một chú ý khác là vì các chân PD2, PD3 là các chân ngắt nên ta phải set các chân này là Input (set thanh ghi DDRD).

Hình 1.13: Thiết lập ngắt ngoài.

1.4. CÁC BỘ PHẬN NGOẠI VI KHÁC

Ngoài các bộ phận ngoại vi đã được giới thiệu ở trên như: Bộ định thời, các cổng vào ra, EEPROM… Vi điều khiển AT mega8 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, I2C)…. Việc tích hợp các bộ ngoại vi này vào trong chip giúp cho các thiết kế trở nên thuận tiên hơn, kích thước bo mạch cũng gọn gàng hơn.

1.4.1. Giới thiệu về bộ biến đổi ADC của ATmega8

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). Bô ADC của Atmega8 có độ phân giải 10 bit, sai số tuyệt đối là 2LSB, dải tín hiệu ngõ vào từ 0V-Vcc, 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)… Có rất nhiều phương pháp chuyển đổi ADC. Tuy nhiên, phương pháp chuyển đổi cơ bản và phổ biến nhất là phương pháp chuyển đổi trực tiếp (direct converting) hoặc flash ADC. Các bộ chuyển đổi ADC theo phương pháp này được cấu thành từ một dãy các bộ so sánh (như opamp), các bộ so sánh được mắc song song và được kết nối trực tiếp với tín hiệu analog cần chuyển đổi. Một điện áp tham chiếu (reference) và một mạch chia áp được sử dụng để tạo ra các mức điện áp so sánh khác nhau cho mỗi bộ so sánh.

Độ phân giải (Resolution): Độ phân giải được dùng để chỉ số bit cần thiết để chứa hết các mức giá trị digital ngõ ra. Trong trường hợp có 8 mức giá trị ngõ ra, chúng ta cần 3 bit nhị phân để mã hóa hết các giá trị này, vì thế mạch chuyển đổi ADC với 7 bộ so sánh sẽ có độ phân giải là 3 bit. Một cách tổng quát, nếu một mạch chuyển đổi ADC có độ phân giải n bit thì sẽ có 2^n mức giá trị có thể có ở ngõ ra digital. Để tạo ra một mạch chuyển đổi flash ADC có độ phân giải n bit, chúng ta cần đến $2^n - 1$ bộ so sánh, giá trị này rất lớn khi thiết kế bộ chuyển đổi ADC có độ phân giải cao, vì thế các bộ chuyển đổi flash ADC thường có độ phân giải ít hơn 8 bit. Độ phân giải liên quan mật thiết đến chất lượng chuyển đổi ADC, việc lựa chọn độ phân giải phải phù hợp với độ chính xác yêu cầu và khả năng xử lý của bô điều khiển.

Điện áp tham chiếu (reference voltage): Điện áp tham chiếu thường là giá trị điện áp lớn nhất mà bộ ADC có thể chuyển đổi. Trong các bộ ADC,

Vref thường là thông số được đặt bởi người dùng, nó là điện áp lớn nhất mà thiết bị có thể chuyển đổi.

Thanh ghi trong bộ chuyển đổi ADC trên AVR:

Có 4 thanh ghi trong bộ ADC trên AVR trong đó có 2 thanh ghi data chứa dữ liệu sau khi chuyển đổi, 2 thanh ghi điều khiển và chứa trạng thái của ADC:

- **ADMUX (ADC Multiplexer Selection Register)**: là 1 thanh ghi 8 bit điều khiển việc chọn điện áp tham chiếu, kênh và chế độ hoạt động của ADC.

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

- Bit 7:6 - REFS1:0 (Reference Selection Bits): là các bit chọn điện áp tham chiếu cho ADC, 1 trong 3 nguồn điện áp tham chiếu có thể được chọn là: điện áp ngoài từ chân VREF, điện áp tham chiếu nội 2.56V hoặc điện áp AVCC.

Bảng 1.6: Chọn điện áp tham chiếu.

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

- Bit 5- ADLAR (ADC Left Adjust Result): là bit cho phép hiệu chỉnh trái kết quả chuyển đổi.
- Bits 4:0 - MUX4:0 (Analog Channel and Gain Selection Bits): là 5 bit cho phép chọn kênh, chế độ và cả hệ số khuyếch đại cho ADC. Do bộ ADC trên AVR có nhiều kênh và cho phép thực hiện chuyển đổi ADC kiểu so sánh

(so sánh điện áp giữa 2 chân analog) nên trước khi thực hiện chuyển đổi, chúng ta cần set các bit MUX để chọn kênh và chế độ cần sử dụng.

- **ADCSRA (ADC Control and Status RegisterA):** là thanh ghi chính điều khiển hoạt động và chứa trạng thái của module ADC.

7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

ADCL và ADCH (ADC Data Register)**:** 2 thanh ghi chứa giá trị của quá trình chuyển đổi. Do module ADC trên AVR có độ phân giải tối đa 10 bits nên cần 2 thanh ghi để chứa giá trị chuyển đổi. Tuy nhiên tổng số bít của 2 thanh ghi 8 bit là 16, con số này nhiều hơn 10 bit của kết quả chuyển đổi, vì thế chúng ta được phép chọn cách ghi 10 bit kết quả vào 2 thanh ghi này. Bit ADLAR trong thanh ghi ADMUX quy định cách mà kết quả được ghi vào.

ADLAR=0:

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	

ADLAR=1:

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	

Thông thường, 2 thanh ghi data được sắp xếp theo định dạng ADLAR=0, ADCL chứa 8 bit thấp và 2 bit thấp của ADCH chứa 2 bit cao nhất của giá trị thu được.

- **SFIOR(Special FunctionIO Register C)**: thanh ghi chức năng đặc biệt, 3 bit cao trong thanh ghi này quy định nguồn kích ADC nếu chế độ Auto Trigger được sử dụng. Đó là các bit ADTS2:0 (Auto Trigger Source 2:0). Các loại nguồn kích được trình bày trong bảng dưới.

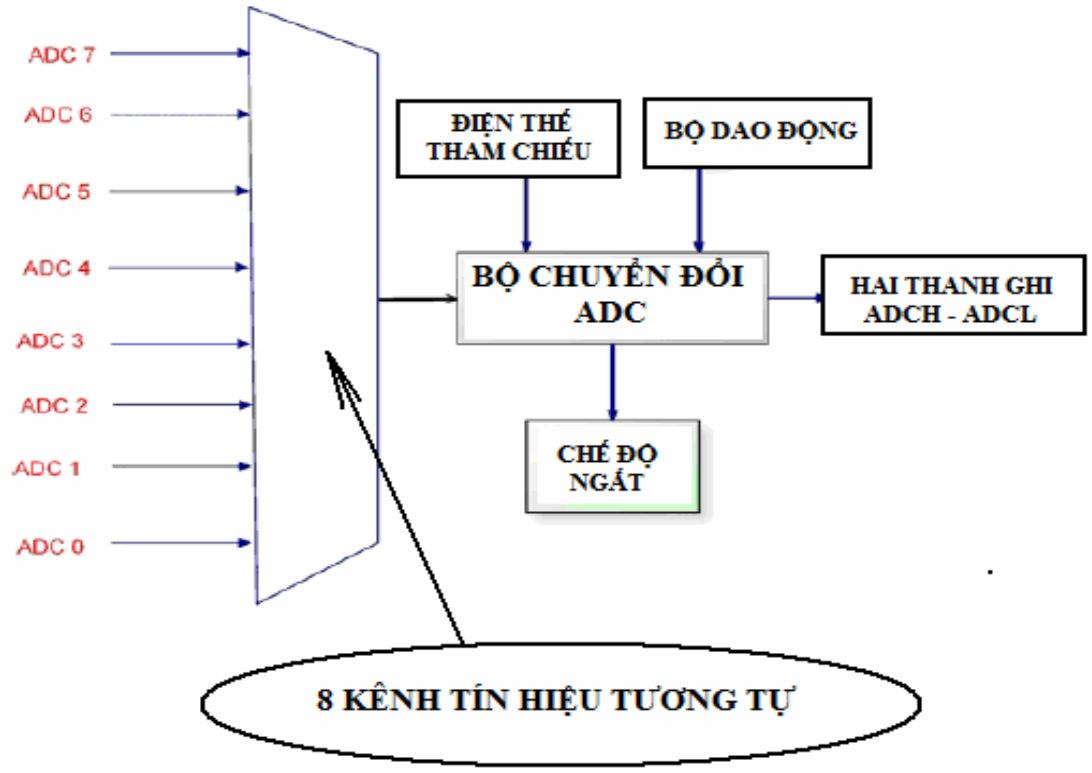
7	6	5	4	3	2	1	0	
ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10	SFIOR
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	

Nguồn kích ADC trong chế độ Auto Trigger:

Bảng 1.7: Nguồn kích ADC trong chế độ Auto Trigger.

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Sơ đồ khối đơn giản của một bộ ADC được thể hiện như sau:



Hình 1.14: Sơ đồ khối đơn giản bộ ADC.

Nguyên tắc hoạt động của khối ADC: Tín hiệu tương tự đưa vào các ngõ ADC 0 – 7 được lấy mẫu và biến đổi thành tín hiệu số tương ứng. Tín hiệu số được lưu hành trong hai thanh ghi ACDH và ADCL. Một ngắt có thể được tạo ra khi hoàn thành một chu trình biến đổi ADC. Bộ ADC của Atmega8 phức tạp hơn nhiều, tuy nhiên cơ sở vẫn dựa vào nguyên tắc trên.

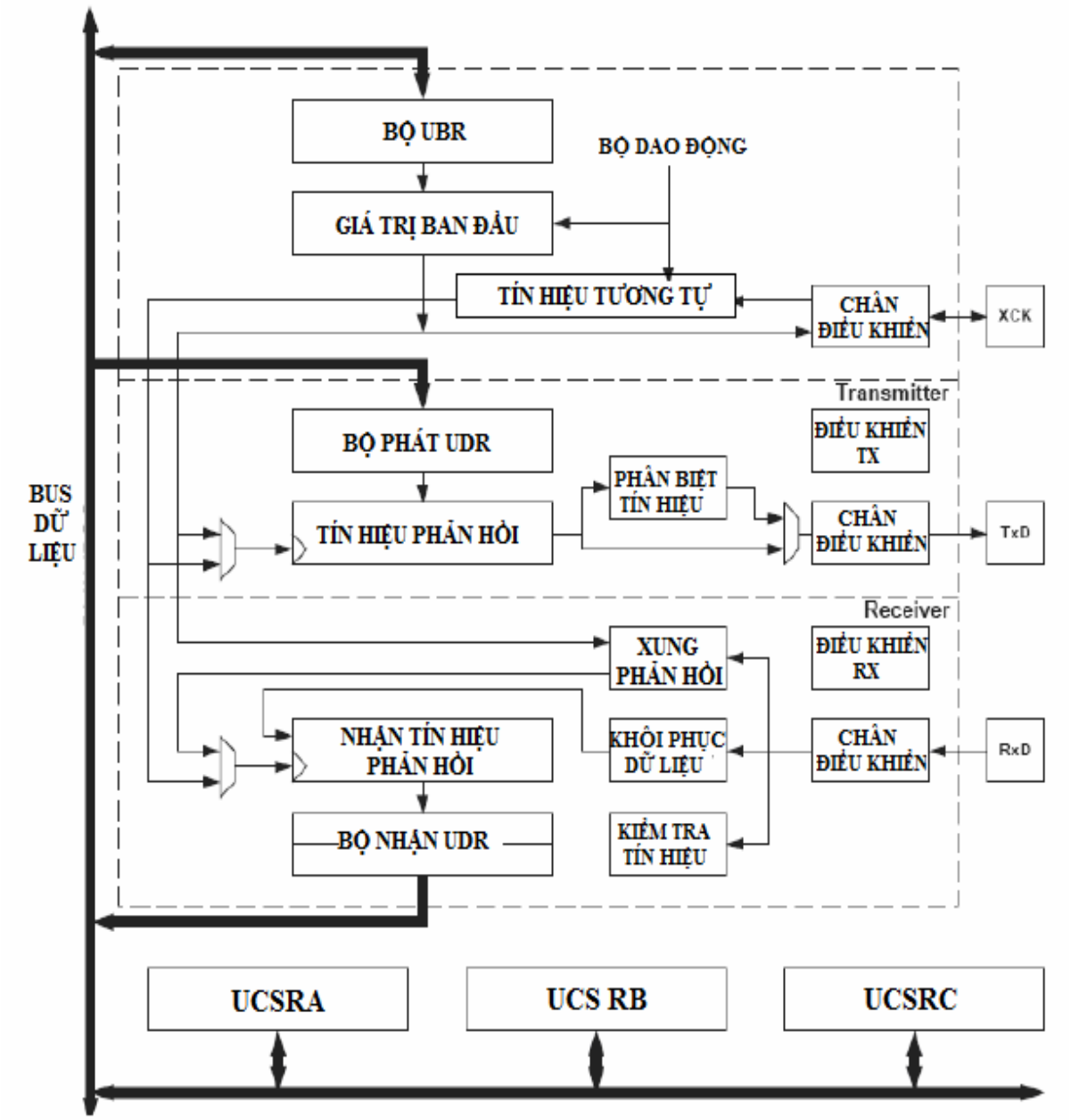
1.4.2. Giới thiệu bộ truyền dữ liệu nối tiếp USART của ATmega 8

USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter): Bộ điều khiển đồng bộ và bất đồng bộ, 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 truyền dữ liệu ra hai chế độ cơ bản là: Chế độ nhận đồng bộ (Synchronous) và chế độ truyền nhận bất đồng bộ (Asynchronous). Ngoài ra, nếu gốc 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 một xung đồng hồ (clock). Do đó, hoạt động truyền và nhận giữ liệu ra đồng thời.

Truyền bất đồng bộ: Là kiểu truyền dữ liệu trong đó mỗi bộ truyền và bộ nhận có bộ dao động xung clock riêng, tốc độ xung clock ở hai khối này có thể khác nhau, nhưng thường không quá 10%. Do đó 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.

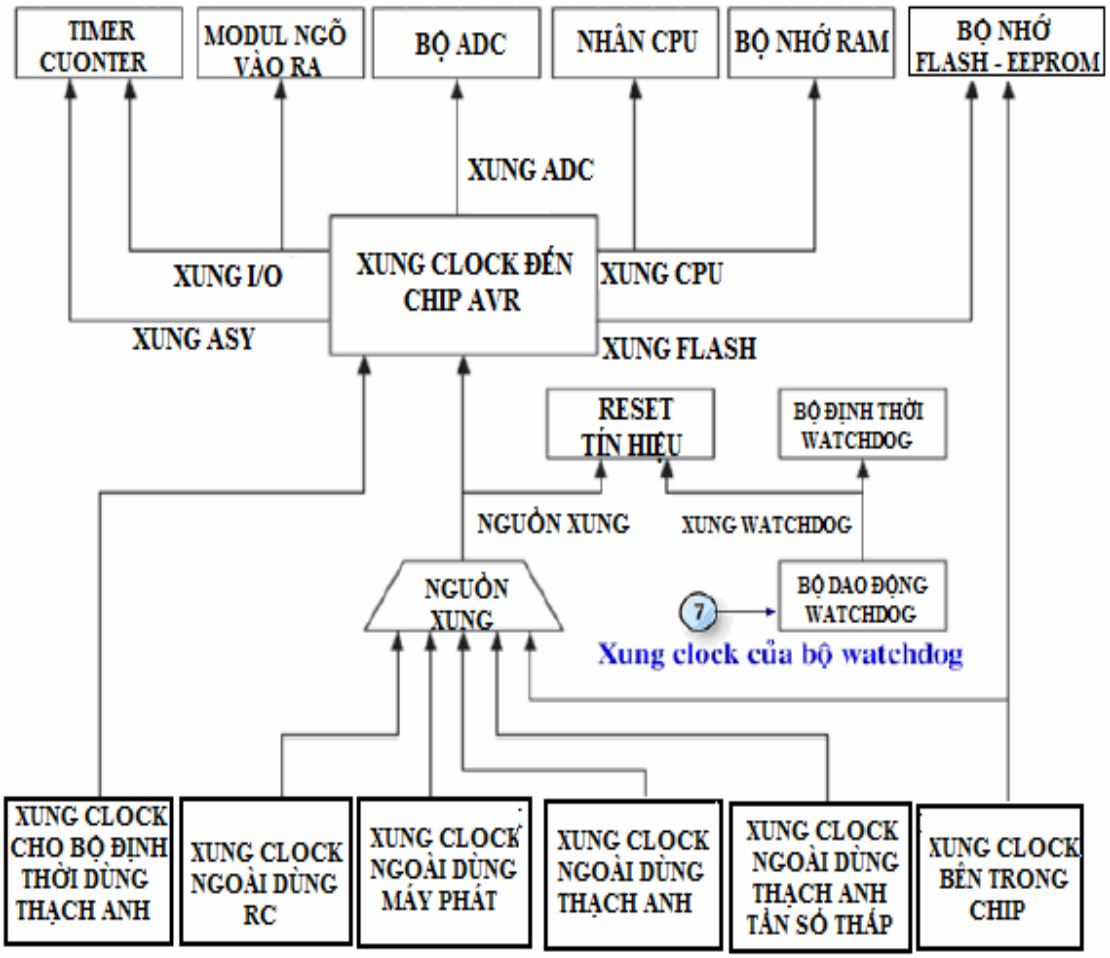
ATmega8 có hai bộ USART là USART0 và USART1. Hai bộ USART này là độc lập nhau, điều này có nghĩa là hai khối USART0 và USART1 có thể hoạt động cùng một lúc. Bên dưới là sơ đồ khối đơn giản của khối USART.



Hình 1.15: Sơ đồ khối bộ USART.

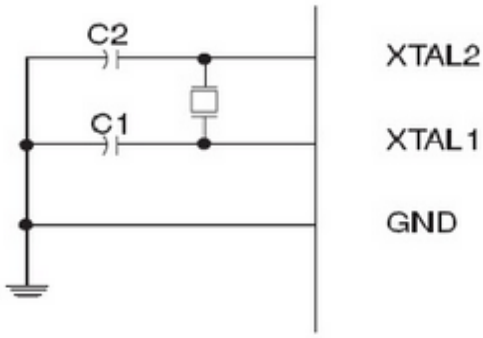
1.4.3. Hệ thống xung CLOCK:

Hệ thống xung clock Atmega8 được chia thành nhiều khối khác nhau, mỗi khối (modul) sẽ cung cấp xung clock cho các khối ngoại vi ứng dụng tương ứng.



Hình 1.16: Sơ đồ hệ thống xung clock cho ATmega8.

1.4.4. Bộ tạo dao động thạch anh:



Hình 1.17: Ghép nối bộ dao động thạch anh.

Các chân XTAL1 và XTAL2 (chân 23 và chân 24) lần lượt là ngõ vào và ngõ ra của bộ khuếch đại đảo được tích hợp sẵn trong chip. Giá trị của tụ C1 và C2 phải bằng nhau và thường có giá trị vào khoảng 12pF – 22pF. Với ATmega8 thì tần số xung clock hệ thống tối đa 16MHz và để đạt được tần số tối đa này bit cầu chì CKOPT phải được lập trình (= 0). Nếu bit CKOPT không được lập trình (= 1) thì tần số tối đa chỉ là 8MHz.

CHƯƠNG 2 :
NGÔN NGỮ LẬP TRÌNH C VÀ PHẦN MỀM LẬP TRÌNH CODEVISIONAVR

2.1. NGÔN NGỮ LẬP TRÌNH C CHO AVR

2.1.1. Khái niệm.

2.1.1.1. Các chú thích và tiền xử lí (PreProcessor)

- **Các chú thích:**

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

- **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 <mega8.h>

Cho phép sử dụng các thanh ghi của Atmega8. Tức báo cho trình biên dịch biết chúng ta đang sử dụng vi điều khiển Atmega8. 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.

- **Các kiểu dữ liệu (Data Types):**

Ngoài các kiểu dữ liệu của C, CodeVisionAVR còn có kiểu dữ liệu **bit** là kiểu dữ liệu 1 bit, nên giải 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.

Bảng 2.1: Các kiểu khai báo dữ liệu

Kiểu dữ liêu	**Kích cỡ (bit)**	**Giới han**
Bit	1	0,1
Char	8	-128 đến 127
Unsigned char	8	0 đến 225
Signed char	8	-128 đện127
Int	16	-32768 đến 32767
Short int	16	-32768 đến 32767
Unsigned int	16	0 đến 65535
Sunged int	16	-32768 đến 32767
Long int	32	-2147483648 đến 2147483647
Unsign long int	32	0 đến 4294967295
Signed long int	32	-2147483648 đến 2147483647
Float	32	± 1.175e38 đến ±3.402e38
Double	32	± 1.175e38 đến ±3.402e38

- **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.

- **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 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.

- 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 khóa. 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 Atmega8 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,…. Do đó, khi chạy vào hàm main, các biến toàn cục, mà thực chất là các ô nhớ, đã có giá trị khởi tạo sẵn. Với các biến cục bộ, trình hợp ngữ không khởi tạo trước giá trị.

- **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

2.1.1.2. 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 từ (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.

2.1.1.3. 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 mà cũng không có tham số trả về.
- Giá trị trả về của hàm được lưu trong các thanh ghi R30, R31, R22, R23.

- **Con trỏ (Pointer):**

Những biến lưu trữ địa chỉ của một biến khác 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 ở 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 đựng 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 trỏ là một con trỏ nguyên
con_tro = &number ;// biến con_tro tới biến number
```

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

2.1.1.4. 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**. (hoặc file header cho từng chip cụ thể, **mega8.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
```

2.1.2. Tóm tắt cấu trúc điều khiển

2.1.2.1. 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
```

2.1.2.2. Vòng lặp While và do – While

while *(expression) starement ; // (1)*

do *statement* ***while*** *(condition); // (2)*

Chức năng của (1) đơn giản chỉ là lập lại statement khi điều kiện expression còn thỏa mãn.

Chức năng của (2) hoàn toàn giống vòng lặp white chỉ trừ 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 thỏa mãn.

Ví dụ:

```
int i ;
while  (I < 128)
{
PORD = I;
i = i*2 ;
}
```

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

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

2.1.2.3. 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 đặt biệt lặp lại một hành động với một số lần nhất định.

• Initialization được thực hiện. Nói chung nó đặt một giá trị ban đầu cho biến điều khiển. Lệnh này được thực hiện chỉ một lần.

• Condition được kiểm tra, nếu nó là đúng vòng lặp tiếp tục còn nếu không vòng lặp kết thúc và statement được bỏ qua.

• Statement được thực hiện. Nó có thể có một lệnh đơn hoặc là một khối lệnh được bao trong một cặp ngoặc nhọn.

• Cuối cùng, increase được thực hiện để tăng biến điều khiển và vòng lặp quay trở lại kiểm tra.

Ví dụ:

```
For (int i = 1; I <= 128; i = i*2)
{
PORD = I ;
}
```

Cấu trúc sau sẽ lập vô hạn giống như cấu trúc while (1)

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

2.1.2.4. 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 thỏa mãn. Lệnh này có thể được dùng đề 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--)
{
PORD = n ;
if (n== 7)
{
break;
}
}
```

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

Nếu sửa lại đoạn code trên như sau:

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

Thì PORTD = 10, 9, 8.

- **Lệnh nhảy *goto***

Lệnh goto cho phép nhảy vô điều kiện với 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.

2.1.2.5. 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 khóa ***break***, sau đó nhẩy đến phần cuối của cấu trúc lựa chọn ***switch***. Còn nếu không, ***switch*** sẽ kiểm tra xem biểu thức có bằng constant 2 hay không. Nếu đúng nó sẽ thực hiện block of instructions 2 cho đến khi tìm thấy từ khóa ***break***. Cuối cùng, nếu giá trị biểu thức không bằng bất kỳ hằng nào được chỉ định ở trên thì 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;
}
```

2.1.3. Chẳng hợp ngữ trong 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”)
```

2.1.4. Tổ chức bộ nhớ SRAM

Trình biên dịch phân chia và quản lí bộ nhớ SRAM của AVR như sau: 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**

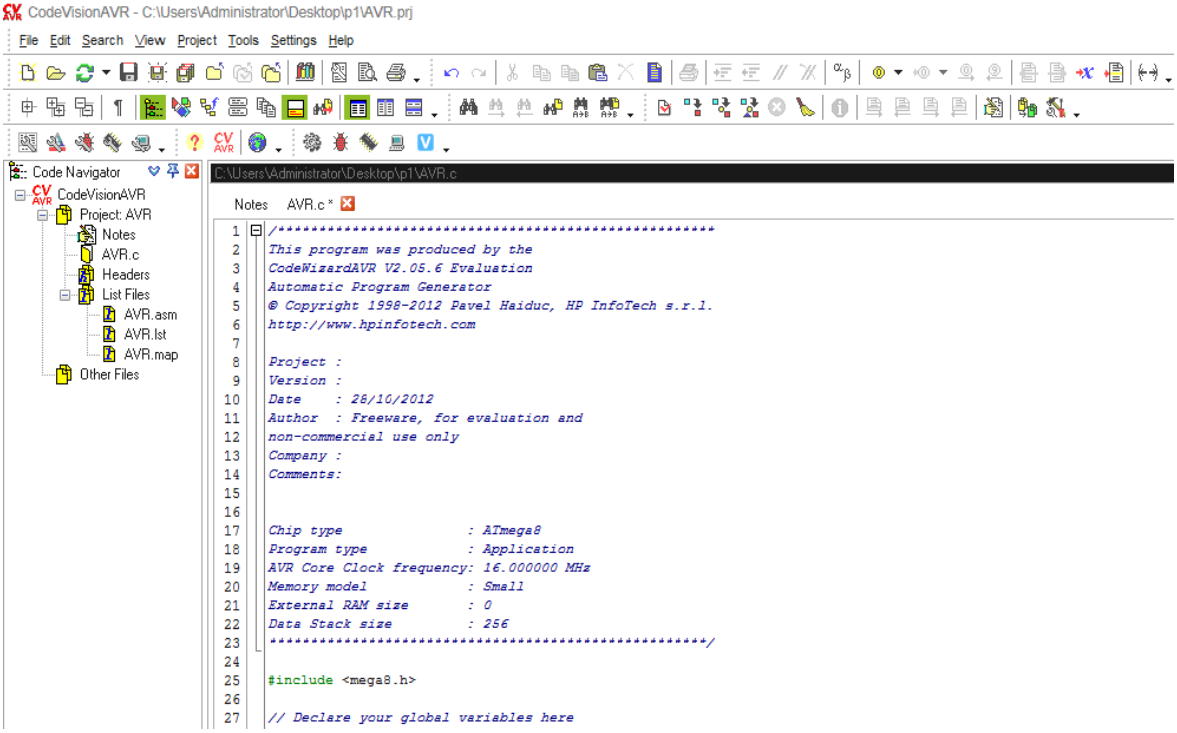
2.2. PHẦN MỀM LẬP TRÌNH CHO VI ĐIỀU KHIỂN AVR

Giới thiệu phần mềm CodeVisionAVR

Hình 2.1: Chương trình lập trình ATmega8.

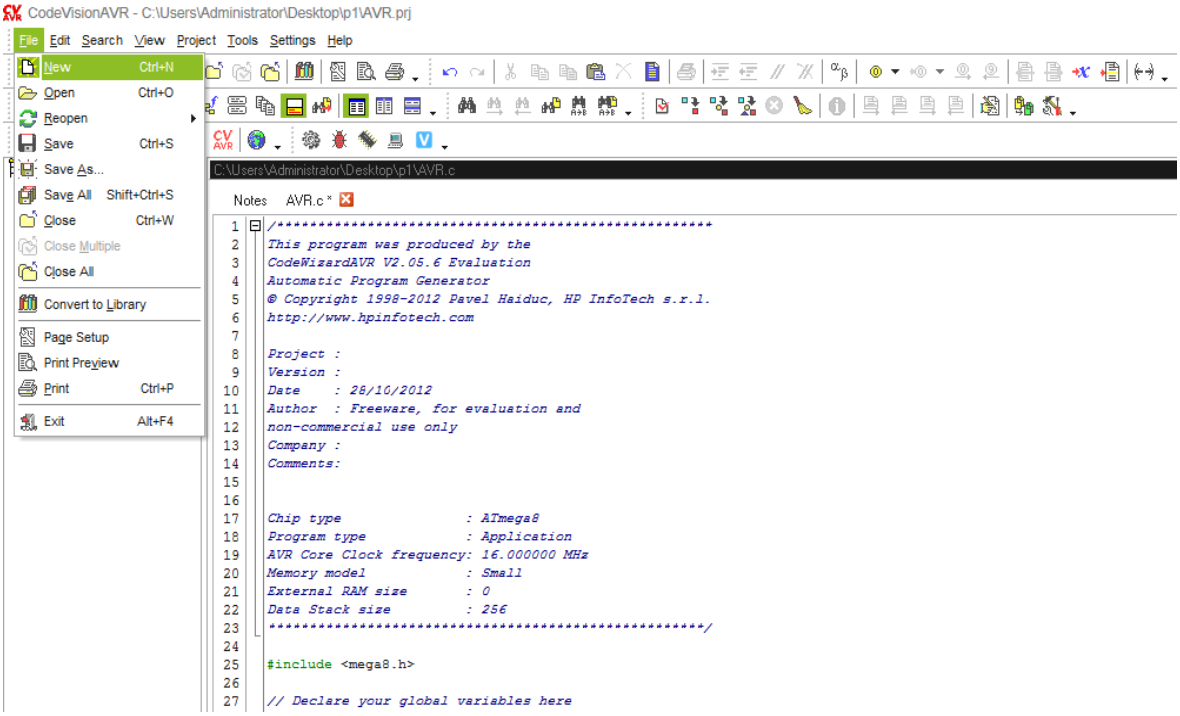
CodeVisionAVR là phần mềm chuyên dụng để lập trình chip AVR, ngôn ngữ lập trình C hay Asm và một số ngôn ngữ thông dụng khác đều có thể chạy trên nền CodeVisionAVR.

Giới thiệu sơ lược về phần mềm Codevision



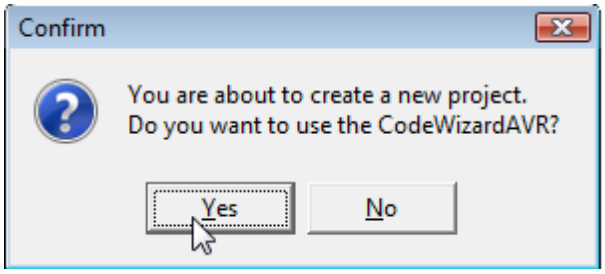
Hình 2.2: Giao diện lập trình phần mềm CodeVisionAVR.

Tạo một chương trình mới **FILE** >> **NEW**



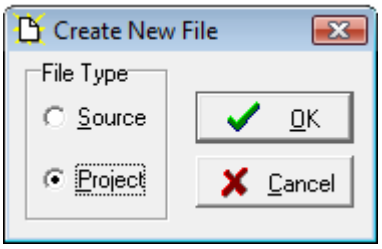
Hình 2.3: Cách tạo một project trên CodeVision AVR.

Nhấn **OK** để tiếp tục các bước tạo project cho phần mềm:



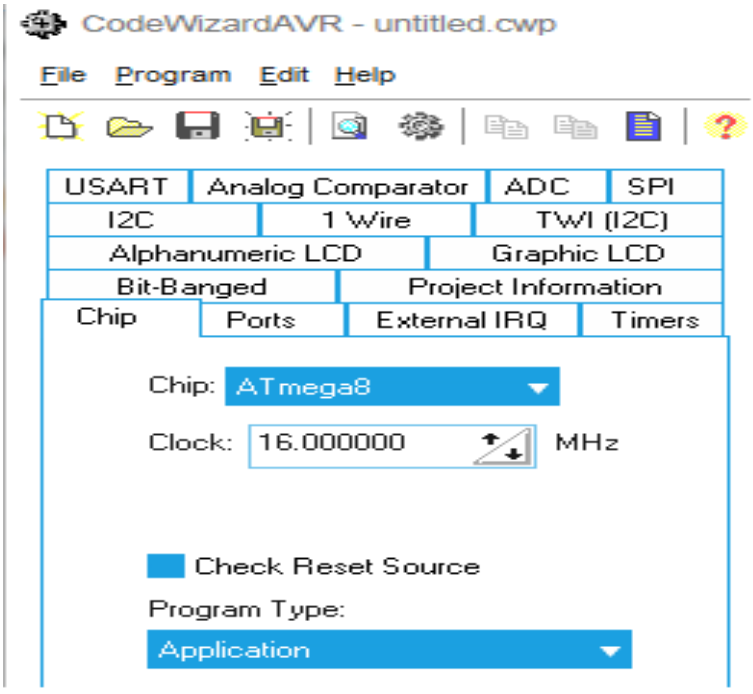
Hình 2.4: Các bước thực hiện.

Check vào nút **Project** >>> **OK**

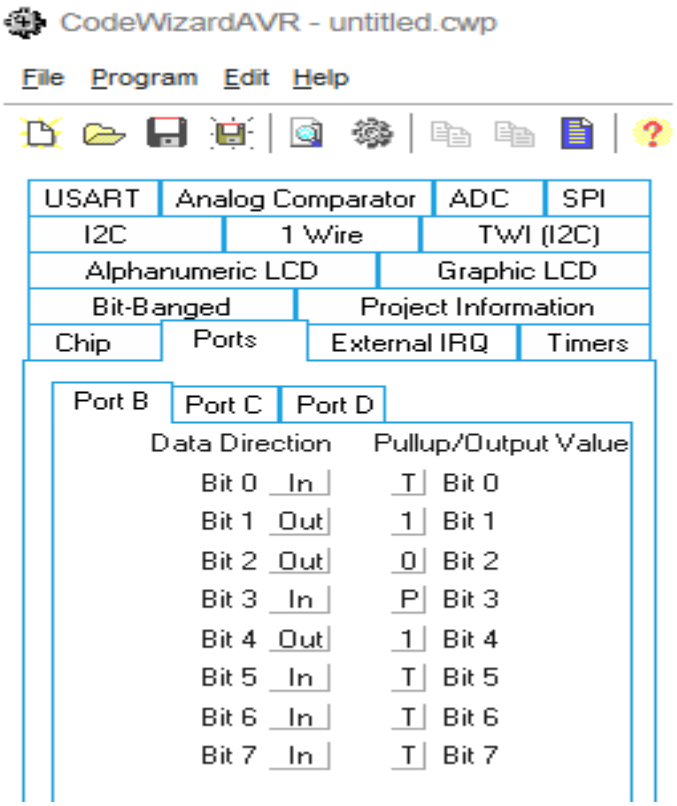


Hình 2.5: Các bước thực hiện.

Chọn thẻ Chip để chọn loại AVR cần lập trình và tốc độ xung Clock



Hình 2.6: Cách chọn loại AVR



Hình 2.7: Chọn cổng vào ra

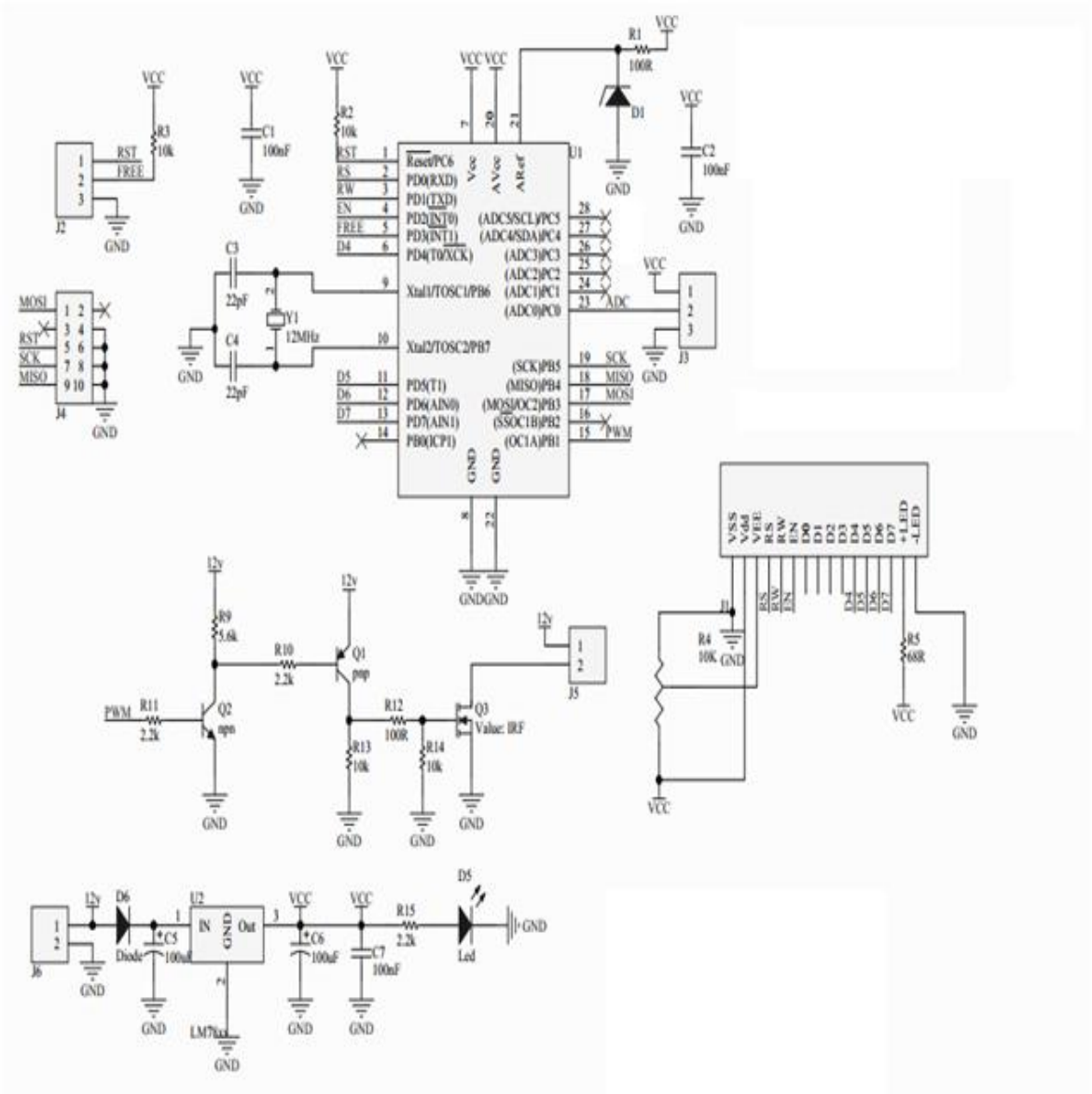
Sau các bước này là ta có thể tiến hành lập trình trên CodeVision AVR.

CHƯƠNG 3:

THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG PHUN SƯƠNG LÀM MÁT TỰ ĐỘNG

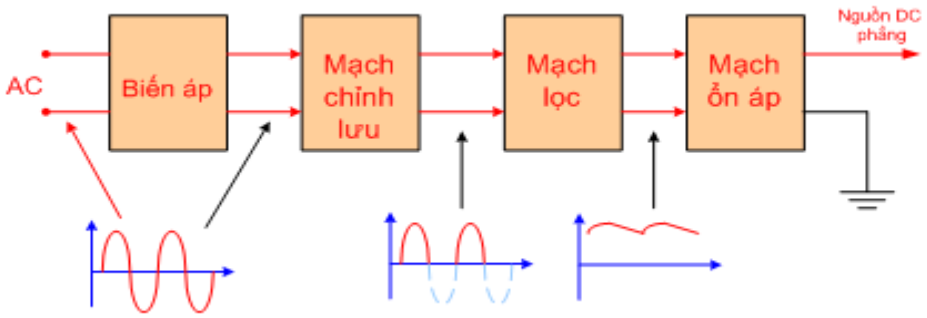
3.1. CHỨC NĂNG CÁC KHỐI VÀ LINH KIỆN SỬ DỤNG TRONG THIẾT KẾ MẠCH

Sơ đồ nguyên lý thiết kế mạch:



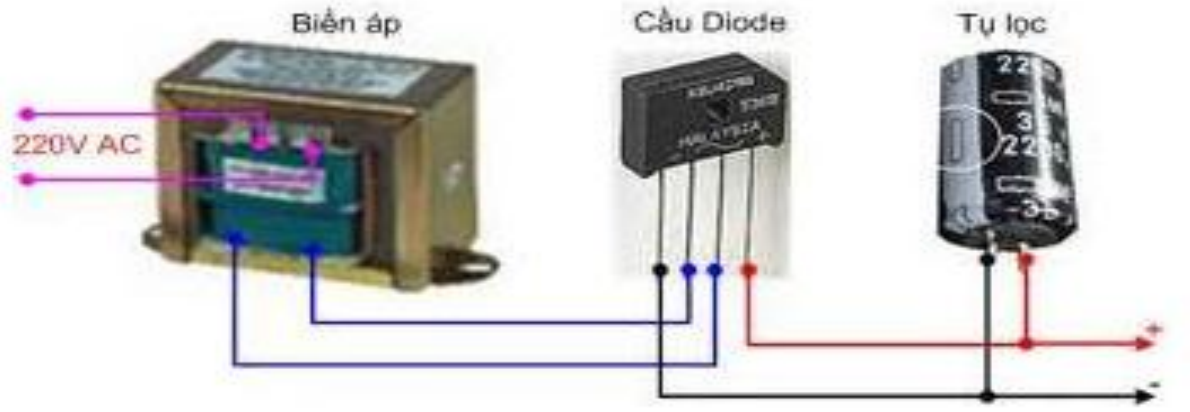
Hình 3.1: Sơ đồ nguyên lý thiết kế mạch.

3.1.1. Khối mạch chỉnh lưu và ổn áp 5V



Hình 3.2: Sơ đồ tổng quát của mạch cấp nguồn.

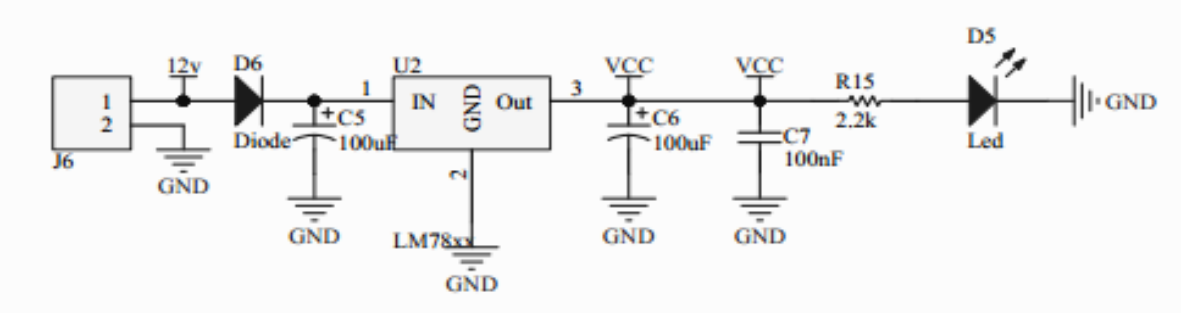
3.1.1.1. Biến áp và mạch chỉnh lưu



Hình 3.3: Mạch chuyển đổi điện áp 12VDC từ nguồn 220VAC ghép nối qua tụ lọc.

Trong đó biến áp nguồn dùng để hạ điện áp xoay chiều từ 220V xuống điện áp 12V qua mạch chỉnh lưu cầu Diode thay đổi điện áp từ dòng xoay chiều thành 1 chiều. Tụ hóa công suất lớn 2200uF dùng làm kho chứa điện có tác dụng lọc gợn xoay chiều sau chỉnh lưu cho nguồn DC phẳng hơn.

3.1.1.2. Mạch nguồn ổn áp 5V



***Hình 3.4*: Mạch nguồn ổn định 5V.**

Thông số chính của mạch:

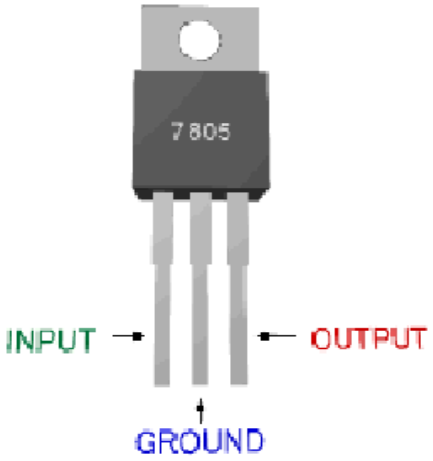
- Điện áp đầu vào từ 12VDC – 40VDC
- Điện áp đầu ra 5V
- Có bảo vệ chống dòng ngược

Các linh kiện chính trong mạch:

- Diode bảo vệ chống dòng ngược
- Tụ điện hóa 100uF - 50V
- Tụ điện không phân cực C104
- Led báo nguồn và điện trở led
- IC ổn áp 7805

- **IC ổn áp 7805:**

7805 gồm có 3 chân kết nối: chân 1 là chân nguồn đầu vào, chân 2 là chân GND, chân 3 là chân lấy điện áp ra.



Hình 3.5: IC ổn áp 7805.

- Chân 1 – 2 là chân điện áp đầu vào: Đây là chân cấp nguồn đầu vào cho 7805 hoạt động. Giải điện áp cho phép đầu vào lớn nhất là 40V.

- Chân 3 là chân điện áp đầu ra : Chân này cho ta lấy điện áp đầu ra ổn định 5V. Đảm bảo đầu ra ổn định luôn nằm trong giải

từ (4,75V đến 5,25V).

Khi cấp nguồn cho 7805 cần phải luôn đảm bảo thông số Vi - Vo > 3V. Tức là điện áp đầu vào phải nằm trong khoảng 8V – 40V. Nếu dưới 8V thì mạch ổn áp không còn tác dụng. Thông thường người ta thường cấp nguồn lớn hơn ít nhất gấp đôi nguồn đầu ra để tránh trường hợp sụt áp đầu vào sinh ra nguồn đầu ra không ổn định trong thời gian ngắn.

- **Thành phần lọc nguồn và lọc nhiễu:**

Các tụ hóa 470uF được dùng để lọc điện áp. Vì đây là điện áp một chiều nhưng chưa được phẳng vẫn còn các gợn nhấp nhô nên các tụ này có tác dụng lọc nguồn cho thành điện áp một chiều phẳng.

- Tụ C5 là tụ lọc nguồn đầu vào cho 7805. Tụ này phải có điện dung đủ lớn để lọc phẳng điện áp đầu vào và điện áp tụ chịu đựng phải lớn hơn điện áp đầu vào.
- Tụ C6 là tụ lọc nguồn đầu ra cho 7805, dùng để lọc nguồn đầu ra cho bằng phẳng.

Trong thành phần một chiều còn có các sóng điều hòa bậc 2, 3…, sóng nhấp nhô có tần số cao, nhiễu bên ngoài. Các sóng này ảnh hưởng đến hoạt động của 7805. Nếu trong mạch tồn tại những thành phần sóng này sẽ làm sai sót khó phát hiện trong mạch làm cho mạch hoạt động không ổn định.

Tụ lọc nhiễu tần số cao C7. Tụ này phải là tụ không phân cực, tụ Ceramic. Tụ này lọc các thành phần trên cho đầu ra đảm bảo cho mạch hoạt động bình thường.

- **Thành phần bảo vệ chống dòng ngược:**

Diode có tác dụng bảo vệ chống dòng ngược và là thành phần cần phải có trong mạch một chiều. Tránh trường hợp lắp ngược nguồn sinh ra hỏng mạch và cháy mạch.

3.1.2. Màn hình Text LCD 16x2

LCD là một thiết bị điện tử dùng để hiển thị, cũng như led 7 đoạn nó dùng để hiển thị dữ liệu nhưng lại có thể hiển thị được cả chữ và số với các hiệu ứng đa dạng và phong phú.

Hình 3.6: Text LCD 16x2.

Sơ đồ chân LCD:

Đối với loại LCD này thường có 16 chân trong đó 14 chân kết nối với bộ vi điều khiển và 2 chân nguồn cho đèn Led nền:

- Các chân 1, 2, 3 là các chân Vss, Vdd, Vee trong đó Vss là chân nối đất, Vee là chân chọn độ tương phản và được mắc qua 1 biến trở 10K một đầu nối với nguồn Vcc, một đầu nối mát với độ tương phản từ 0 – Vdd. Chân Vdd được nối dương nguồn.
- Chân chọn thanh ghi RS (Register Select): Có 2 thanh ghi trong LCD được dùng để chọn thanh ghi như sau:

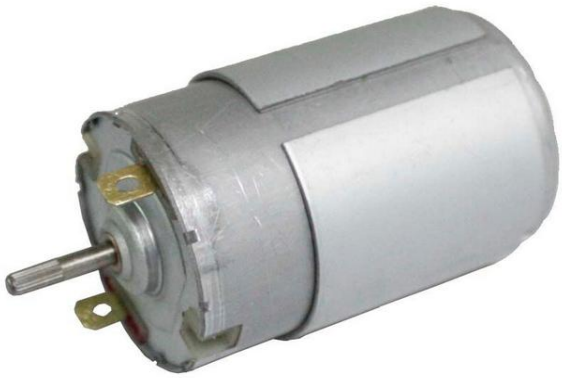
  + Nếu RS = 0 ở chế độ ghi lệnh như xóa màn hình, bật tắt con trỏ,…

  + Nếu RS = 1 ở chế độ ghi dữ liệu như hiển thị ký tự, chữ số lên màn hình.
- Chân Đọc/Ghi (R/W): Đầu vào đọc/ghi cho phép người dùng ghi thông tin lên LCD khi R/W = 0 hoặc đọc thông tin LCD khi R/W = 1.
- Chân cho phép E (Enable): Chân cho phép E được sử dụng bởi LCD để chốt dữ liệu của nó. Khi dữ liệu được đến chân sữ liệu thì cần có 1 xung từ mức cao xuống mức thấp ở chân này để LCD chốt dữ liệu, xung này phải có độ rộng tối thiểu 450ns.
- Chân D0 – D7: Đây là 8 chân dữ liệu 8 bit, được dùng để gửi thông tin lên LCD hoặc đọc nột dung của các thanh ghi trong LCD để hiện thị các chữ cái và các con số chúng ta gửi các mã ASCII của các chữ cái và các con số tương ứng đến các chân này khi bật RS = 1.

- Chân Led nền 15 và 16 là các chân A (Anode) và K (Cathode). Trong đó chân A được nối qua điện trở Led và chân K được nối mát.

3.1.3. Động cơ DC 12V

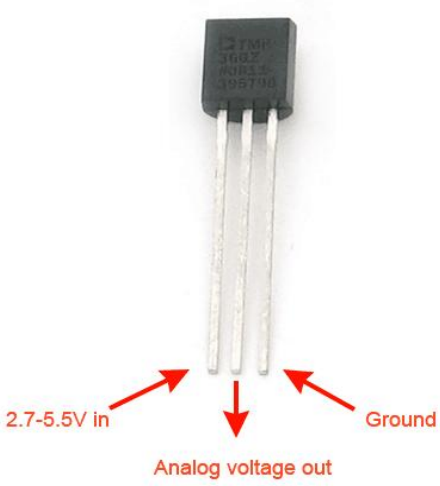
Về phương diện điều khiển tốc độ động cơ điện một chiều có nhiều ưu việt hơn so với các loại động cơ khác, không những nó có khả năng điều chỉnh tốc độ dễ dàng mà cấu trúc mạch lực, mạch điều khiển đơn giản hơn đồng thời lại đạt chất lượng điều chỉnh cao trong dải điều chỉnh tốc độ rộng.



Hình 3.7: Động cơ 12V DC.

3.1.4. Cảm biến nhiệt độ LM335

Gồm có 3 chân chính: 2 chân cấp nguồn và 1 chân xuất tín hiệu Analog. Khi ta cấp điện áp 5V cho LM335 thì nhiệt độ đo được từ cảm biến sẽ chuyển thành điện áp tương ứng tại chân số 2 (Vout). Đây là cảm biến nhiệt độ có khoảng đo từ -40^{o}C – 100^{o}C, có độ chính xác cao và độ nhạy lớn. Tín hiệu ngõ ra tuyến tính với tín hiệu ngõ vào.

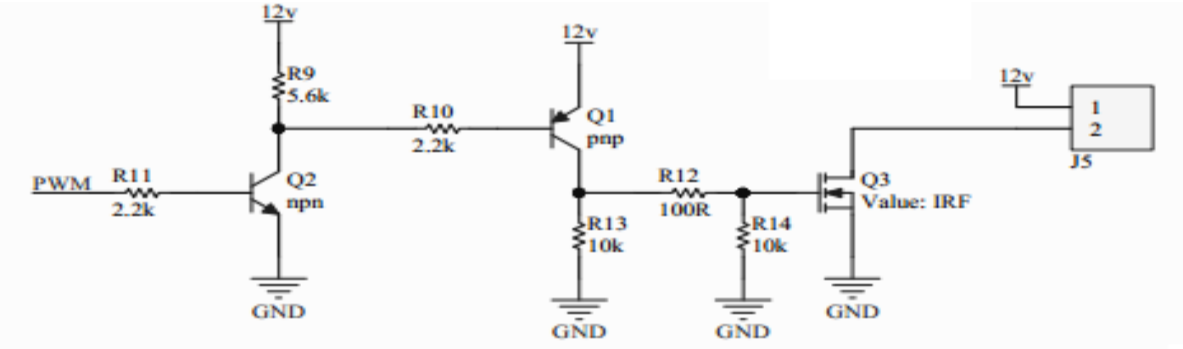


***Hình 3.8*:** LM335.

Các chỉ tiêu kĩ thuật:

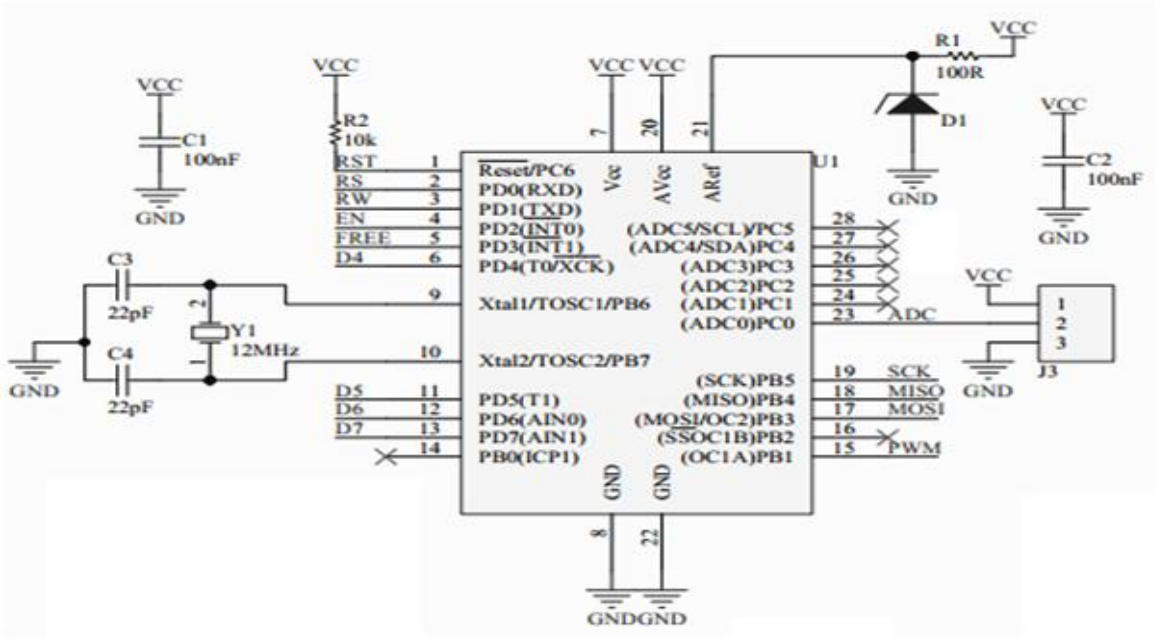
- LM335 có độ biến thiên theo nhiệt độ là 10mV/K^{o}.
- Có độ ổn định cao ở 25^{o}C chỉ có sai số 1% .
- Tiêu tán công suất thấp.
- Dòng làm việc từ 0,4mA – 5mA.
- Trở kháng đầu ra thấp 1 Ohm.

3.1.5. Khối tạo xung PWM



***Hình 3.9*:** Khối tạo xung.

3.1.6. Khối trung tâm điều khiển



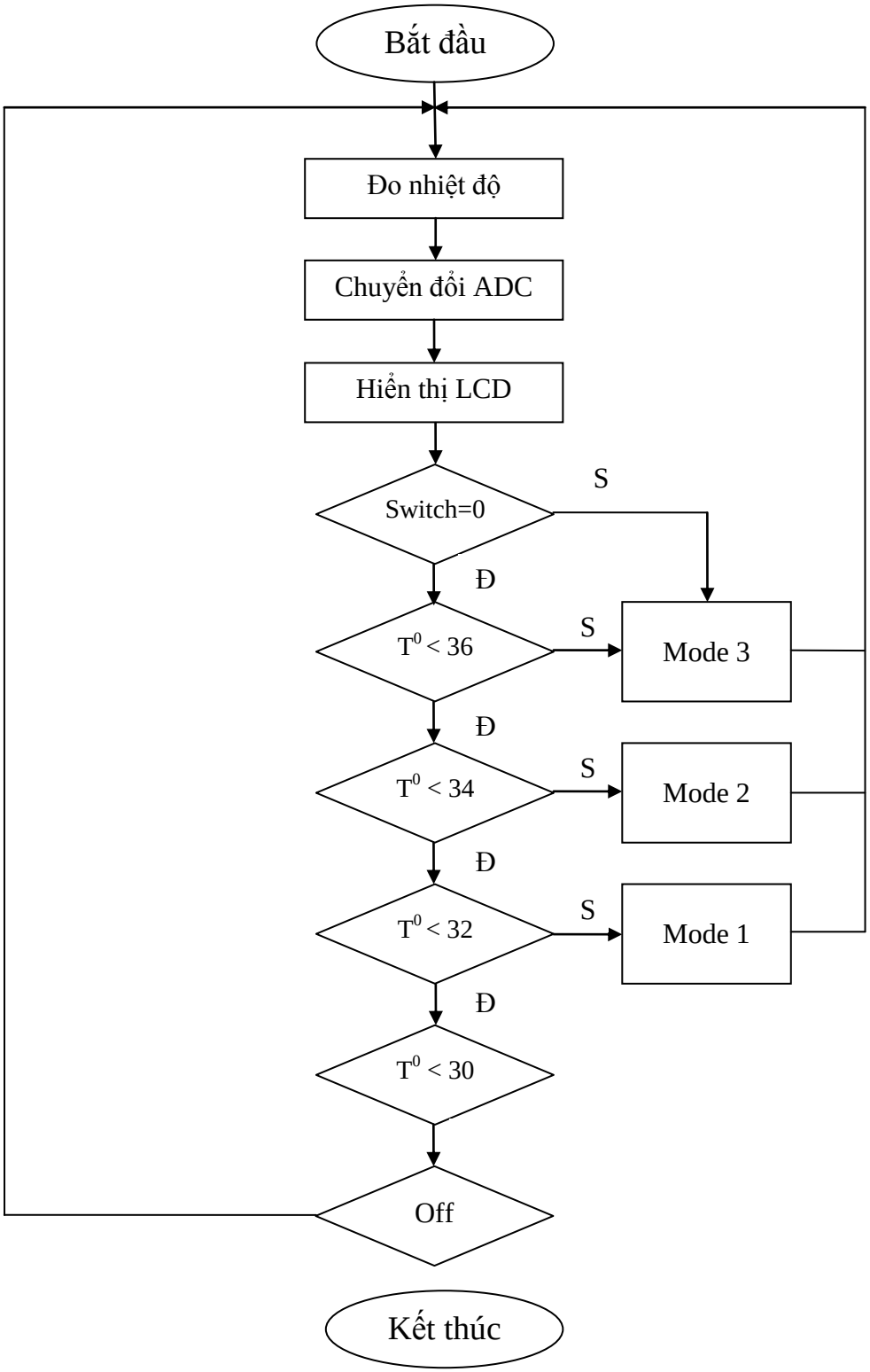
Hình 3.10: Khối điều khiển.

3.2. NGUYÊN LÝ HOẠT ĐỘNG

Khối cảm biến chuyển đổi giá trị nhiệt độ thành giá trị điện áp đưa đến khối chuyển đổi tương tự sang số. Khối chuyển đổi tương tự sang số chuyển đổi giá trị điện áp thành giá trị số dựa vào việc so sánh với giá trị điện áp chuẩn mà khối tạo điện áp chuẩn tạo ra và số bít đầu ra. Giá trị số của nhiệt độ được đưa đến khối xử lý để so sánh với giá trị cài đặt để điều khiển động cơ và chuyển đổi giá trị số của nhiệt độ thành mã BCD để hiển thị. Khối nút nhấn

dùng để điều khiển động cơ hoạt động hết công suất khi nhấn và trả về auto khi nhả.

3.3. LƯU ĐỒ THUẬT TOÁN ĐIỀU KHIỂN



Hình 3.11. Lưu đồ thuật toán điều khiển.

Trong đó:

- Mode 1: Motor = 75% Call duty
- Mode 2: Motor = 85% Call duty
- Mode 3: Motor = 100% Call duty

3.4. CHƯƠNG TRÌNH ĐIỀU KHIỂN BẰNG CODEVISIONAVR C

```
#include <mega8.h>
#define PWM OCR1AL  //  Motor
#define switch PIND.3  //  nút nhấn Switch
unsigned char Data_LM35=0;
bit Mode=0;
#include <delay.h>
// Alphanumeric LCD functions
#include <alcd.h>
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x08;
// Place your code here
if(switch==0) Mode=3;    //  Nếu nhấn nút Switch thì quay động cơ với tốc độ
                              cao nhất.
else    Mode=0;
}
#define FIRST_ADC_INPUT 0
#define LAST_ADC_INPUT 0
unsigned int adc_data[LAST_ADC_INPUT-FIRST_ADC_INPUT+1];
#define ADC_VREF_TYPE 0x00
// ADC interrupt service routine
```

```
// with auto input scanning
interrupt [ADC_INT] void adc_isr(void)
{
static unsigned char input_index=0;
// Read the AD conversion result
adc_data[input_index]=ADCW;
// Select next ADC input
if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT))
  input_index=0;
ADMUX=(FIRST_ADC_INPUT | (ADC_VREF_TYPE &
0xff))+input_index;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
}
// Declare your global variables here
void lcd_put_int(int num)         //Xuất số nguyên ra LCD
{
        int temp;
        unsigned char i = 0, c[5];
        temp = num;
        if (temp != 0) {
                if (temp < 0){
                        lcd_putchar('-');
                        temp = - temp;
                }
                while(temp){
```

```
                c[i++] = temp%10;
                temp /= 10;
            }
            while(i) lcd_putchar(c[--i] + '0');
        }
        else lcd_putchar('0');
}
void main(void)
{
// Declare your local variables here
// Input/Output Ports initialization
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=Out
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=0
State0=T
PORTB=0x00;
DDRB=0x02;
// Port C initialization
// Func6=In Func5=Out Func4=Out Func3=Out Func2=In Func1=In
Func0=In
// State6=T State5=0 State4=0 State3=0 State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x38;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTD=0x00;
DDRD=0x00;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 11.719 kHz
TCCR0=0x05;
TCNT0=0x08;
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 46.875 kHz
// Mode: Ph. correct PWM top=0x00FF
// OC1A output: Non-Inv.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x81;
TCCR1B=0x04;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
```

```
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
MCUCR=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;
// USART initialization
// USART disabled
UCSRB=0x00;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
// ADC initialization
// ADC Clock frequency: 187.500 kHz
```

```
// ADC Voltage Reference: AREF pin
ADMUX=FIRST_ADC_INPUT | (ADC_VREF_TYPE & 0xff);
ADCSRA=0xCE;
// SPI initialization
// SPI disabled
SPCR=0x00;
// TWI initialization
// TWI disabled
TWCR=0x00;
// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTD Bit 0
// RD - PORTD Bit 1
// EN - PORTD Bit 2
// D4 - PORTD Bit 4
// D5 - PORTD Bit 5
// D6 - PORTD Bit 6
// D7 - PORTD Bit 7
// Characters/line: 16
lcd_init(16);
// Global enable interrupts
#asm("sei")
lcd_gotoxy(2,0);
lcd_putsf("DO AN MON HOC");
delay_ms(1000);
lcd_clear();
lcd_gotoxy(2,0);
```

```
lcd_putsf("HE THONG PHUN");
lcd_gotoxy(2,1);
lcd_putsf("SUONG LAM MAT");
delay_ms(1000);
lcd_clear();
Led_1=0, Led_2=1, Led_3=1;
while (1)
    {
    // Place your code here
    //Data_LM35=(adc_data[0]*4.89)/10;     //  Quy đổi ra nhiệt độ
    Data_LM35=((adc_data[0]-558.58)/2.048);//5v
    delay_ms(100);
    lcd_clear();  //    Xóa  LCD
    lcd_putsf("NHIET DO: ");
    lcd_put_int(Data_LM35);
    lcd_gotoxy(0,1);
    if(Mode==1)   PWM=250, Led_1=0, Led_2=1, Led_3=1;    // Chạy pull
tốc độ
    else if(Mode==0 && Data_LM35>=31 && Data_LM35<33)
PWM=191, Led_1=0, Led_2=1, Led_3=1, lcd_gotoxy(0,1),
lcd_putsf("MODE: 75%");        //  Nếu ở dải nhiệt độ 1
    else if(Mode==0 && Data_LM35>=33 && Data_LM35<36)
PWM=216, Led_1=1, Led_2=0, Led_3=1, lcd_gotoxy(0,1),
lcd_putsf("MODE: 85%");        // Nếu ở dải nhiệt độ 2
    else if(Mode==0 && Data_LM35<31)
 PWM=0, Led_1=1, Led_2=1, Led_3=1, lcd_gotoxy(0,1),
lcd_putsf("MODE: STOP");      // Nếu ở dải nhiệt độ 0
    else if(Mode==0 && Data_LM35>=36)
```

```
PWM=255, Led_1=0, Led_2=0, Led_3=0, lcd_gotoxy(0,1),
lcd_putsf("MODE: 100%");        // Nếu ở dải nhiệt độ 3
      else if(Mode==1)
   PWM=255, lcd_putsf("MODE: 100%"), lcd_putchar(4+48);
      }
}
```

KẾT LUẬN

Sau 12 tuần làm đồ án tốt nghiệp, đến nay đồ án của em đã hoàn thành với đề tài "**Thiết kế, xây dựng hệ thống phun sương làm mát tự động"** do Ths.Nguyễn Trọng Thắng và Ks.Ngô Quang Vĩ trực tiếp hướng dẫn.

Trong đồ án này em đã giải quyết được những vấn đề sau:

- Mạch sau khi thi công chạy đúng như yêu cầu và ổn định nhưng sai số nhiệt độ vẫn còn lớn ($\pm 1^0$C). Để khắc phục ta cần thực hiện lấy giá trị trung bình của khoảng 5 đến 10 lần đo thì giá trị dùng để so sánh và hiển thị sẽ chính xác hơn và ít biến động.
- Giải quyết được vấn đề tăng giảm tốc độ động cơ bằng việc băm xung điều khiển theo nhiệt độ và hiển thị các giá trị đo được trên LCD.

Hướng phát triển đề tài:

- Ngoài các chức năng sẵn có, ta có thể phát triển thêm các chức năng khác như: Bộ hẹn giờ tắt mở tùy theo nhu cầu, mục đích của người sử dụng. Gắn thêm cảm biến đo độ ẩm phục vụ tưới tiêu trong trồng trọt, chăn nuôi,…

Vì thời gian và trình độ còn hạn chế nên việc thực hiện đồ án còn nhiều thiếu sót … Kính mong nhận được sự chỉ dẫn và góp ý tận tình của tất cả quý thầy cô.

Cuối cùng chúng em xin chân thành cảm ơn sự đóng góp ý kiến của tất cả quý thầy cô và sự nhiệt tình của các bạn đã giúp đỡ em thực hiện đề tài trong suốt thời gian qua.

Hải phòng, ngày 26 tháng 11 năm 2012

Sinh viên thực hiện

Trịnh Minh Đồng

TÀI LIỆU THAM KHẢO

1. Ths.Phạm Hùng Kim Khánh (2008). ***Giáo trình vi điều khiển***. Nhà xuất bản khoa học và kỹ thuật.

2. Ngô Diên Tập (2009). ***Kỹ thuật vi điều khiển với AVR***. Nhà xuất bản khoa học và kĩ thuật.

3. Lê Trung Thắng (2008). ***Tài liệu về chip AVR và lập trình C***.

4. Pavel Haiduc, HP InfoTech (1998), nhà phát triển phần mềm lập trình CodeVisionAVR.

5. Trang wed tham khảo:

http://www.dientuvietnam.net/

http://tailieu.vn/

http://alldatasheet.com/

http://www.hocavr.com/