**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA ĐIỆN – ĐIỆN TỬ**

**NGÀNH CÔNG NGHỆ KĨ THUẬT MÁY TÍNH**

🙢 🕮 🙠

**BÁO CÁO MÔN HỌC**

**KIẾN TRÚC VÀ TỔ CHỨC MÁY TÍNH**

**ĐỀ TÀI: TÌM HIỂU VI ĐIỀU KHIỂN ATMEGA328P VÀ LẬP TRÌNH CÁC VÍ DỤ CƠ BẢN SỬ DỤNG HỢP NGỮ ASM**

**Sinh viên thực hiện:**

1. Đỗ Tam Khôi 22119092
2. Nguyễn Phạm Duy Mẫn 22119102
3. Lê Thanh Quốc 22119124
4. Nguyễn Bình Nam Trung 22119142
5. Nguyễn Xuân Trường 22119146

6. Ngô Trần Tuấn Vỹ 22119158

**Giảng viên hướng dẫn:**

**TS. Phạm Văn Khoa**

**- Thành phố Hồ Chí Minh, tháng 6/2024 -**

**MỤC LỤC**

[**1.** **GIỚI THIỆU** 1](#_Toc169690853)

[**2.** **TỔNG QUAN** 2](#_Toc169690854)

[**2.1.** **Đối tượng nghiên cứu** 2](#_Toc169690857)

[**2.2.** **Công cụ hỗ trợ** 2](#_Toc169690858)

[**2.3.** **Phân chia công việc** 2](#_Toc169690859)

[**3.** **THỰC HIỆN** 4](#_Toc169690860)

[**3.1.** **Tổng quan vi điều khiển ATmega 328P** 4](#_Toc169690861)

[**3.2.** **Kiến trúc vi điều khiển ATmega328P** 6](#_Toc169690862)

[**3.2.1.** **Sơ đồ khối vi điều khiển ATmega328P** 6](#_Toc169690863)

[**3.2.2.** **Memory map của ATmega328P** 8](#_Toc169690864)

[**3.2.3.** **Các thanh ghi thông dụng** 9](#_Toc169690865)

[**3.3.1.** **Khái niệm** 12](#_Toc169690866)

[**3.3.2.** **Tập lệnh** 13](#_Toc169690867)

[**3.4.** **Hoạt động GPIO** 14](#_Toc169690868)

[**3.4.1.** **Khái niệm về GPIO** 14](#_Toc169690869)

[**3.4.2.** **Cách hoạt động của GPIO** 14](#_Toc169690870)

[**3.4.3.** **Một số ứng dụng của GPIO** 17](#_Toc169690871)

[**3.5.** **Hoạt động ngắt:** 17](#_Toc169690872)

[**3.5.1** **Khái niệm cơ bản về hoạt động ngắt:** 17](#_Toc169690873)

[**3.5.2** **Quy trình phục vụ ngắt** 18](#_Toc169690874)

[**3.5.3** **Hoạt động ngắt của ATmega328p** 18](#_Toc169690875)

[**3.6.** **Giao tiếp UART** 34](#_Toc169690876)

[**4.** **KẾT QUẢ THỰC HIỆN** 46](#_Toc169690877)

[**4.1.** **Hoạt động GPIO** 46](#_Toc169690878)

[**4.2.** **Hoạt động ngắt** 47](#_Toc169690879)

[**4.3.** **Hoạt động giao tiếp truyền thông UART** 49](#_Toc169690886)

[**5.** **KẾT LUẬN VÀ PHƯƠNG HƯỚNG PHÁT TRIỂN** 52](#_Toc169690887)

[**6.** **TÀI LIỆU THAM KHẢO** 53](#_Toc169690888)

1. **GIỚI THIỆU**

Vi điều khiển là một bộ vi xử lý nhỏ có chức năng đầy đủ, được thiết kế đặc biệt để có thể được nhúng vào các thiết bị điện tử. Đặc tính chủ yếu của vi điều khiển là khả năng lập trình để thực hiện nhiều loại tác vụ, từ đơn giản đến phức tạp.

Chúng thích hợp cho việc đặt trong các thiết bị điện tử như điều khiển từ xa, máy giặt, lò vi sóng, và nhiều hơn nữa. Đặc biệt, vi điều khiển có thể được lập trình để điều khiển các hệ thống phức tạp như hệ thống nhà thông minh, từ việc điều khiển đèn LED đến việc điều khiển toàn bộ hệ thống.

Để hiểu rõ về các vi điều khiển và cách hoạt động của chúng, nhóm em chọn đề tài “Tìm hiều vi điều khiển ATmega 328P và các ví dụ cơ bản ”. Ở đề tài này, chúng em sẽ đi sâu vào vi điều khiển, tìm hiểu cách hoạt động của nó. cũng như các khía cạnh kỹ thuật của vi điều khiển, bao gồm cách chúng hoạt động, cách lập trình chúng

1. **TỔNG QUAN**

## **Đối tượng nghiên cứu**

* Vi điều khiển ATmega328P

Đây là một vi điều khiển phổ biến do được sử dụng trong các board mạch Arduino, có tính ứng dụng cao

* Ngôn ngữ: Hợp ngữ

Đây là một loại ngôn ngữ lập trình được thiết kế để sử dụng bởi các nhà phát triển viết các chương trình chạy trực tiếp trên CPU của máy tính. Đây là ngôn ngữ cấp thấp, có nghĩa là nó khá gần với mã máy mà CPU có thể thực thi được.

* Hoạt động GPIO
* Hoạt động ngắt
* Hoạt động truyền thông UART

## **Công cụ hỗ trợ**

* Atmel Studio 7.0

Là phần mềm lập trình được tạo ra bởi Atmel, hỗ trợ lập trình ngôn ngữ C, C# và hợp ngữ

* Proteus

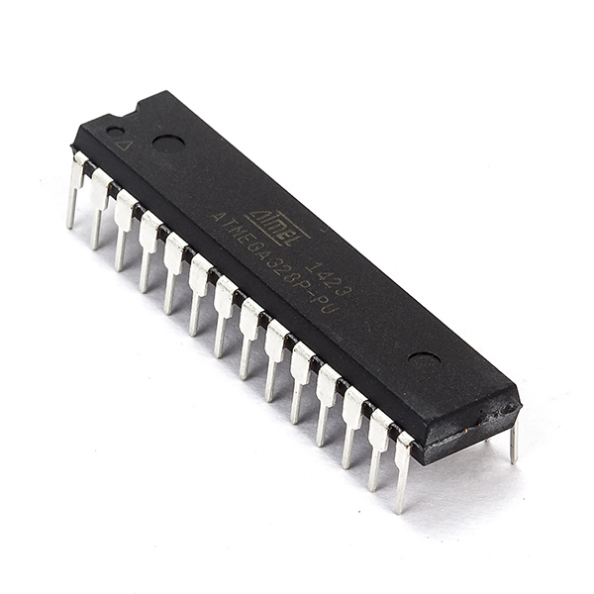
Là phần mềm thiết kế mạch và mô phỏng, dùng để mô phỏng hoạt động của các vi điều khiển

## **Phân chia công việc**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Người thực hiện  Công việc | Đỗ Tam Khôi | Nguyễn Phạm Duy Mẫn | Lê Thanh Quốc | Nguyễn Xuân Trường | Nguyễn Bình Nam Trung | Ngô Trần Tuấn Vỹ |
| Giới thiệu về vi điều khiển ATmega328P |  |  |  |  |  |  |
| Kiến trúc vi điều khiển |  |  |  |  |  |  |
| Kiến trúc tập lệnh |  |  |  |  |  |  |
| Hoạt động GPIO |  |  |  |  |  |  |
| Hoạt động ngắt |  |  |  |  |  |  |
| Hoạt động truyền thông UART |  |  |  |  |  |  |
| Mô phỏng lập trình |  |  |  |  |  |  |
| Bản trình chiếu |  |  |  |  |  |  |
| Báo cáo |  |  |  |  |  |  |

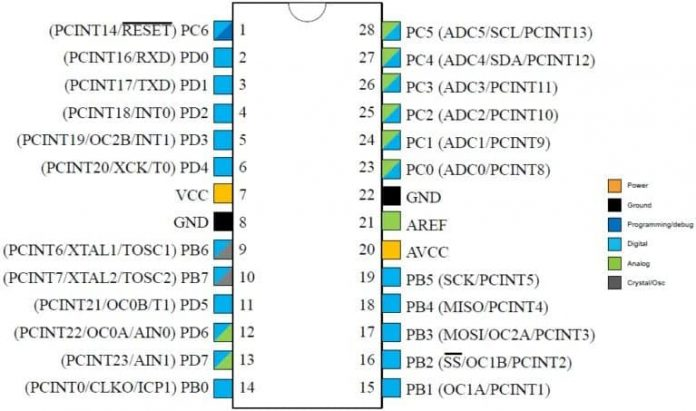
1. **THỰC HIỆN**
   1. **Tổng quan vi điều khiển ATmega 328P**

ATmega 328P là một bộ vi điều khiển tiên tiến và có nhiều tính năng, nó nó được biết đến rộng rãi trong những vi điều khiển của Atmel vì được sử dụng trong bo mạch Arduino UNO. là một vi điều khiển thuộc họ megaAVR được sử dụng để xử lý các chương trình lớn. ATmega 328p là một vi điều khiển 8 bit dựa trên kiến trúc RISC với bộ nhớ chương trình vô cùng lớn



Hình 1: Vi điều khiển ATmega328P

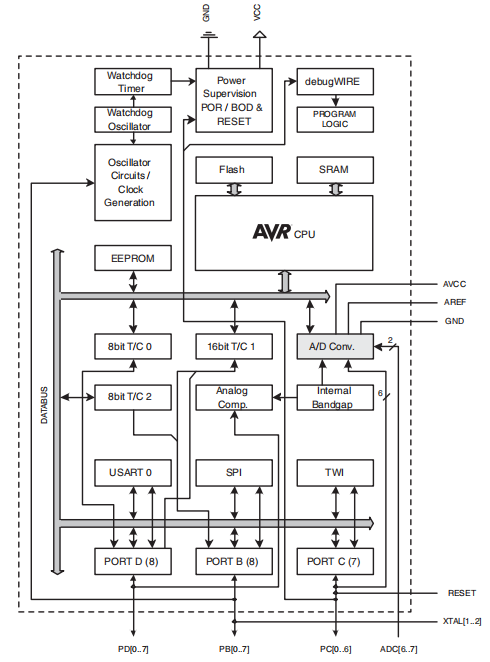
Sơ đồ chân ATmega328p:



Hình 2: Sơ đồ chân ATmega328P

Chức năng các chân:

* Chân I/O digital: Có ba cổng digital là PORT B, PORT C và PORT D. Các chân này có thể được sử dụng làm I/O digital. Phải xác định trước các chân trước khi sử dungj làm I/O hoặc các chức năng khác vì các chân này không có chức năng cụ thể . Các chân I/O digital của bộ điều khiển là: PB (0-7), PC (0-6) và PD (0-7)
* Chân ngắt: IN0 - GPIO4, IN1 - GPIO5
* Module giao tiếp UART: RX - GPIO2, TX - GPIO3
* Sử dụng chân xung clock bên ngoài để giữ xung nhịp đồng bộ hóa dữ liệu. XCK-GPIO6
* Giao thức SPI ATmega328P: MOSI - GPIO17, MISO - GPIO18, SS - GPIO16, SCK - GPIO19
* Module giao tiếp I2C: SDA - GPIO27, SCL - GPIO28
* Các module timer: T0 - GPIO6, T1 - GPIO11, TOSC1 - GPIO9, TOSC2 - GPIO10, ICP1 - GPIO
* Hệ thống xung clock: Xung clock bên trong và xung clock bên ngoài có thể được chia tần số bởi Prescaler và nhận giá trị ở chân xuất ra bên ngoài: CLKO - GPIO14
* Module so sánh: AN0 (Positive) - GPIO12, AN1 (Negative) - GPIO13
* Kênh CAPTURE / COMPARE / PWM: OC0B - GPIO11, OC0A - GPIO12, OC1A - GPIO15, OC1B - GPIO16, OC2A - GPIO17, OC2B - GPIO5
* Kênh chuyển đổi tín hiệu analog sang digital: ADC0 - GPIO23, ADC1 - GPIO24, ADC2 - GPIO25, ADC3 - GPIO26, ADC4 - GPIO27, ADC5 - GPIO28, AVCC - Chân 20
* Chân AREF: AREF - GPIO21
* Chân RESET: RESET- GPIO1
* Chân nguồn: Các chân nguồn của vi điều khiển là: VCC - Chân 7, GND - Chân 8, chân 22
* Bộ dao động: XTAL1 - GPIO9, XTAL2 - GPIO10
  1. **Kiến trúc vi điều khiển ATmega328P**
     1. **Sơ đồ khối vi điều khiển ATmega328P**



Hình 3: Sơ đồ khối bên trong ATmega328P

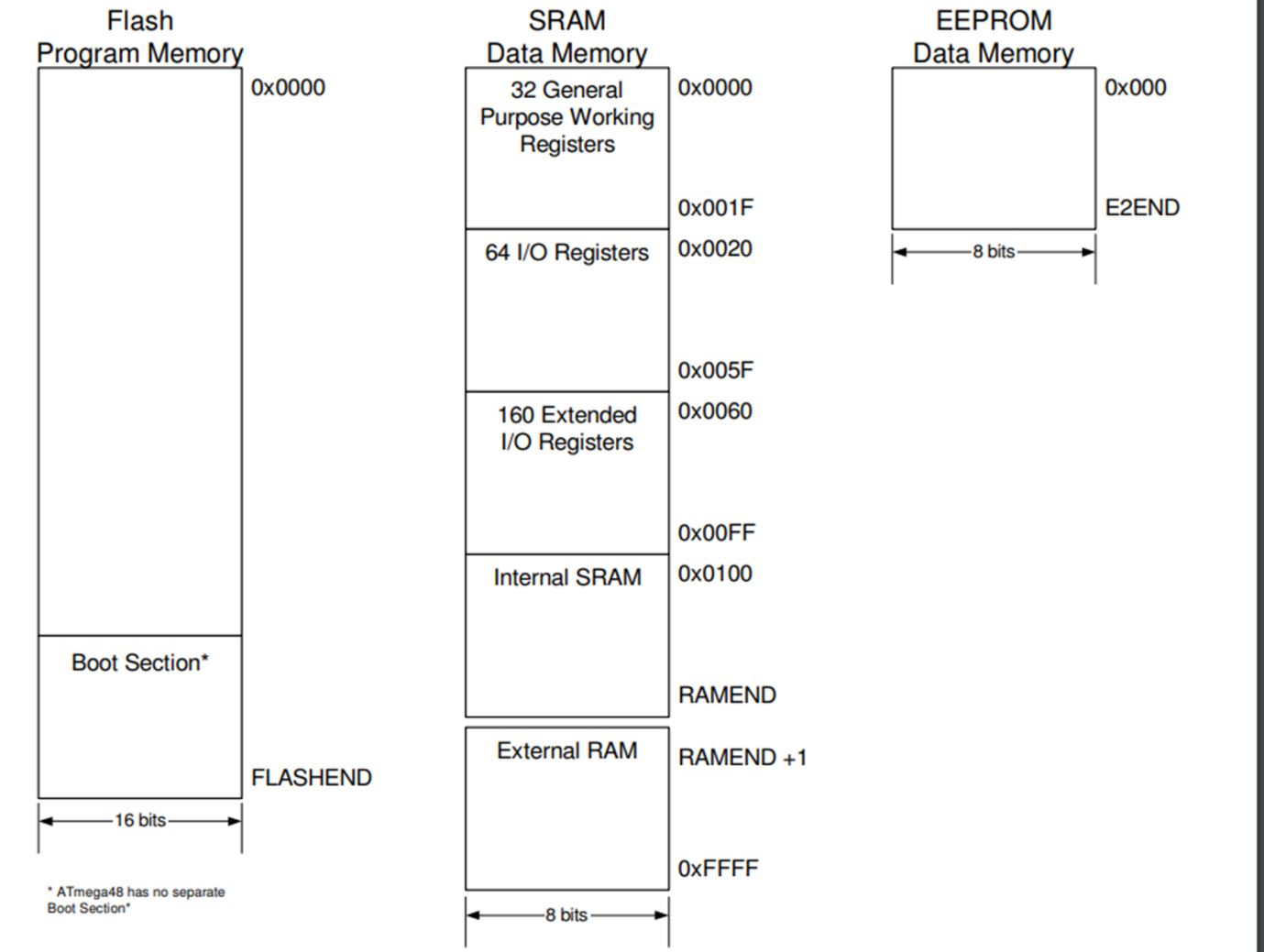
ATmega328P sử dụng kiến trúc Havard và vì vậy nó sẽ bao gồm 2 bộ nhớ riêng biệt là bộ nhớ dữ liệu và bộ nhớ lệnh.

Bộ nhớ lệnh: CPU tìm nạp và thực thi mã lệnh được lưu trong bộ nhớ Flash. Bộ nhớ Flash có dung lượng lên đến 32K byte lập trình trong hệ thống với tính năng đọc và ghi

Bộ nhớ dữ liệu: SRAM và EEPROM là hai dạng bộ nhớ dữ liệu được ATmega328P sử dụng. Trong khi EEPROM cung cấp 1K byte bộ nhớ không bay hơi giúp lưu trữ ổn định cho dữ liệu phải được lưu trữ thì SRAM cung cấp 2K byte một bộ nhớ bay hơi

ATmega328P cung cấp các tính năng sau: 23 dòng I / O mục đích chung, 32 thanh ghi làm việc cho mục đích chung. Có 1 Watchdog timer có thể lập trình là một bộ định thời theo dõi các chương trình vi điều khiển nhằm kiểm tra xem các lệnh có nằm trong tầm kiểm soát hay đã ngừng hoạt động hay chưa. Hệ thống POR đảm bảo vi điều khiển sẽ được khởi động ở cùng một điều kiện mỗi khi được bật nguồn và hệ thống BOD để giám sát mức điện áp hoạt động trong quá trình hoạt động bằng cách so sánh nó với mức kích họat cố định. Hệ thống debugWIRE sử dụng giao diện một một dây hai chiều để điều khiển luồng lệnh, thực thi các lệnh trong CPU và lập trình bộ nhớ không bay hơi khác nhau. Hệ thống Analog Comp dùng để so sánh điện áp và hệ thống A/D Conv dùng để là hệ thống mạch thực hiện chuyển đổi một tín hiệu analog (tín hiệu tương tự) liên tục, ví dụ như tín hiệu âm thanh thanh micro, hay tín hiệu ánh sáng trong máy ảnh kỹ thuật số, thành tín hiệu số. Ba Timers / Counters linh hoạt với các chế độ so sánh, ngắt bên trong và bên ngoài. Cổng nối tiếp SPI là giao thức giao tiếp nối tiếp đồng bộ được sử dụng để giao tiếp với nhiều thiết bị. USART là loại giao tiếp nối tiếp có thể được lập trình để giao tiếp đồng bộ hoặc không đồng bộ. Giao tiếp TWI được sử dụng cho các vi điều khiển thông thường. Giao tiếp TWI cho phép nhà thiết kế hệ thống kết nối tới 128 thiết bị khác nhau chỉ bằng hai đường bus hai chiều, một cho đồng hồ (SCL) và một cho dữ liệu (SDA). Phần cứng bên ngoài duy nhất cần thiết để triển khai bus là một điện trở kéo lên duy nhất cho mỗi đường bus TWI. Bộ so sánh (Analog comp) sử dụng để so sánh điện áp hai đầu vào analog, với đầu ra cho biết điện áp nào vào cao hơn

* + 1. **Memory map của ATmega328P**



Hình 4: Memory map ATmega328P

Do sủ dụng kiến trúc Havard nên ATmega 328P có 3 bộ nhớ riêng biệt là: 32K byte bộ nhớ Flash, 2K byte bộ nhớ SRAM và 1K byte bộ nhớ EEPROM. Đối với SRAM, bộ nhớ này bao gồm 32 byte đầu để chứa các thanh ghi chung, 64 byte để chứa các thanh ghi liên quan đến các chức năng ngoại vi trong vi điều khiển, 1 byte chứa thanh ghi trạng thái, và 160 chứa các ngoại vi mở rộng, 2048 byte là SRAM nội bộ hay còn được gọi là SRAM dữ liệu và có thể được sử dụng làm ngăn xếp (Stack Pointer).

* + 1. **Các thanh ghi thông dụng**

**Thanh ghi trạng thái**

Thanh ghi trạng thái được sử dụng để lưu những kết quả của lệnh số học thực hiện gần đây nhất. Thông tin này có thể được sử dụng để thay đổi dòng lệnh nhằm thực hiện các chương trình con có điều kiện. Thanh ghi trạng thái được cập nhật sau các hoạt động của ALU, như được chỉ định trong tham chiếu tập lệnh, giúp loại bỏ các lệnh so sánh chuyên dụng, giúp code nhanh và gọn hơn



Hình 5: Thanh ghi trạng thái ATmega328P

**Bit 0 – C (Carry Flag: Cờ nhớ)**: là bit nhớ từ bit 7 trong các phép đại số hoặc logic. Cờ C=1 khi kết quả tràn từ bit 7 sang

**Bit 1 – Z (Zero Flag: Cờ 0)**: cờ này được set bằng 1 nếu kết quả phép toán đại số hay phép Logic bằng 0 và bằng 1 với kết quả phép logic khác 0

**Bit 2 – N (Negative Flag: Cờ âm)**: cờ này được set nếu kết quả phép toán đại số hay phép Logic là số âm. N=0 thì kếtr quả là số dương, N=1 thì kết quả là số số âm

**Bit 3 – V (Two’s complement Overflow Flag: Cờ tràn của bù 2)**: Đối với số nhị phân 8 bit, chỉ có thể biểu diễn số nhị phân có dấu dưới dạng bù 1 từ 127 đến -128. Nếu giá trị vượt quá khoảng này, thanh ghi sẽ được set lên là 1

**Bit 4 – S (Sign Bit: Bit dấu)**: Cờ S báo dấu chính xác của kết quả, S=0 báo kết quả dương, S=1 báo kết quả âm. Bit S là kết quả phép XOR giữa 1 cờ N và V, S=N V.

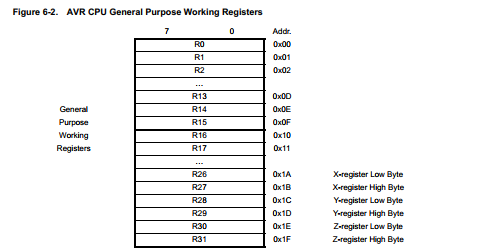
**Bit 5 – H (Half Carry Flag: Cờ nhờ nữa)**: cờ H là cờ nhớ trong 1 vài phép toán đại số và phép Logic. Cờ H=1 khi khi có tràn từ bit 3 sang bit 4. Cờ này hiệu quả đối với các phép toán với số BCD.

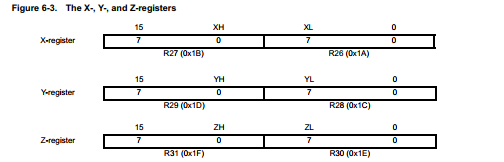
**Bit 6 – T (Bit Copy Storage)**: được sử dụng trong 2 Instruction BLD (Bit LoaD) và BST (Bit STorage). Lệnh BLD cho phép sao chép bit T vào 1 bit bất kỳ trong 1 thanh ghi thuộc tập thanh ghi GPRs. Lệnh BST cho phép sao chép 1 bit bất kỳ trong trong1 thanh ghi GPRs vào bit T. Thường được dùng để sao chép bit từ giá trị cảm biến

**Bit 7 – I (Global Interrupt Enable)**: Bit này phải được set lên 1 nếu trong chương trình có sử dụng ngắt. Sau khi set bit này, bạn muốn kích hoạt loại ngắt nào cần set các bit ngắt riêng của ngắt đó. Hai instruction dùng riêng để Set và Clear bit I là SEI và CLI.

**Thanh ghi chung**

CPU sử dụng các thanh ghi 8 bit này để chứa dữ liệu tạm thời dùng trong trong các phép toán số học và logic. Các thanh ghi này từ R0 đến R31 có địa chỉ tương ứng 0x00 đến 0x1F cho các họ AVR. Tập thanh ghi chia làm 2 phần, mỗi phần có 16 thanh ghi: R0 - R15, R15 - R31. Đối với thanh ghi từ R0 - R15 thì các lệnh ngay lập tức chẳng hạn như LDI sẽ không được sử dụng, vì vậy ta thường ưu tiên sử dụng thanh ghi từ R16 – R30.





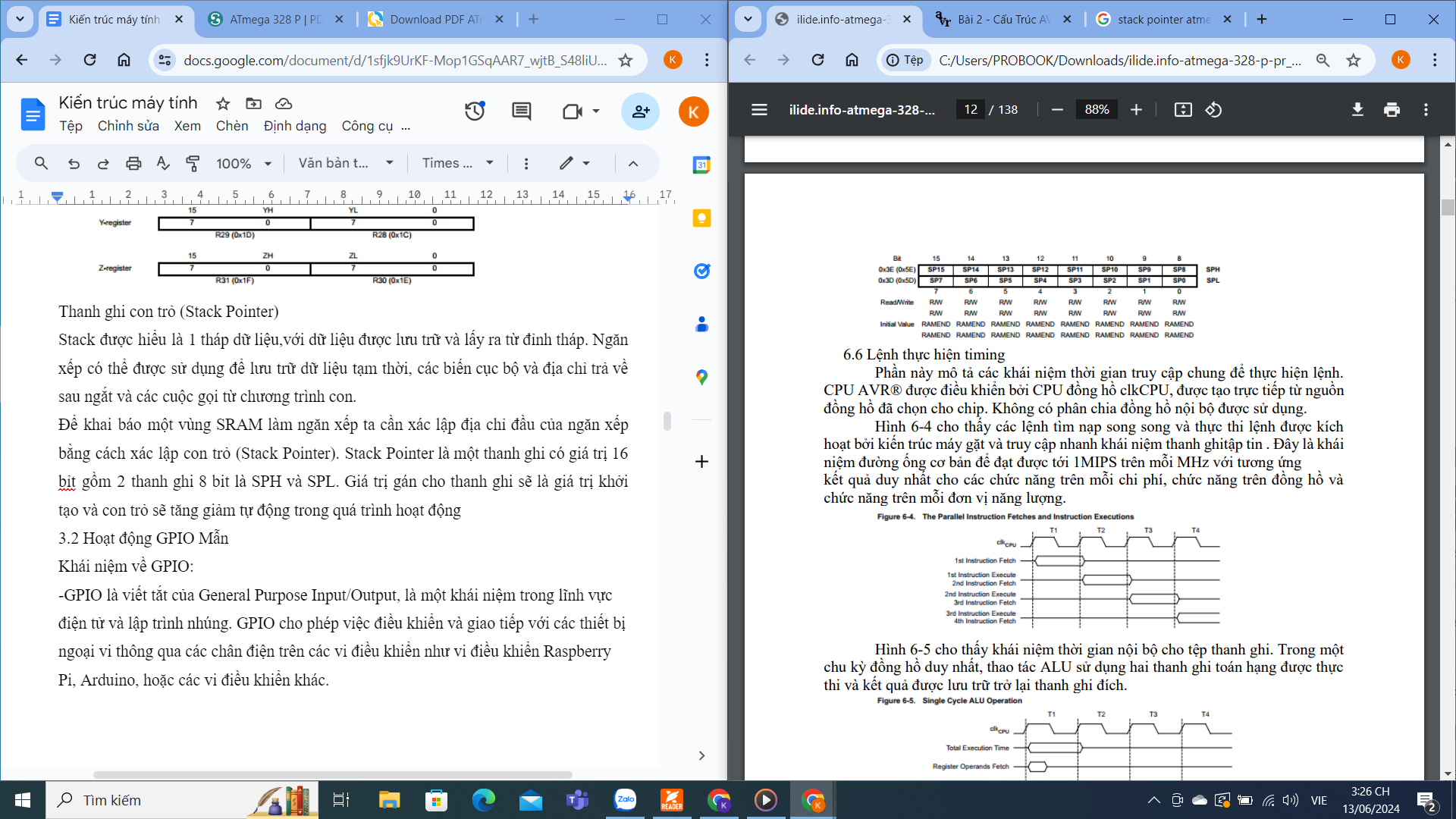
Hình 6: Thanh ghi chung ATmega328P

Có 6 thanh ghi R26 đến R31 được ghép thành 3 thanh ghi là 16 bit là thanh ghi X,Y và Z. Các thanh ghi này là con trỏ thanh ghi địa địa chỉ gián tiếp dùng dùng để truy truy xuất bộ nhớ dữ dữ liệu

**Thanh ghi con trỏ (Stack Pointer)**

Stack được hiểu là 1 tháp dữ liệu,với dữ liệu được lưu trữ và lấy ra từ đỉnh tháp. Ngăn xếp có thể được sử dụng để lưu trữ dữ liệu tạm thời, các biến cục bộ và địa chỉ trả về sau ngắt và các cuộc gọi từ chương trình con.

Để khai báo một vùng SRAM làm ngăn xếp ta cần xác lập địa chỉ đầu của ngăn xếp bằng cách xác lập con trỏ (Stack Pointer). Stack Pointer là một thanh ghi có giá trị 16 bỉt gồm 2 thanh ghi 8 bit là SPH và SPL. Giá trị gán cho thanh ghi sẽ là giá trị khởi tạo và con trỏ sẽ tăng giảm tự động trong quá trình hoạt động



Hình 7: Thanh ghi con trỏ (Stack Pointer)

* 1. **Kiến trúc tập lệnh của ATmega328p**
     1. **Khái niệm**

Kiến trúc tập lệnh (ISA) là bộ quy tắc và giao diện mà một vi xử lý hoặc một bộ xử lý sử dụng để thực hiện các lệnh và thao tác với bộ nhớ. ISA định rõ các loại lệnh mà một vi xử lý có thể thực hiện, cách các lệnh được mã hóa, cấu trúc của các thanh ghi và bộ nhớ, cũng như cách thông tin được truyền đạt giữa các thành phần của hệ thống như bộ nhớ, bộ điều khiển và các thành phần ngoại vi.

Kiến trúc tập lệnh quy định cách mà các chương trình máy (code) được viết và biên dịch, và nó cũng cung cấp một giao diện cho phần cứng, cho phép các lập trình viên tương tác với hệ thống. ISA thường được phân loại theo nhiều cách, bao gồm chiều rộng lệnh (8-bit, 16-bit, 32-bit, 64-bit), kiểu bộ nhớ (Von Neumann hoặc Harvard), các loại thanh ghi, và các chế độ chạy (user mode, supervisor mode, kernel mode). ISA cũng có thể bao gồm các tiêu chuẩn giao tiếp với các thành phần ngoại vi như UART, SPI, I2C, và GPIO.

* + 1. **Tập lệnh**

30 Mã lệnh

* Các lệnh ALU: ADC, ADD, AND, NOP, CP, EOR, OR, MOV
* Các lệnh ngay lập tức: CPI, ORI, ANDI, LDI
* Các lệnh Logic một ngôi: COM, NEG, ASR, LSR
* Các lệnh Load/Store: LDS, STS
* Các lệnh nhảy: BRBS, BRBC (còn gọi là BRZS, BRZC, BRCS, BRCC)

Nhảy có điều kiện dựa trên các bit của SREG ("Mã điều kiện")

* Các lệnh Nhập/Xuất: IN, OUT
* Các lệnh Gọi/Nhảy: CALL, JMP
* Các lệnh Trả về: RET, RETI
* Các lệnh Stack: PUSH, POP
* Các lệnh Nhảy Tương đối: RCALL, RJMP

Chế độ địa chỉ nhằm xác định cách giải quyết vị trí của một toán hạng

Địa chỉ không phải là bộ nhớ: ngay lập tức, trực tiếp thanh ghi, v.v.

Địa chỉ bộ nhớ: trực tiếp dữ liệu, gián tiếp dữ liệu, gián tiếp dữ liệu với độ lệch, v.v

**Các lệnh Điều khiển**

Được sử dụng để thay đổi chuỗi các lệnh (bằng cách thay đổi PC)

Nhánh có Điều kiện (Nhánh được thực hiện nếu một điều kiện cụ thể là đúng (BRBC, BRBS)). BRCC/BRCS là đồng nghĩa của BRBC/BRBS với trường "sss" là chỉ số bit SREG của mã điều kiện. Độ lệch có dấu được cộng vào PC để thu được PC mới. Nếu không, nhánh không được thực hiện. PC không thay đổi, chỉ trỏ tới lệnh tiếp theo theo thứ tự

Nhánh không có điều kiện (Nhảy hoặc Gọi). Luôn thay đổi PC (không có điều kiện): JMP, RJMP, CALL, RCALL

* 1. **Hoạt động GPIO** 
     1. **Khái niệm về GPIO**

GPIO là viết tắt của General Purpose Input/Output, là một khái niệm trong lĩnh vực điện tử và lập trình nhúng. GPIO cho phép việc điều khiển và giao tiếp với các thiết bị ngoại vi thông qua các chân điện trên các vi điều khiển như vi điều khiển Raspberry Pi, Arduino, hoặc các vi điều khiển khác.

Chức năng của các chân GPIO có thể được cấu hình để hoạt động như đầu vào hoặc đầu ra. Khi được cấu hình là đầu vào, chúng có thể đọc trạng thái của tín hiệu đầu vào từ các thiết bị ngoại vi như cảm biến hoặc công tắc. Khi được cấu hình là đầu ra, chúng có thể điều khiển các thiết bị khác như đèn LED, servo motor, hay bất kỳ thiết bị nào có thể được điều khiển bằng tín hiệu điện.

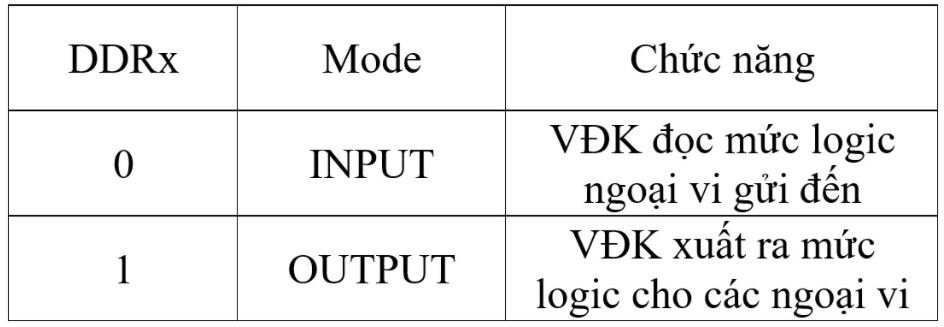
Việc sử dụng GPIO cho phép những ứng dụng nhúng có khả năng tương tác với môi trường xung quanh, làm cho chúng trở nên linh hoạt và có thể thực hiện nhiều chức năng khác nhau.

* + 1. **Cách hoạt động của GPIO**

Chân I/O digital: Bộ vi điều khiển này có ba cổng digital (B, C, D) là PORT B, PORT C và PORT D. Tất cả các chân này có thể được sử dụng làm I/O digital. Trên hết, mỗi cổng có thể được sử dụng cho các chức năng khác. Để sử dụng làm I/O hoặc cho các chức năng khác, phải được xác định trước vì các chân không có chức năng mặc định:

Với mỗi port B, C, D của ATmega328p thì chúng ta cần biết cách sử dụng 3 thanh ghi:

* DDRx (Data Direction Register): Các bit trong thanh ghi này quy định hướng dữ liệu của các chân GPIO. Mỗi bit trong DDRx tương ứng với một chân GPIO. Khi bit tương ứng được thiết lập thành 1, chân GPIO tương ứng được đặt làm đầu ra; khi bit được thiết lập thành 0, chân GPIO trở thành đầu vào.



Hình 8: Thanh ghi DDR

* PORTx (Port Output Register): Các bit trong thanh ghi này quản lý trạng thái đầu ra của các chân GPIO khi chúng được đặt là đầu ra. Mỗi bit trong PORTx tương ứng với một chân GPIO. Khi một bit trong PORTx được thiết lập thành 1, chân GPIO tương ứng sẽ được đặt ở trạng thái cao (HIGH); khi bit được thiết lập thành 0, chân GPIO sẽ ở trạng thái thấp (LOW).

Với một chân đang là INPUT (DDRx, n = 0 với x thuộc B,C,D; n thuộc 0 => 7) :

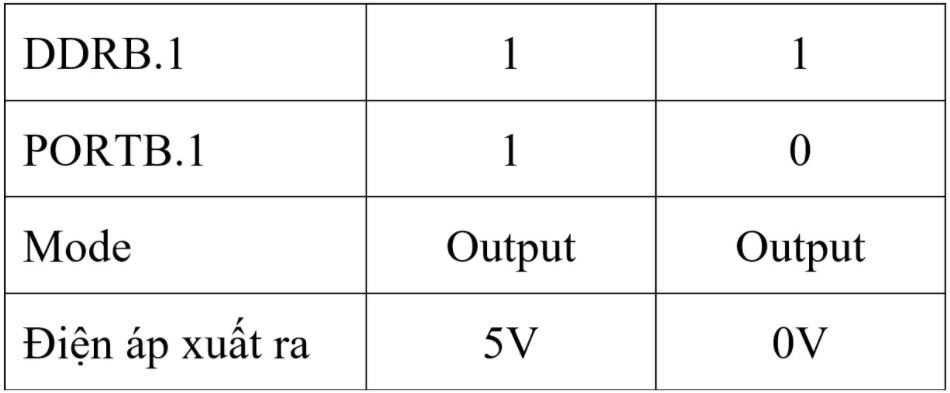
* Khi đặt mức 1, điện trở nội kéo lên được kích hoạt (chế độ Pull Up).
* Khi đặt mức 0, tắt điện trở kéo lên, chân đó chuyển sang chế độ tổng trở cao (HiZ).



Hình 9: Thanh ghi PORT đầu ra

Với một chân đang là OUTPUT (DDRx, n = 1):

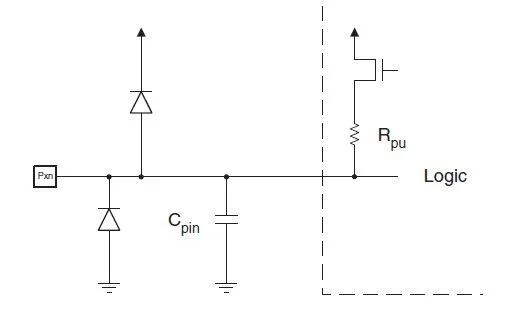
* Khi đặt mức 1, chân đó xuất ra mức 1 (5V)
* Khi đặt mức 0, chân đó xuất ra mức 0 (0V)



Hình 10: Thanh ghi PORT đầu vào

* PINx (Port Input Pins): tương tự với 2 thanh ghi trên, mỗi bit của nó đại diện cho 1 chân của PORT x, giá trị trên mỗi bit thuộc thanh ghi này trả về mức logic hiện tại của chân tương ứng, tức là chúng ta sẽ đọc được mức logic của chân bằng bit tương ứng trong thanh ghi PIN.

Chân I/O: Tất cả các cổng đều có điện trở kéo lên mà không phụ thuộc vào điện áp cung cấp. Các chân đều có điốt bảo vệ cho cả VCC và Ground.



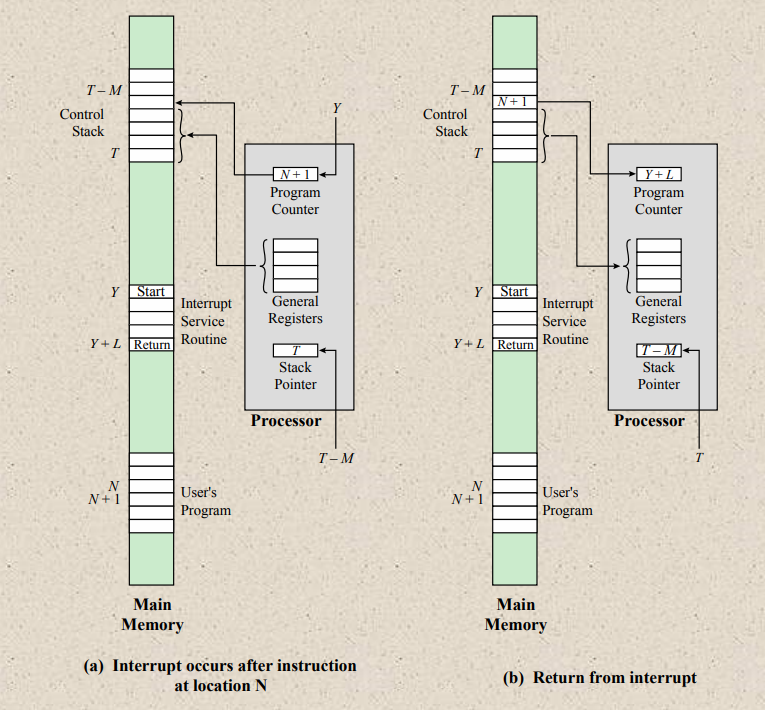
Hình 11: Hình minh họa chân I/O

* + 1. **Một số ứng dụng của GPIO**
* Điều khiển đèn Led: có thể sử dụng các chân GPIO để điều khiển đèn LED, bật/tắt hoặc điều chỉnh độ sáng của đèn.
* Đọc nút nhấn: Các chân GPIO có thể được cấu hình để đọc trạng thái của các nút nhấn hoặc các công tắc.
* Giao tiếp với cảm biến: Đọc dữ liệu từ các cảm biến như cảm biến nhiệt độ, cảm biến ánh sáng, v.v
* Giao tiếp với các thiết bị ngoại vi
* Ghi và đọc dữ liệu từ bộ nhớ ngoại vi
  1. **Hoạt động ngắt:** 
     1. **Khái niệm cơ bản về hoạt động ngắt:**

Ngắt (Interrupts), là một “tín hiệu khẩn cấp” gửi đến bộ xử lí, yêu cầu bộ xử lí tạm ngưng 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 được gọi là trình phục vụ ngắt - ISR (Interrupt service routine). Sau khi kết thúc nhiệm vụ trong ISR, bộ xử lí sẽ quay về thực hiện các nhiệm vụ dang dở.

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.

1. **Quy trình phục vụ ngắt**



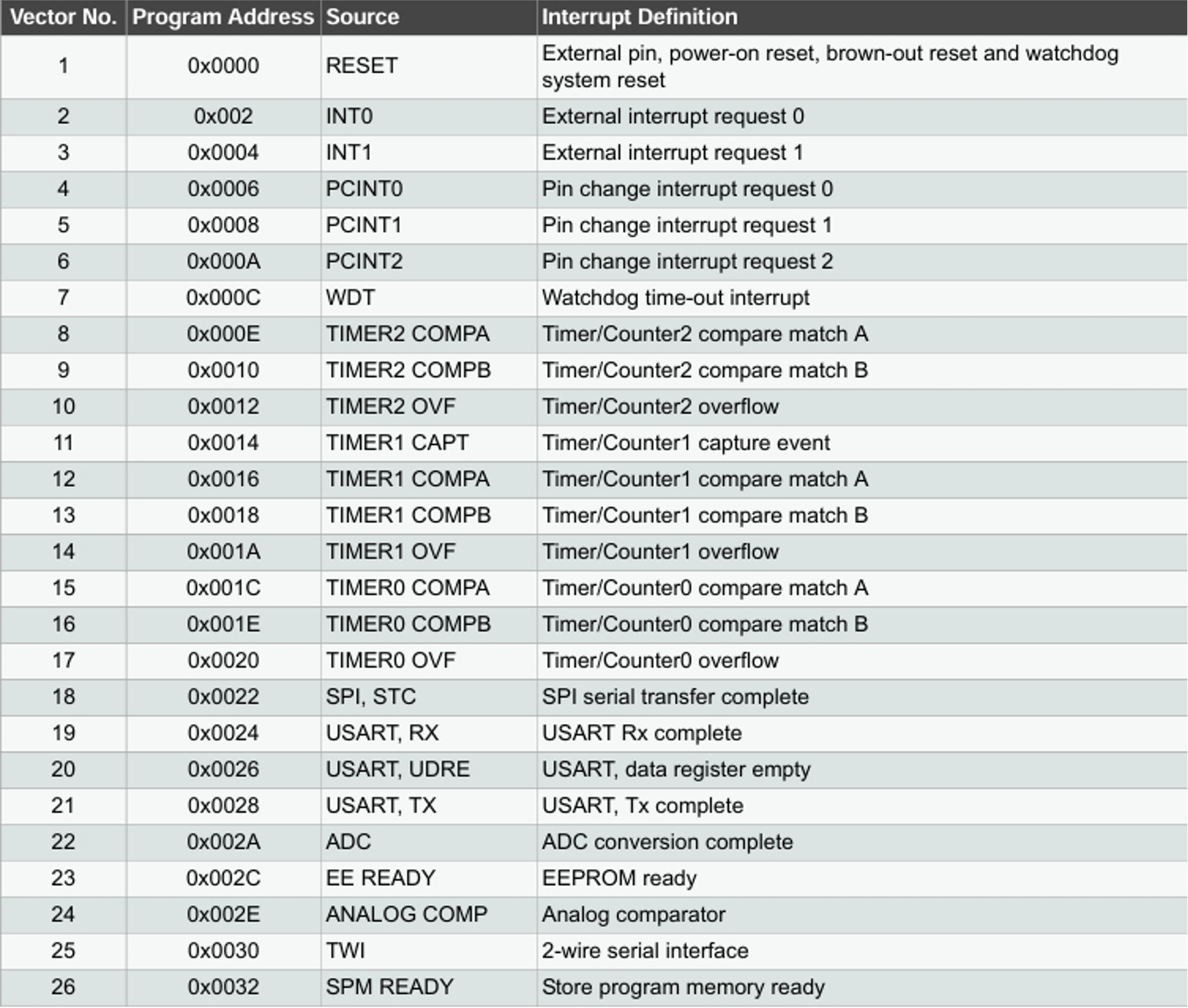
Hình 12: Quy trình phục vụ ngắt

Ở hình a: ngắt xảy ra khi chương trình chính thực hiện xong chương trình tại vị trí thứ N. Sau đó nội dung của chương trình tại vị trí (N+1) sẽ được lưu vào stack tại vị trí (T-M). Lúc này, chương trình sẽ thực hiện chương trình ngắt tại vị trí Y.

Ở hình b: khi trở về từ ngắt, dữ liệu của PC và General Registers sẽ được khôi phục từ stack. Stack pointer cũng sẽ được đặt lại T. Lúc này chương trình chính sẽ tiếp tục thực thi chương trình tại vị trí N+1.

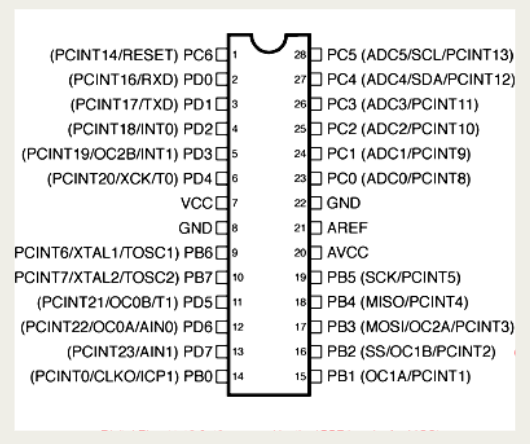
1. **Hoạt động ngắt của Atmega328P**

ATmega328P hỗ trợ cho 25 nguồn ngắt khác nhau. Danh sách đầy đủ các vectơ được hiển thị trong bảng Vector ngắt. Danh sách này cũng xác định mức độ ưu tiên của các ngắt khác nhau. Địa chỉ càng thấp thì mức độ ưu tiên càng cao. RESET có mức ưu tiên cao nhất và tiếp theo là INT0 – Yêu cầu ngắt bên ngoài 0.



Bảng 1: Vector ngắt của Atmega328P

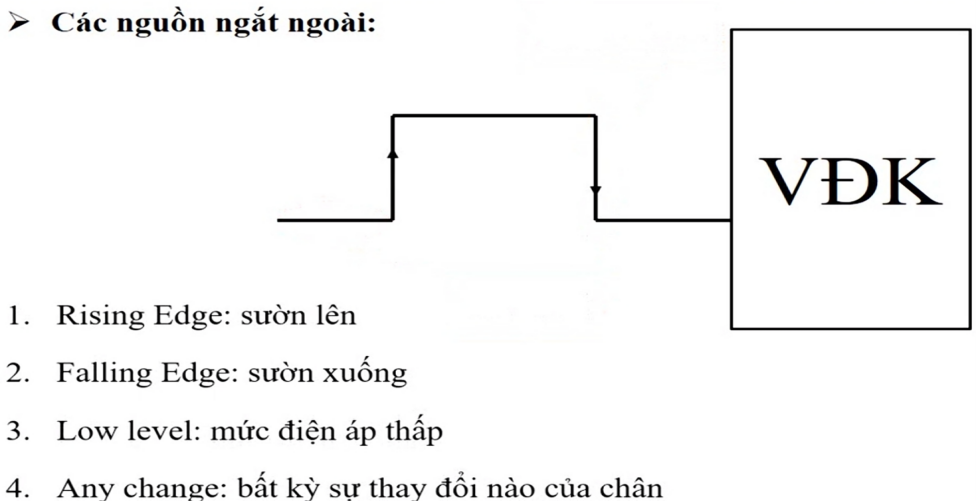
ATmega328p có 2 chân ngắt là PD2(INT0) và PD3(INT1)



Hình 13: Sơ đồ chân của ATmega328P

* + 1. **Ngắt ngoài**

Ngắt ngoài là ngắt xảy ra khi có sự thay đổi mức/cạnh trên các chân ngắt của vi điều khiển.

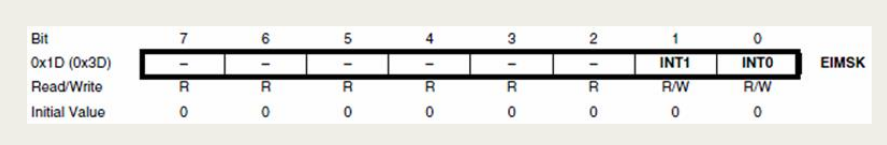


Hình 14: Các nguồn ngắt ngoài

Giả sử chúng ta dùng button để tạo ra ngắt thì sẽ có bốn chế độ có thể xảy ra khi chúng ta nhấn và thả các button. Nếu không nhấn, trạng thái các chân INT là HIGH, 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.

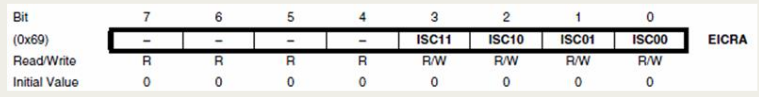
**Các thanh ghi sử dụng trong ngắt ngoài của Atmega328P:**

**Thanh ghi EIMSK**: là thanh ghi dùng để kích hoạt các chân ngắt. EIMSK là một thanh ghi 8 bit nhưng chỉ có 2 bit cao (bit 1 và bit 0) là được sử dụng cho điều khiển ngắt. Bit 1 – 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.



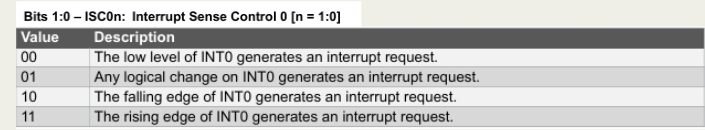
Hình 15: Thanh ghi EIMSK

**Thanh ghi EICRA:** là thanh ghi xác lập chế độ ngắt. EICRA là một thanh ghi 8 bit nhưng đối với hoạt động ngắt ngoài, chúng ta chỉ quan tâm đến 4 bit thấp của nó.

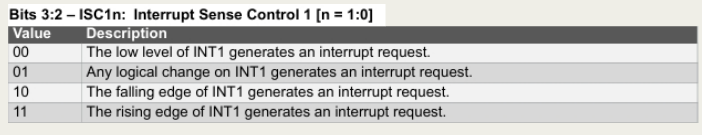


Hình 16: Thanh ghi EICRA

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 tóm tắt bên dưới cho ta biết chức năng của các bit trên, đây là bảng “chân trị” của 2 bit ISC11, ISC10 và bảng chân trị cho các bit ISC01, ISC00.

******

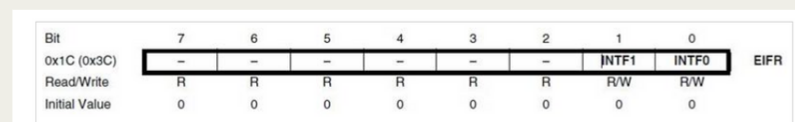
Bảng 2: Chân trị của các bit 1 và bit 0



Bảng 3: Chân trị của các bit 3 và bit 2

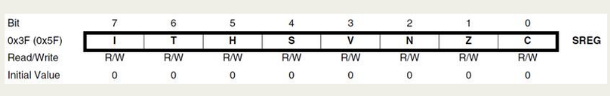
Ví dụ chúng ta muốn cho INT1 là ngắt cạnh xuống (Falling Edge) trong khi INT0 là ngắt cạnh lên (Rising Edge), chúng ta phải đặt cho thanh ghi EICRA có giá trị là 0x0B (0x0B = 0000**1011** nhị phân) trong chương trình.

**Thanh ghi EIFR:** là thanh ghi cờ ngắt. Thanh ghi này 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 EIFR được trình bày trong hình ngay bên dưới.



Hình 17: Thanh ghi EIFR

**Thanh ghi SREG**: là thanh ghi trạng thái. 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. Tức là chúng ta phải set cho bit 7 - I lên 1 để cho phép ngắt toàn cục. Trong lập trình Assembly thì có hai lệnh riêng để Set và Clear bit I là SEI và CLI.



Hình 18: Thanh ghi SREG

**Timer/Counter Interrupts (Ngắt Bộ Định Thời/Đếm)**

Là một số sự kiện khẩn cấp bên trong bộ vi điều khiển xảy ra bằng cách sử dụng bộ định thời của vi điều khiển, buộc vi điều khiển tạm dừng thực hiện chương trình hiện tại, phục vụ ngay lập tức nhiệm vụ mà ngắt yêu cầu – nhiệm vụ này gọi là trình phục vụ ngắt (**ISR:** Interrupt Service Routine).

ATmega328P có ba bộ định thời/tính toán (Timer/Counter): Timer/Counter0 (8-bit), Timer/Counter1 (16-bit) và Timer/Counter2 (8-bit).

**Timer/Counter0 (8-bit) :**

**Chức năng:**

* Hai đơn vị so sánh đầu ra độc lập.
* Thanh ghi so sánh đầu ra được đệm đôi.
* Xóa bộ đếm thời gian khi so sánh khớp (tự động tải lại).
* Bộ điều biến độ rộng xung đúng pha (PWM) không bị trục trặc.
* Chu kỳ xung điện có thể thay đổi.
* Máy phát tần số.
* Ba nguồn ngắt độc lập (TOV0, OCF0A và OCF0B).

**Sơ đồ khối:**

A diagram of a computer system

Description automatically generated

Hình 19: Sơ đồ khối của bộ định thời Timer/Counter0

**Timer/Counter1 (16-bit)**

**Chức năng**

* Thiết kế 16-bit đích thực (tức là cho phép điều khiển 16-bit PWM).
* Hai đơn vị so sánh đầu ra độc lập.
* Thanh ghi so sánh đầu ra được đệm đôi.
* Một đơn vị chụp đầu vào.
* Bộ khử tiếng ồn thu đầu vào.
* Xóa bộ đếm thời gian khi so sánh khớp (tự động tải lại).
* Bộ điều biến độ rộng xung đúng pha (PWM) không bị trục trặc.
* Chu kỳ xung điện có thể thay đổi.
* Máy phát tần số.
* Bộ đếm sự kiện bên ngoài.
* Bốn nguồn ngắt độc lập (TOV1, OCF1A, OCF1B và ICF1).

**Sơ đồ khối:**

A diagram of a machine

Description automatically generated

Hình 20: Sơ đồ khối của bộ định thời Timer/Counter 1

**Timer/Counter2 (8-bit).**

**Chức năng:**

* Bộ đếm kênh đơn.
* Xóa bộ đếm thời gian khi so sánh khớp (tự động tải lại).
* Bộ điều biến độ rộng xung đúng pha (PWM) không bị trục trặc.
* Máy phát tần số.
* Bộ đếm trước đồng hồ 10 bit.
* Tràn và so sánh khớp các nguồn ngắt (TOV2, OCF2A và OCF2B).
* Cho phép xung nhịp từ tinh thể đồng hồ 32kHz bên ngoài độc lập với đồng hồ I/O.

**Sơ đồ khối:**

A diagram of a data center

Description automatically generated

Hình 21: Sơ đồ khối của bộ định thời Timer/Counter 2

**Các nguồn ngắt**

Mỗi bộ Timer/Counter có các ngắt riêng lẻ:

**Overflow Interrupt**: Xảy ra khi giá trị đếm vượt quá giá trị tối đa.

**Compare Match Interrupt**: Xảy ra khi giá trị đếm bằng với giá trị so sánh được cài đặt.

**Bộ chia tỉ lệ (pre-scaler):**

Prescaler là chia tần số dao động của nguồn dao động (thạch anh, bộ dao động riêng của Watchdog timer…) xuống một tần số thấp hơn để dễ sử dụng cho mục đích của mình. Ví dụ như nếu bạn dùng timer để tạo ngắt sẽ có một số tỉ lệ chia như chia 8, 64, 256, 1024. nếu muốn ngắt xảy ra giữa 2 lần lớn như 1 giây chẳng hạn. lúc đó bạn phải dùng cách chia 1024. và chia 8 không đáp ứng được.

Trong vi điều khiển ATmega 328P có 5 lựa chọn prescaler: 1 ,8 ,64 ,256 ,1024.

**Cách tính toán**

VD: Giả sử ta muốn đếm tràn của bộ định thời 1 (Timer/ Counter 1) thời gian là 3s mà prescaler được chọn là 1024.

Ta có, chip Atmega328P trên Arduino chạy ở 16MHz, prescaler = 1024, vì vậy thời gian để TCNT1 tăng lên 1 đơn vị  là 1024/16MHz = 64us, thời gian để T/C1 đếm từ 0 đến 65535 là 4us\*65536 = 4,194204s, mà thời gian chúng ta cần tạo là 3 s , do đó ta cần 3s/64us = 46875 lần đếm. Giá trị ban đầu của TCNT1 = 65536 - 46875 = 18661.

**Các thanh ghi phục vụ ngắt cho bộ định thời:**

**Timer/Counter 0**

**TCNT0 :(Timer/Counter Register**) là thanh ghi lưu giá trị đếm của Timer/Counter

**A black line with numbers and letters

Description automatically generated**

Hình 22: Thanh ghi TCNT0

**OCR0A: (Output Compare Register A) và OCR0B (Output Compare Register B)** là hai thanh ghi so sánh của bộ định thời hai thanh ghi này có thể được thiết lập giá trị để so sánh với thanh ghi lưu giá trị đếm để phục vụ mục đích ngắt

**A black line with letters and numbers

Description automatically generated**

Hình 23: Thanh ghi OCR0A

**A black line with numbers and letters

Description automatically generated**

Hình 24: Thanh ghi OCR0B

**TIFR0:(Interrupt Flag Register)** là thanh ghi cờ ngắt, khi một loại ngắt được kích hoạt thì bit đại diện cho loại ngắt đó sẽ tự động được thiết lập lên 1

**A black line with numbers and a line

Description automatically generated with medium confidence**

Hình 25: Thanh ghi TIFR0

**TIMSK0: (Timer/Counter Interrupt Mask Register)** là thanh ghi mặt nạ ngắt cho bộ định thời. Nếu bạn muốn ngắt theo chế dộ nào thì bạn chỉ cần thiết lập các bit đại diện cho loại ngắt đó lên 1

**A black and white diagram

Description automatically generated with medium confidence**

Hình 26: Thanh ghi TIMSK0

**TCCR0A – Timer/Counter Control Register A và TCCR0B – Timer/Counter Control Register B** là hai thanh ghi điều khiển của Timer/Counter 0 thanh ghi TCCR0B sẽ là thanh ghi chọn chế độ cho bộ định thời, còn thanh ghi TCCR0A và bit thứ 3 của thanh ghi TCCR0B sẽ chọn xung nhịp cho Timer/Counter 0

**A black and white text

Description automatically generated**

Hình 27: Thanh ghi TCCR0A

**A black and white image of a bar

Description automatically generated**

Hình 28: Thanh ghi TCCR0B

**A screenshot of a computer

Description automatically generated**

Bảng 4: Mô tả chọn bit đồng hồ

**A screenshot of a computer

Description automatically generated**

Bảng 5: Chọn chế độ cho bộ định thời 0

**Timer/Counter 1**

**TCNT1 :(Timer/Counter Register)** là thanh ghi lưu giá trị đếm của Timer/Counter 1**A black text on a white background

Description automatically generated**

Hình 29: Thanh ghi TCNT1

**OCR1A: (Output Compare Register A) và OCR1B (Output Compare Register B)** là hai thanh ghi so sánh của bộ định thời hai thanh ghi này có thể được thiết lập giá trị để so sánh với thanh ghi lưu giá trị đếm để phục vụ mục đích ngắt

**A black text on a white background

Description automatically generated**

Hình 30: Thanh ghi OCR1A

**A close-up of a number

Description automatically generated**

Hình 31: Thanh ghi OCR1B

**TIFR1:(Interrupt Flag Register)** là thanh ghi cờ ngắt khi một loại ngắt được kích hoạt thì bit đại diện cho loại ngắt đó sẽ tự động được thiết lập lên 1

**A black line with black text

Description automatically generated**

Hình 32: Thanh ghi TIFR1

**TIMSK1: (Timer/Counter Interrupt Mask Register)** là thanh ghi mặt nạ ngắt cho bộ định thời. Nếu bạn muốn ngắt theo chế dộ nào thì bạn chỉ cần thiết lập các bit đại diện cho loại ngắt đó lên 1

**A black and white math symbol

Description automatically generated with medium confidence**

Hình 33: Thanh ghi TIMSK1

**TCCR1A – Timer/Counter Control Register A và TCCR1B – Timer/Counter Control Register B** là hai thanh ghi điều khiển của Timer/Counter 1 thanh ghi TCCR1B sẽ là thanh ghi chọn chế độ cho bộ định thời, còn thanh ghi TCCR1A và bit thứ 3 và 4 của thanh ghi TCCR1B sẽ chọn xung nhịp cho Timer/Counter 1

A black line with letters and numbers

Description automatically generated

Hình 34: Thanh ghi TCCR1A

**A black and white text

Description automatically generated**

Hình 35: Thanh ghi TCCR1B

**A table with numbers and symbols

Description automatically generated**

Bảng 6: Chọn chế độ của bộ định thời

A screenshot of a computer

Description automatically generated

Bảng 7: Mô tả bit chọn đồng hồ

**Timer/Counter 2**

**TCNT2 :(Timer/Counter Register)** là thanh ghi lưu giá trị đếm của Timer/Counter 2

A black and white text

Description automatically generated

Hình 36: Thanh ghi TCNT2

**OCR2A: (Output Compare Register A) và OCR2B (Output Compare Register B)** là hai thanh ghi so sánh của bộ định thời hai thanh ghi này có thể được thiết lập giá trị để so sánh với thanh ghi lưu giá trị đếm để phục vụ mục đích ngắt

**A black line with numbers and letters

Description automatically generated**

Hình 37: Thanh ghi OCR2A

**A black line with letters and numbers

Description automatically generated**

Hình 38: Thanh ghi OCR2B

**TIFR2:(Interrupt Flag Register)** là thanh ghi cờ ngắt khi một loại ngắt được kích hoạt thì bit đại diện cho loại ngắt đó sẽ tự động được thiết lập lên 1

**A black line with letters and numbers

Description automatically generated**

Hình 39: Thanh ghi TIFR2

**TIMSK2: (Timer/Counter Interrupt Mask Register)** là thanh ghi mặt nạ ngắt cho bộ định thời. Nếu bạn muốn ngắt theo chế độ nào thì bạnchỉ cần thiết lập các bit đại diện cho loại ngắt đó lên 1

**A black line with black text

Description automatically generated**

Hình 40: Hình ảnh thanh ghi TIMSK2

**TCCR2A – (Timer/Counter Control Register A) và TCCR2B – (Timer/Counter Control Register B)** là hai thanh ghi điều khiển của Timer/Counter 2 thanh ghi TCCR2B sẽ là thanh ghi chọn chế độ cho bộ định thời, còn thanh ghi TCCR2A và bit thứ 3 của thanh ghi TCCR2B sẽ chọn xung nhịp cho Timer/Counter 2

**A black text on a white background

Description automatically generated**

Hình 41: Thanh ghi TCCR2A

**A black and white text

Description automatically generated with medium confidence**

Hình 42: Thanh ghi TCCR2B

**A screenshot of a computer

Description automatically generated**

Bảng 8: Bảng mô tả bit chọn đồng hồ

A table with text and numbers

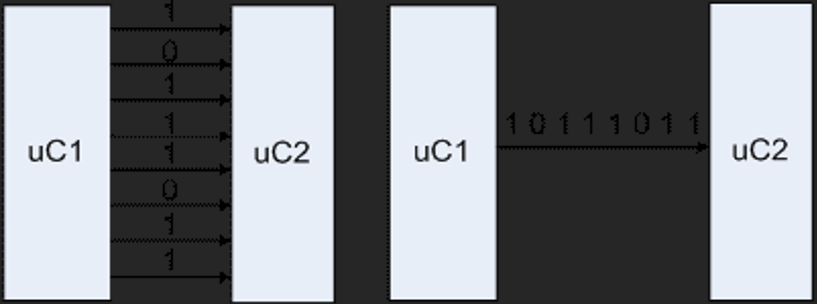
Description automatically generated

Bảng 9: Chọn chế độ của bộ định thời 2

* 1. **Giao tiếp UART**

Thuật ngữ USART trong tiếng anh là viết tắt của cụm từ: Universal Synchronous & Asynchronous serial Reveiver and Transmitter, nghĩa là bộ truyền nhận nối tiếp đồng bộ và không đồng bộ.

Truyền thông nối tiếp: giả sử bạn đang xây dựng một ứng dụng phức tạp cần sử dụng nhiều vi điều khiển (hoặc vi điều khiển và máy tính) kết nối với nhau. Trong quá trình làm việc các vi điều khiển cần trao đổi dữ liệu cho nhau. Giả sử dữ liệu cần trao đổi là các mã có chiều dài 8 bits, bạn có thể sẽ nghĩ đến cách kết nối đơn giản nhất là kết nối 1 PORT (8 bit) của mỗi vi điều khiển với nhau, mỗi line trên PORT sẽ chịu trách nhiệm truyền/nhận 1 bit dữ liệu. Đây gọi là cách giao tiếp song song. Nhược điểm của cách truyền này là số đường truyền quá nhiều. Hệ thống truyền thông song song thường rất cồng kềnh và vì thế kém hiệu quả. Truyền thông nối tiếp sẽ giải quyết vần đề này, trong tuyền thông nối tiếp dữ liệu được truyền từng bit trên 1 (hoặc một ít) đường truyền. Vì lý do này, cho dù dữ liệu của bạn có lớn đến đâu bạn cũng chỉ dùng rất ít đường truyền.



Hình 43: Truyền song song và truyền nối tiếp

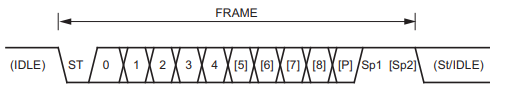
Vì dữ liệu cần được “chia nhỏ” thành từng bit khi truyền/nhận, tốc độ truyền sẽ bị giảm. Mặt khác, để đảm bảo tính chính xác của dữ liệu, bộ truyền và bộ nhận cần có những “thỏa hiệp” hay những tiêu chuẩn nhất định. Phần tiếp theo trong chương này giới thiệu các tiêu chuẩn trong truyền thông nối tiếp không đồng bộ.

Khái niệm “đồng bộ” để chỉ sự “báo trước” trong quá trình truyền. Bằng cách “báo trước” này tất cả các bit dữ liệu có thể truyền/nhận dễ dàng với ít “rủi ro” trong quá trình truyền. Tuy nhiên, cách truyền này đòi hỏi ít nhất 2 đường truyền cho 1 quá trình (send or receive).

Truyền thông “không đồng bộ” chỉ cần một đường truyền cho một quá trình. “Khung dữ liệu” đã được chuẩn hóa bởi các thiết bị nên không cần đường xung nhịp báo trước dữ liệu đến. Ví dụ 2 thiết bị đang giao tiếp với nhau theo phương pháp này, chúng đã được thỏa thuận với nhau rằng cứ 1ms thì sẽ có 1 bit dữ liệu truyền đến, như thế thiết bị nhận chỉ cần kiểm tra và đọc đường truyền mỗi mili-giây để đọc các bit dữ liệu và sau đó kết hợp chúng lại thành dữ liệu có ý nghĩa. Truyền thông nối tiếp không đồng bộ vì thế hiệu quả hơn truyền thông đồng bộ (không cần nhiều lines truyền). Tuy nhiên, để quá trình truyền thành công thì việc tuân thủ các tiêu chuẩn truyền là hết sức quan trọng. Chúng ta sẽ bắt đầu tìm hiểu các khái niệm quan trọng trong phương pháp truyền thông này.

Baud rate (tốc độ Baud): như trong ví dụ trên về việc truyền 1 bit trong 1ms, bạn thấy rằng để việc truyền và nhận không đồng bộ xảy ra thành công thì các thiết bị tham gia phải “thống nhất” nhau về khoảng thời dành cho 1 bit truyền, hay nói cách khác tốc độ truyền phải được cài đặt như nhau trước, tốc độ này gọi là tốc độ Baud. Theo định nghĩa, tốc độ baud là số bit truyền trong 1 giây. Ví dụ nếu tốc độ baud được đặt là 19200 thì thời gian dành cho 1 bit truyền là 1/19200 ~ 52.083us.

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



Hình 44: Hình ảnh khung truyền

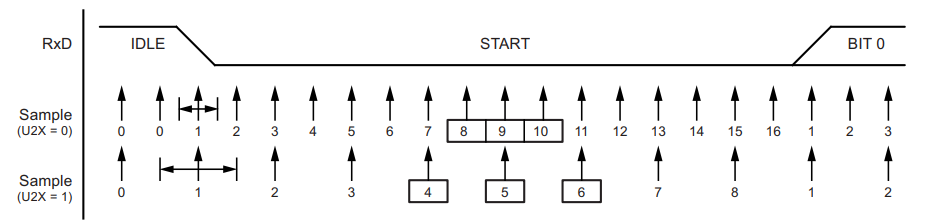
Start bit: là bit đầu tiên được truyền trong một frame truyền, bit này có chức năng báo cho thiết bị nhận biết rằng có một gói dữ liệu sắp được truyền tới. Ở module USART trong AVR, đường truyền luôn ở trạng thái cao khi nghỉ (Idle), nếu một chip AVR muốn thực hiện việc truyền dữ liệu nó sẽ gởi một bit start bằng cách hạ đường truyền xuống mức 0.

Data: data hay dữ liệu cần truyền là thông tin mà chúng ta cần gởi và nhận. Data không nhất thiết phải là gói 8 bit, với AVR bạn có thể quy định số lượng bit của data là 5, 6, 7, 8 hoặc 9. Trong truyền thông nối tiếp UART, bit có ảnh hưởng nhỏ nhất (LSB) của data sẽ được truyền trước và cuối cùng là bit có ảnh hưởng lớn nhất (MSB).

Parity bit: là bit dùng kiểm tra dữ liệu truyền đúng không (một cách tương đối). Có 2 loại là parity chẵn và parity lẻ. Parity chẵn nghĩa là số lượng số 1 trong dữ liệu bao gồm bit parity luôn là số chẵn. Ngược lại tổng số lượng các số 1 trong parity lẻ luôn là số lẻ. Ví dụ, nếu dữ liệu của bạn là 10101011 nhị phân, có tất cả 5 số 1 trong dữ liệu này, nếu parity chẵn được dùng, bit parity sẽ mang giá trị 1 để đảm bảo tổng các số 1 là số chẵn (6 số 1).

Stop bits: là một hoặc các bit báo cho thiết bị nhận rằng một gói dữ liệu đã được gởi xong. Sau khi nhận được stop bits, thiết bị nhận sẽ tiến hành kiểm tra khung truyền để đảm bảo tính chính xác của dữ liệu. Stop bits là các bits bắt buộc xuất hiện trong khung truyền, trong AVR USART có thể là 1 hoặc 2 bit. Giá trị của stop bit luôn là giá trị nghỉ (Idle) và là ngược với giá trị của start bit, giá trị stop bit trong AVR luôn là mức cao (5V).

**Trình tự phát hiện bit bắt đầu:**



Hình 45: Hình ảnh lấy mấu start bit

Khi phát hiện tín hiệu đang ở mức cao (idle) xuống thấp (start bit) , trình tự phát hiện bit bắt đầu.

Tốc độ lấy mấu nhanh hơn tốc độ truyền 16 lần nên một start bit sẽ có 16 mấu ở chế độ thường là 8 mấu ở chế độ gấp đôi tốc độ. bộ phục hồi sẽ dựa vào mấu 8, 9 ,10 ở chế độ thường và mấu 4, 5, 6 ở chế độ gấp đôi tốc độ để quyết định xem có nhận được start bit hợp lệ không. Nếu phần lớn trong 3 bit này là mức cao, strart bit bị từ chối và tiếp tục chờ start bit tiếp theo. Khi phát hiện start bit hợp lệ, quá trình khôi phục dữ liệu có thể bắt đầu.

Phục hồi dữ liệu không đồng bộ:

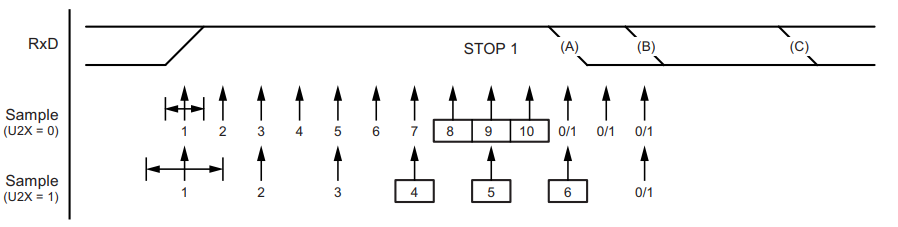
Ảnh có chứa hàng, ảnh chụp màn hình, biểu đồ, văn bản

Mô tả được tạo tự động

Hình 46: Hình ảnh quyết định giá trị mỗi bit dữ liệu

Tương tự với phương pháp phát hiện bit bắt đầu, bộ khôi phục sẽ dựa vào mấu thứ 8, 9, 10 ở chế độ thường và mấu thứ 4, 5, 6 ở chế độ gấp đôi tốc độ để quyết định xem bit đó mang giá trị cao hay thấp

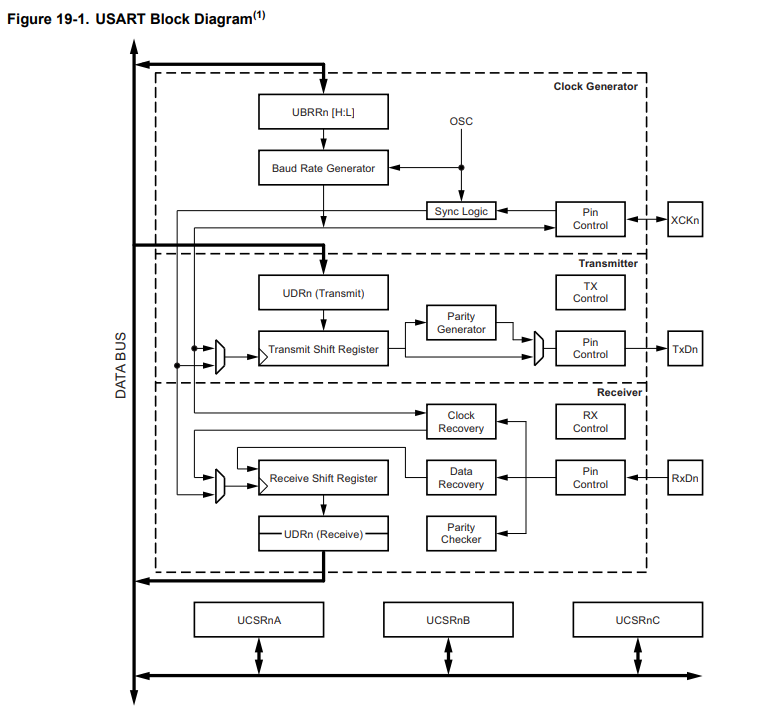
**Phát hiện stop bit:**



Hình 47: Hình ảnh phát hiện stop bit

Việc phát hiện stop bit cũng sé dựa vào mấu thứ 8, 9, 10 ở chế độ thường và mấu thứ 4, 5, 6 ở chế độ gấp đôi tốc độ. Quá trình chuyển đổi từ cao xuống thấp mới cho biết bit bắt đầu của khung mới có thể xuất hiện ngay sau bit cuối cùng được sử dụng cho quá trình phát hiện stop bit. Đối với chế độ tốc độ bình thường, mẫu mức thấp đầu tiên có thể ở điểm được đánh dấu (A) trong hình trên. Đối với chế độ tốc độ gấp đôi mức thấp đầu tiên phải bị trì hoãn đến (B). (C) đánh dấu một bit dừng có độ dài đầy đủ. Việc phát hiện bit bắt đầu sớm ảnh hưởng đến phạm vi hoạt động của máy thu.

**UART trong Atmega328P**

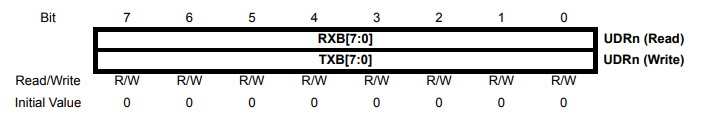


Hình 48: Sơ đồ khối của bộ UART trong ATmega328P

Vi điều khiển ATmega328P có 1 module truyền thông nối tiếp USART. Có 3 chân chính liên quan đến module này đó là chân xung nhịp - XCK (chân đồng bộ), chân truyền dữ liệu – TXD (Transmitted Data) và chân nhận dữ liệu – RXD (Reveived Data) được chia theo các đường nét đứt như trong sơ đồ khối. Trong đó, chân XCK được sử dụng như là chân phát hoặc nhận xung giữ nhịp trong chế độ truyền đồng bộ. Tuy nhiên chúng ta không khảo sát chế độ truyền thông đồng bộ, vì thế bạn chỉ cần quan tâm đến 2 chân TXD và RXD. Vì các chân truyền/nhận dữ liệu chỉ đảm nhiệm 1 chức năng độc lập (hoặc là truyền, hoặc là nhận), để kết nối các chip AVR với nhau (hoặc kết nối AVR với thiết bị hỗ trợ UART khác) bạn phải đấu chéo 2 chân này. TXD của thiết bị này kết nối với RXD của thiết bị kia và ngược lại.

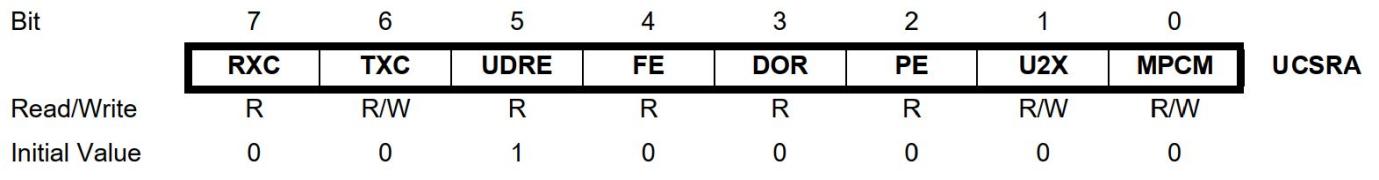
Cũng như các thiết bị khác trên AVR, tất cả hoạt động và tráng thái của module USART được điều khiển và quan sát thông qua các thanh ghi trong vùng nhớ I/O. Có 5 thanh ghi quan trọng cho hoạt động và điều khiển của USART.

UDRn - USART I/O Data Register n: Thanh ghi dữ liệu là 1 thanh ghi 8 bit chứa giá trị nhận được và phát đi của USART. Thực chất thanh ghi này có thể coi như 2 thanh ghi TXB (Transmit data Buffer) và RXB (Reveive data Buffer) có chung địa chỉ I/O. Đọc UDR thu được giá trị thanh ghi đệm dữ liệu nhận, viết giá trị vào UDR tương đương đặt giá trị vào thanh ghi đệm phát, chuẩn bị để gởi đi.



Hình 49: Thanh ghi UDR

UCSRA (USART Control and Status Register A): là 1 trong 3 thanh ghi điều khiển hoạt động của module USART.



Hình 50: Thanh ghi điều khiển trạng thái UCSRA

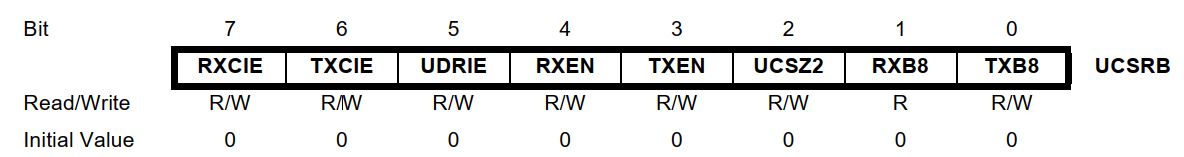
Thanh ghi UCSRA chủ yếu chứa các bit trạng thái như bit báo quá trình nhận kết thúc (RXC), truyền kết thúc (TXC), báo thanh ghi dữ liệu trống (UDRE), khung truyền có lỗi (FE), dữ liệu tràn (DOR), kiểm tra parity có lỗi (PE), bit gấp đôi tốc độ truyền (U2X), bit hoạt động chế độ đa xử lí (MPCM), dưới đây là 1 số bit đáng chú ý trong thanh ghi này

Bit 7 – RXC (USART Receive Complete): bit này được đặt khi có dữ liệu chưa nhận xong và bị xóa khi bộ đệm dữ liệu trống  
Bit 6 – TXC (USART Transmit Complete): Bit này được đặt khi dữ liệu được chuyển ra ngoài và xóa khi dữ liệu truyền đi hết

Bit 5 – UDRE (USART Data Register Empty): khi bit bày bằng 1 nghĩa là thanh ghi dữ liệu UDR đang trống và sẵn sàng cho một nhiệm vụ truyền hay nhận tiếp theo. Vì thế nếu bạn muốn truyền dữ liệu đầu tiên bạn phải kiểm tra xem bit UDRE có bằng 1 hay không, sau khi chắc chắn rằng UDRE=1 hãy viết dữ liệu vào thanh ghi UDR để truyền đi. Nếu bit này không được set thì dữ liệu được ghi vào UDRn sẽ bị UART bỏ qua

Bit 1 – U2X: bit chỉ định gấp đôi tốc độ truyền, khi bit này được set lên 1, tốc độ truyền so cao gấp 2 lần so với khi bit này mang giá trị 0.

UCSRB (USART Control and Status Register B): đây là thanh ghi quan trọng điều khiển USART. Vì thế chúng ta sẽ khảo sát chi tiết từng bit của thanh ghi này.



Hình 51: Thanh ghi điều khiển trạng thái UCSRB

Bit 7 - RXCIE (Receive Complete Interrupt Enable): bit cho phép ngắt khi quá trình nhận kết thúc. Việc nhận dữ liệu truyền bằng phương pháp nối tiếp không đồng bộ thường được thực hiện thông qua ngắt, vì thế bit này thường được set bằng 1 khi USART được dùng để nhận dữ liệu.

Bit 6 - TXCIE (Transmit Complete Interrupt Enable): hoạt động tương tự như bit thứ 7 nhưng bit này cho phép ngắt khi quá trình truyền kết thúc.

Bit 5 - UDRIE (USART Data Register Empty Interrupt Enable): bit cho phép ngắt khi thanh ghi dữ liệu UDR trống.

Bit 4 - RXEN (Receiver Enable): bit quan trọng điều khiển bộ nhận của USART, để kích hoạt chức năng nhận dữ liệu bạn phải set bit này lên 1.

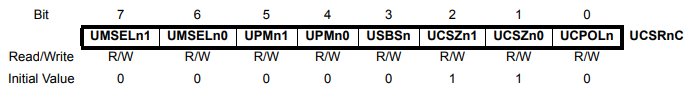
Bit 3 - TXEN (Transmitter Enable): bit điều khiển bộ phát. Set bit này lên 1 bạn sẽ khởi động bộ phát của USART.

Bit 2 - UCSZ2 (Chracter size): bit này kết hợp với 2 bit khác trong thanh ghi UCSRC quy định độ dài của dữ liệu truyền/nhận. Chúng ta sẽ khảo sát chi tiết khi tìm hiểu thanh ghi UCSRC.

Bit 1 - RXB8 (Receive Data Bit 8): gọi là bit dữ liệu 8. Bạn nhớ lại rằng USART trong AVR có hỗ trợ truyền dữ liệu có độ dài tối đa 9 bit, trong khi thanh ghi dữ liệu là thanh ghi 8 bit. Do đó, khi có gói dữ liệu 9 bit được nhận, 8 bit đầu sẽ chứa trong thanh ghi UDR, cần có 1 bit khác đóng vai trò bit thứ chín, RXD8 là bit thứ chín này. Bạn chú ý là các bit được đánh số từ 0, vì thế bit thứ chín sẽ có chỉ số là 8, vì lẽ đó mà bit này có tên là RXD8.

Bit 0 - TXB8 (Transmit Data Bit 8): tương tự như bit RXD8, bit TXB8 cũng đóng vai trò bit thứ 9 truyền thông, nhưng bit này được dùng trong lúc truyền dữ liệu.

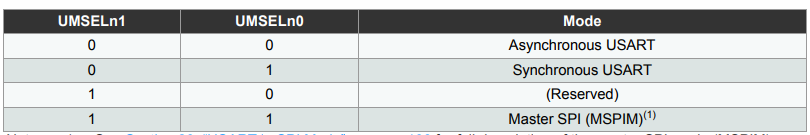
UCSRC (USART Control and Status Register C): thanh ghi này chủ yếu quy định khung truyền và chế độ truyền.



Hình 52: Thanh ghi điều khiển trạng thái UCSRC

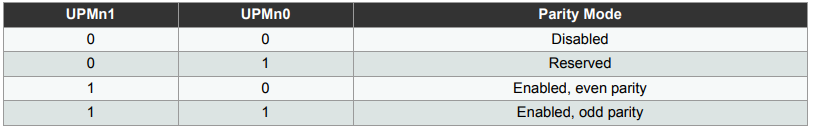
Các bit trong thanh ghi UCSRC được mô tả như sau:

Các bit UMSEL1 và UMSEL0 là các bit lựa chọn chế độ hoạt động của UART. Nếu  UMSEL1=0 và UMSEL0=0, chế độ không đồng bộ được chọn, nếu UMSEL0=1, chế độ đồng bộ được kích hoạt.



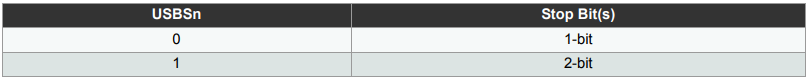
Bảng 10: Thiết lập bit chọn chế độ (Mode select)

Các bit UPM1 và UPM0( Parity Mode) được dùng để quy định kiểm tra pariry. Nếu UPM[1:0]=00, parity không được sử dụng (mode này thường được sử dụng), UPM[1:0]=01 không được sử dụng, UPM[1:0]=10 thì parity chẵn được dùng, UPM[1:0]=11 parity lẻ được sử dụng.



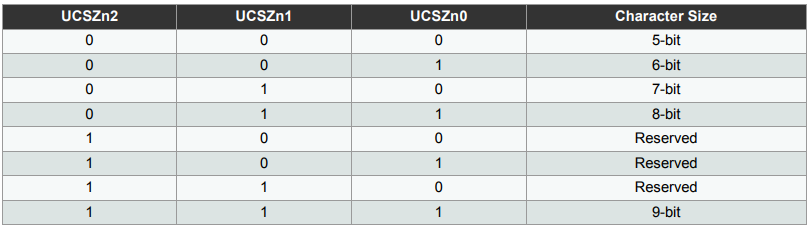
Bảng 11: Thiết lập bit kiểm tra chẵn lẻ (Parity bit)

Bit USBS (Stop bit Select): Bit Stop trong khung truyền bằng AVR USART có thể là 1 hoặc 2 bit, nếu USBS=0 thì Stop bit chỉ là 1 bit trong khi USBS=1 sẽ có 2 Stop bit được dùng.



Bảng 12: Thiết lập số bit stop

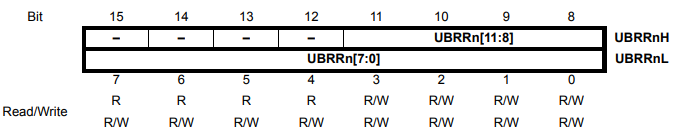
Hai bit UCSZ1 và UCSZ2 (Character Size) kết hợp với bit UCSZ2 trong thanh ghi UCSRB tạo thành 3 bit quy định độ dài dữ liệu truyền, nhận.



Bảng 13: Thiết lập độ dài khung truyền dữ liệu trong UART

UCPOL (Clock Pority) là bit chỉ cực của xung kích trong chế độ truyền thông đồng bộ. Nếu UCPOL=0, dữ liệu sẽ thay đổi thay đổi ở cạnh lên của xung nhịp, nếu UCPOL=1, dữ liệu thay đổi ở cạnh xuống xung nhịp. Nếu bạn sử dụng chế độ truyền thông không đồng bộ, hãy set bit này bằng 0.

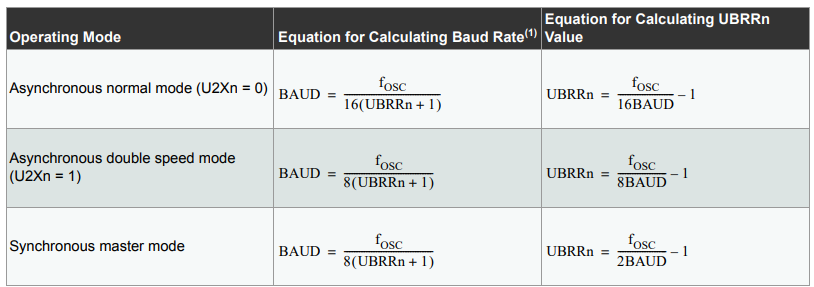
UBRRnL and UBRRnH – USART Baud Rate Registers: 2 thanh ghi quy định tốc độ truyền (baudrate).



Hình 53: Thanh ghi quy định tốc độ baud

Như bạn quan sát trong hình trên, chỉ có 4 bit thấp của UBRRH được dùng, 4 bit này kết hợp với 8 bit trong thanh ghi UBRRL tạo thành thanh ghi 12 bit quy định tốc độ baud. Chú ý là nếu bạn viết giá trị vào thanh ghi UBRRL, tốc độ truyền sẽ tức thì được cập nhật, vì thế bạn phải viết giá trị vào thanh ghi UBRRH trước khi viết vào thanh ghi UBRRL. Giá trị gán cho thanh ghi UBRR không phải là tốc độ baud, nó chỉ được USART dùng để tính tốc độ truyền.

Bảng dưới đây hướng dẫn cách tính tốc độ baud dựa vào giá trị của thanh ghi UBRR và ngược lại, cách tính giá trị cần thiết gán cho thanh ghi UBRR khi đã biết tốc độ baud.

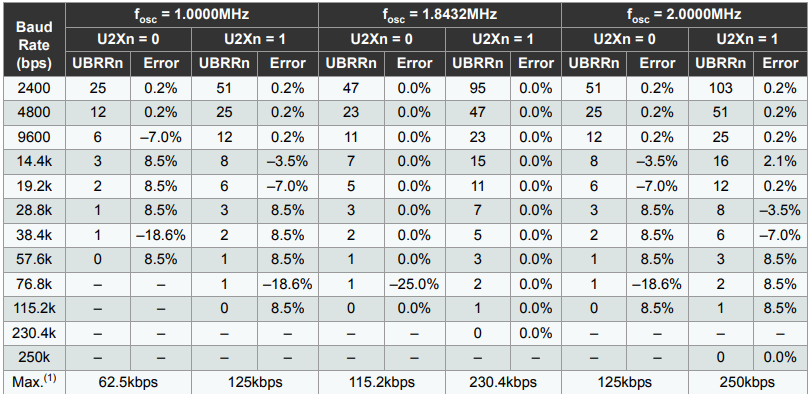
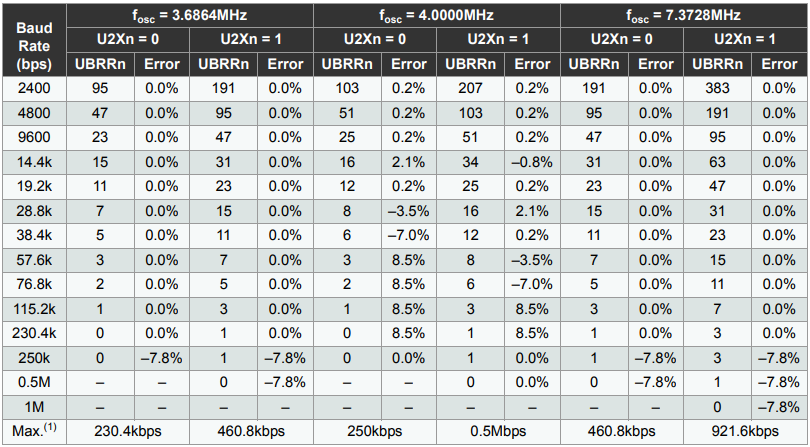
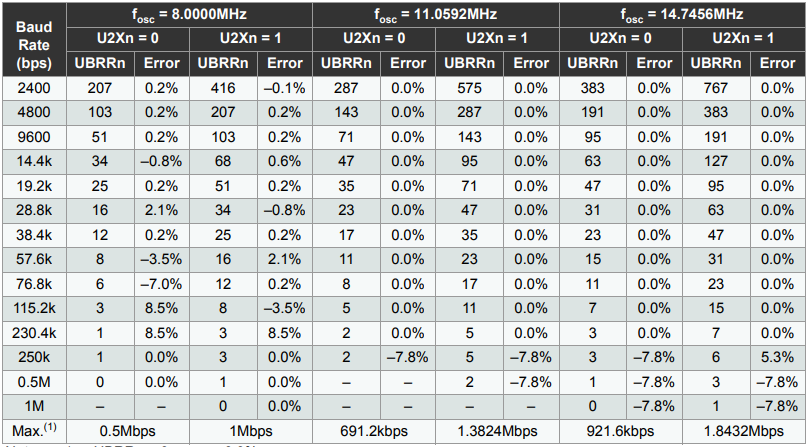


Bảng 14: Tính giá trị cho thanh ghi tốc độ truyền dữ liệu

Trong đó: *fosc* là tần số dao động của thạch anh hoặc tần số dao động của xung nội

* BAUD là tốc độ baud
* UBRR là giá trị được gán vào thanh ghi

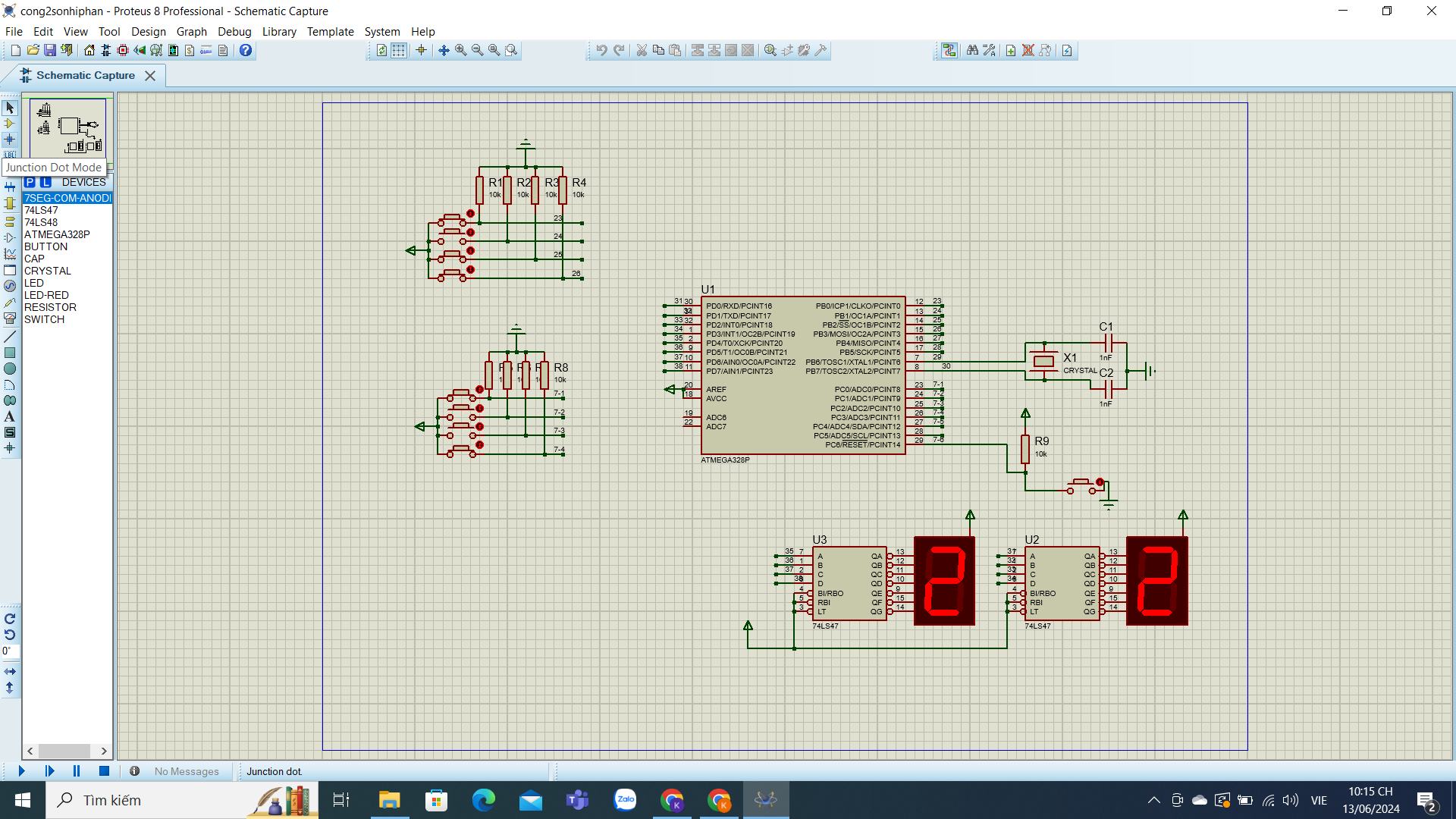
Ta cũng có thể dựa vào bảng sau đây để xác định được các giá trị từ các thông số cho trước



Bảng 15: Một số cài đặt UBRR cho các tần số dao động thường được sử dụng

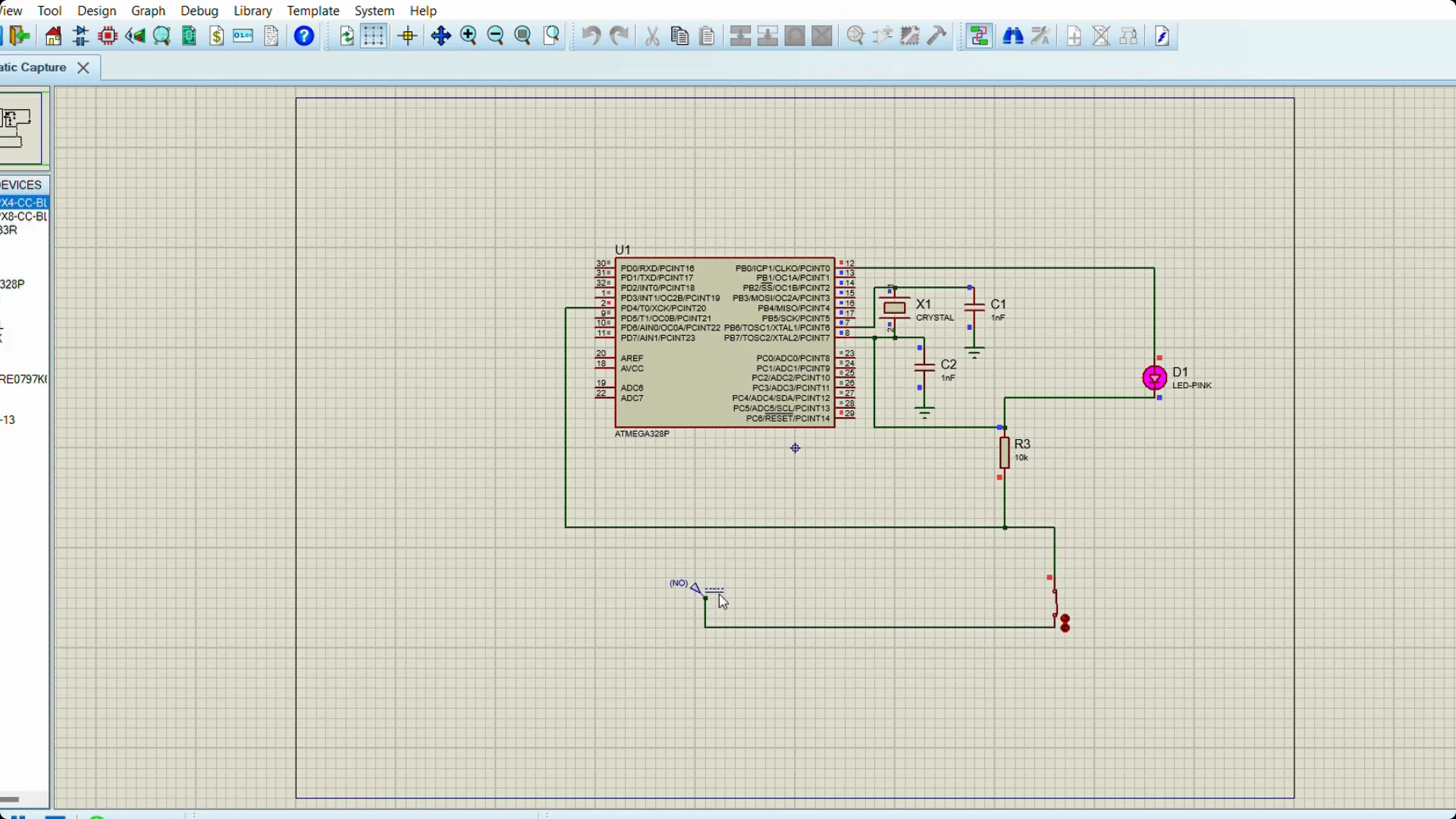
1. **KẾT QUẢ THỰC HIỆN**

## **Hoạt động GPIO**



Hình 43: Mạch cộng 2 số nhị phân 4 bit

Đây là mạch cộng 2 số nhị phân, xuất ra số BCD để cho qua IC 74LS47 chuyển từ số BCD sang led 7 đoạn. Vi điều khiển nhận hai số nhị phân từ Port B và Port C và xuất ra số BCD ở Port D.



Hình 44: Đèn sáng nhấp nháy khi bấm nút

Khi công tắc mở: Chân PD4 clear (bit 0) làm cho chân PB0 cũng ở mức thấp nên led không sáng Khi công tắc đóng: Chân PD4 được set bit lên 1, chân PB0 chuyển trạng thái liên tục (có delay) khiến cho led nhấp nháy

## **Hoạt động ngắt**

Ngắt ngoài



Hình 45: Hình ảnh sơ đồ ứng dụng ngắt ngoài

Sơ đồ nguyên lý ứng dụng ngắt ngoài sử dụng nút nhấn để kích hoạt ngắt INT0 (chân PD2), tác động cạnh xuống. Chương trình chính dịch 7 led từ trên xuống khi nhấn nút sẽ chuyển sang chương trình ngắt và sau khi thực hiện xong chương trình ngắt thì sẽ tiếp tục thực hiện chương trình chính.

Ngắt Timer

A computer screen shot of a computer

Description automatically generated

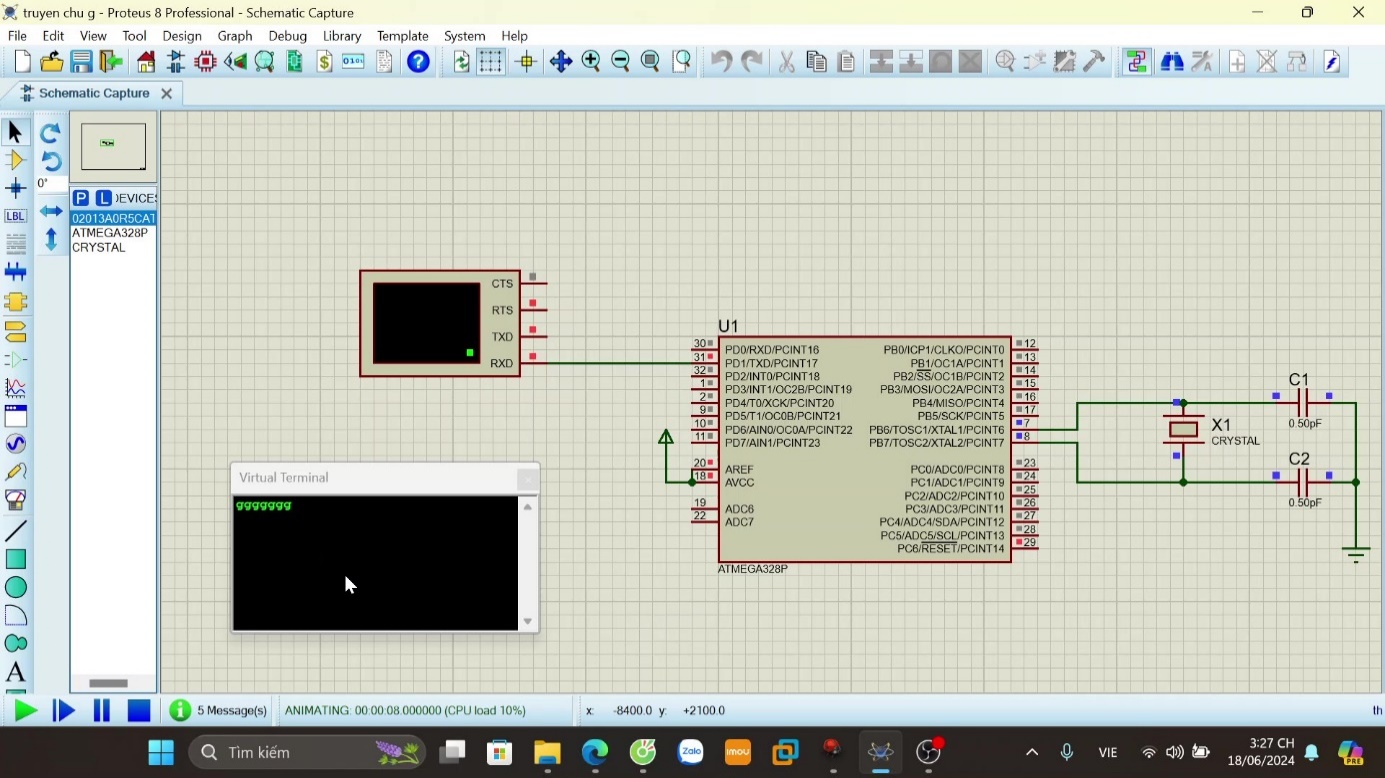
Hình 46: Hình ảnh sơ đồ ngắt Timer

Đây là vi điều khiển Atmega328P sử dụng ngắt timer ở chế độ đếm tràn cứ 3s chương trình chính đang chạy được thể hiện ở bóng đèn D1 đang chớp tắt ở Port PB3 sẽ ngưng hoạt động và chuyển đến chương trình phục vụ ngắt được thể hiện ở bóng đèn D2(sáng) ở Port PD6 và khi chương trình phục vụ ngắt thực hiện xong thì chương trình chính sẽ tiếp tục được thực hiện.



## **Hoạt động giao tiếp truyền thông UART**

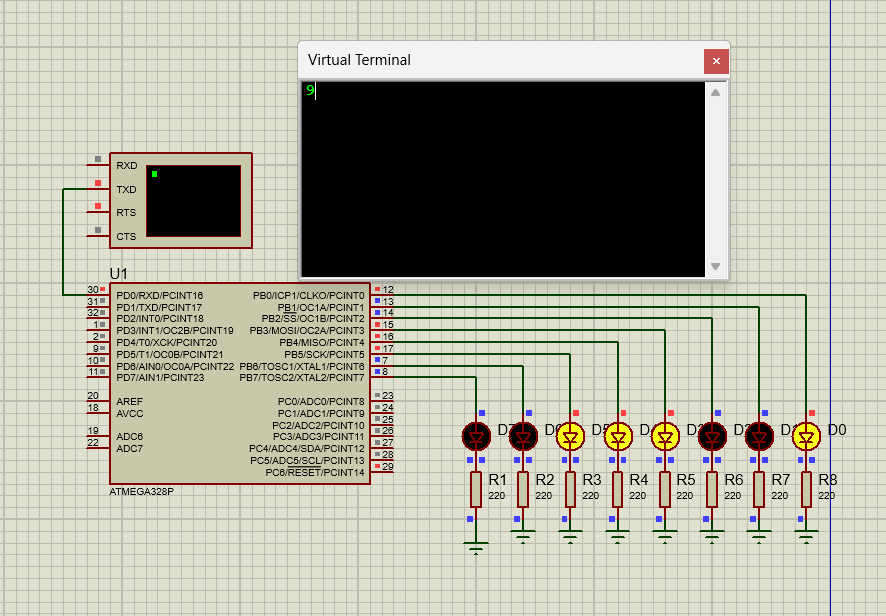
Truyền một ký tự



Hình 46: Hình ảnh sơ đồ mạch truyền dữ liệu bằng UART

Đây là mạch sử dụng truyền thông nối tiếp đồng bộ (UART) bằng ATmega328P để truyền liên tục ký tự `g` với tốc độ baud là 9600 baud/s. Chân truyền của atemega được nối với chân nhận của termial, termial đang cho thấy atemega đang truyền ký tự `g` liên tục.

Nhận dữ liệu trong UART



Hình 47: Mô phỏng nhận dữ liệu UART từ thiết bị đầu cuối ảo

Quá trình nhận dữ liệu chỉ xảy ra khi bit RXEN trong thanh ghi UCSRB được set bằng 1 và chân nhận dữ liệu RXD phải được nối với một nguồn phát (chân TXD của thiết bị ảo). Các thông số truyền thông như tốc độ baud và khung truyền trong bộ nhận phải được cài đặt như của bộ phát. Ta cấu hình trên vi điều khiển các chân của cổng B làm ngõ ra, khi nhận dữ liệu từ thiết bị đầu cuối ảo lúc này là 1 ký tự ascii nhập từ bàn phím, máy thu sẽ xuất ra cổng B và hiển thị trên các led bằng mã nhị phân 8 bit của mã hexa tương ứng với MSB là R1 và LSB là R8. Ví dụ trong hình trên ta nhập ký tự ASCII “9” có mã hexa là “39” thì máy thu sẽ hiển thị trên led chuỗi mã “0011 1001”.

1. **KẾT LUẬN VÀ PHƯƠNG HƯỚNG PHÁT TRIỂN**

Sau một thời gian nghiên cứu tìm hiểu, chúng em đã hoàn thành được đề tài và giải quyết được một vài vấn đề sau:

* Tìm hiểu được cấu trúc của vi điều khiển, kiến trúc tập lệnh, vài hoạt động cơ bản của vi điều khiển ATmega328P
* Lập trình được một vài ví dụ cơ bản của vi điều khiển này

Với đề tài này, trong tương lai chúng em sẽ phát triển ở mức cao hơn, tìm hiểu các chức năng còn lại của vi điều khiển cũng như ứng dụng thực tế của vi điều khiển này trong cuộc sống như hệ thống điều khiển máy móc công nghiệp hay các ứng dụng dựa trên IoT

1. **TÀI LIỆU THAM KHẢO**
2. The AVR Microcontroller and Embedded Systems: Using Assembly and C, Muhammad Ali Mazidi, Sarmad NaimiSepehr Naimi
3. ATmega328P datasheet
4. Học cùng AVR - bài 2: Cấu trúc AVR:

Link: <https://www.hocavr.com/2018/06/bai-2-cau-truc-avr.html>

1. RJH coding, Status Register

Link: <https://www.rjhcoding.com/avr-asm-sreg.php>

1. Arxterra, The Real World of External Interrupts:

Link: https://www.arxterra.com/11-atmega328p-external-interrupts/

1. Học AVR, Bài 3 – Ngắt ngoài:

Link: <https://www.hocavr.com/2018/06/bai-3-ngat-ngoai.html>

1. **Controllerstech, How to receive data via UART using Interrupt**

Link: <https://controllerstech.com/avr-4-how-to-receive-data-via-uart-interrupt/>

1. Mecsu, Bộ vi điều khiển ATmega328P

Link: <https://mecsu.vn/ho-tro-ky-thuat/bo-vi-dieu-khien-atmega328p.ZGO?fbclid=IwZXh0bgNhZW0CMTAAAR1EWk1P6Bo0lg8BkKPCsAMgEpeBim6EE-UA4pmVnd5ekYdCDi76rPcZkuA_aem_ZmFrZWR1bW15MTZieXRlcw>

1. 08 ATmega328P GPIO