TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
Viện Công nghệ thông tin & Truyền thông
----- oOo -----

BÁO CÁO MÔN PROJECT 1
Đề tài:
Tìm hiểu về AVR.

Ngành:	Kĩ thuật máy tính.
Giáo viên hướng dẫn:	TS. Nguyễn Đình Thuận.
Họ và tên sinh viên:	Nguyễn Bá Quân
Mã số sinh viên:	20173319
Lớp, khóa:	KTMT.06 – K62

Hà Nội, ngày 19 tháng 12 năm 2019.

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
Viện Công nghệ thông tin & Truyền thông
----- oOo -----

BÁO CÁO MÔN PROJECT 1
Đề tài:
Tìm hiểu về AVR.

Ngành: Kĩ thuật máy tính.

Giáo viên hướng dẫn: TS. Nguyễn Đình Thuận.
Họ và tên sinh viên: Nguyễn Bá Quân
Mã số sinh viên: 20173319
Lớp, khóa: KTMT.06 – K62

Hà Nội, ngày 19 tháng 12 năm 2019.

Mục lục

Hình ảnh sử dụng.

Chương I: Lý do chọn đề tài.

Thế giới ngày nay với khoa học kĩ thuật phát triển mạnh mẽ, cuộc sống con người ngày càng được phát triển tốt hơn. Góp phần to lớn trong quá trình phát triển của khoa học kĩ thuật là sự phát triển mạnh mẽ của vi xử lý. Từ bộ vi xử lý đầu tiên Intel 4004 thương mại đầu tiên năm 1971, đến nay ngành công nghiệp vi xử lý đã phát triển vượt bậc và đa dạng như: 8951, PIC, AVR, ARM, …

Vì vậy, trong môn project1 này em xin chọn đề tài: "**Tìm hiểu về ARM**" để làm báo cáo môn học.

Chỉ trong một kì học ngắn với vốn kiến thức còn hạn chế nên bài báo cáo chắc chắn còn nhiều thiếu sót, em rất mong nhận được sự chỉ bảo của thầy để hoàn thiện bài báo cáo của mình.

Chương II: Tìm hiểu về đề tài.

1. Giới thiệu về AVR.

AVR là họ vi điều khiển do hãng Atmel sản xuất lần đầu năm 1996. AVR là chip điều khiển 8 bits với tập lệnh đơn giản hóa – RISC (Reduced Instruction Set Computer), một kiểu cấu trúc đang thể hiện ưu thế trong các bộ xử lý.

1.1. Ưu điểm.

So với các chip vi điều khiển 8 bits khác, AVR có nhiều đặc tính hơn hẳn, hơn cả trong tính ứng dụng và đặc biệt là về chức năng:

- Có thể sử dụng xung clock lên đến 16MHz, hoặc sử dụng xung clock nội lên đến 8 MHz (sai số 3%).
- Bộ nhớ chương trình Flash có thể lập trình lại rất nhiều lần và dung lượng lớn, có SRAM (Ram tĩnh) lớn, và đặc biệt có bộ nhớ lưu trữ lập trình được EEPROM.
- Bộ nhớ chương trình Flash có thể lập trình lại rất nhiều lần và dung lượng lớn, có SRAM (Ram tĩnh) lớn, và đặc biệt có bộ nhớ lưu trữ lập trình được EEPROM.
- Nhiều ngõ vào ra (I/O PORT) 2 hướng (bi-directional).
- 8 bits, 16 bits timer/counter tích hợp PWM (Pulse Width Modulation).
- Các bộ chuyển đổi Analog – Digital phân giải 10 bits, nhiều kênh.
- Chức năng Analog comparator.
- Giao diện nối tiếp USART (tương thích chuẩn nối tiếp RS-232).
- Giao diện nối tiếp Two –Wire –Serial (tương thích chuẩn I2C) Master và Slaver.
- Giao diện nối tiếp SPI (Serial Peripheral Interface).
- ...

Gần như không cần mắc thêm bất kì linh kiện nào khi sử dụng AVR, thậm chí không càn nguồn tạo xung clock cho chip (thường là các khối thạch anh).

Thiết bị lập trình (mạch nạp) cho AVR rất đơn giản, có loại mạch nạp chỉ cần vài điện trở là có thể làm được. Một số AVR còn hỗ trợ lập trình on – chip bằng bootloader không cần mạch nạp, …

Cấu trúc AVR được thiết kế tương thích với C.

1.2. Phân loại

AVR thường được phân loại như sau:

- **tinyAVR** – dòng ATtiny:
  - Bộ nhớ chương trình: 0,5 – 32KB.
  - Gồm 6 – 32 pin package.
  - Giới hạn giao tiếp ngoại vi.
  - Chỉ phù hợp cho các ứng dụng đơn giản.
- **megaAVR** – dòng ATmega:
  - Bộ nhớ chương trình: 4 – 256 KB.
  - Gồm 28 – 100 pin package.
  - Mở rộng tập lệnh.
  - Mở rộng bộ nhớ ngoại vi.
  - Thích hợp cho các ứng dụng từ đơn giản đến phức tạp.
- **XMEGA** – dòng ATxmega:
  - Bộ nhớ chương trình: 16 – 284 KB.
  - Gồm 44 – 64 – 100 pin package.
  - Tăng hiệu suất và tính năng, ví dụ như DMA (Direct Memory Access), hỗ trợ mã hóa.
  - Mở rộng tập ngoại vi với ADC.
  - Thích hợp cho các ứng dụng phức tạp, cần bộ nhớ chương trình lớn và tốc độ cao.

Hình 1: Các dòng AVR: tinyAVR, megaAVR, XMEGA AVR

2. Chương trình sử dụng hợp ngữ (ASM - Assembly language).

Để hiểu thấu đáo về AVR, bạn cần lập trình bằng chính ngôn ngữ của nó là hợp ngữ. Như vậy hợp ngữ giúp bạn hiểu tường tận về AVR, và tất nhiên bạn phải hiểu cấu trúc AVR (chúng ta sẽ tìm hiểu ở phần sau).

Sau đây chúng ta sẽ tìm hiểu về tập lệnh của hợp ngữ mà rất hay sử dụng khi lập trình cho AVR.

2.1. Tập lệnh dùng cho Register Files.

- **LDI** (LoaD Immediate).
  - Cú pháp: LDI Rd, K;
  - Chức năng: Load hằng số K vào thanh ghi Rd.

- Giới hạn: chỉ áp dụng cho các thanh gì từ R16 đến R31.

- **MOV** (MOVe).
  - Cú pháp: MOV Rd, Rr;
  - Chức năng: sao chép giá trị trong thanh ghi Rr vào thanh ghi Rd.
  - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.

- **CLR** (Clear Register).
  - Cú pháp: CLR Rd;
  - Chức năng: xóa giá trị trong thanh ghi Rd.
  - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.

- **SER** (Set Register).
  - Cú pháp: SER Rd;
  - Chức năng: set tất cả các bit trong thanh ghi Rd lên 1.
  - Giới hạn: Chỉ áp dụng cho thanh ghi từ R16 đến R31.

- **CBR** (Clear Bit in Register).
  - Cú pháp: CBR Rd, K;
  - Chức năng: xóa các bit trong thanh ghi Rd với "mặt nạ" K, nếu bit nào trong K là 1 thì bit tương ứng trong Rd sẽ bị xóa.
  - Giới hạn: Chỉ áp dụng cho thanh ghi từ R16 đến R31.

- **SBR** (Set Bit in Register).
  - Cú pháp: SBR Rd, K;
  - Chức năng: set các bit trong thanh ghi Rd với "mặt nạ" K, nếu bit trong K là 1 thì bit tương ứng trong Rd sẽ được set lên 1.
  - Giới hạn: Chỉ áp dụng cho thanh ghi từ R16 đến R31.

- **BLD** (Bit LoaD from T Flag).
  - Cú pháp: BLD Rd, b;
  - Chức năng: Load giá trị trong cờ T của thanh ghi SREG của bit thứ b trong thanh ghi Rd. Đây cũng là chức năng chính của cờ T.
  - Giới hạn: Áp dụng cho tất cả các thanh ghi trong RF.

- **BST** (Bit Storage from T Flag).
  - Cú pháp: BST Rd, b;
  - Chức năng: Copy bit thứ b trong thanh ghi Rd vào trong cờ T của thanh ghi SREG.
  - Giới hạn: Áp dụng cho tất cả các thanh ghi trong RF.

- **CPI** (COMPARE with Immediate).
  - Cú pháp: CPI Rd, K;
  - Chức năng: so sánh thanh ghi Rd với hằng số K, lệnh này làm thay đổi nhiều bit trong thanh ghi SREG trong đó sự thay đổi của cờ Zero là quan

trọng nhất, nếu Rd = K cờ Z=1, ngược lại Z=0, sử dụng đặc điểm thay đổi của cờ Z kết hợp với lệnh BRNE hoặc BREQ chúng ta có thể tạo thành một lệnh rẽ nhánh.
    - Giới hạn: chỉ áp dụng cho các thanh ghi từ R16 đến R31.
- **ANDI** (AND with Immediate).
    - Cú pháp: ANDI Rd, K;
    - Chức năng: thực hiện phép Logic AND giữa thanh ghi Rd với hằng số K và kết quả đặt lại trong Rd.
    - Giới hạn: chỉ áp dụng cho các thanh ghi từ R16 đến R31.
- **AND** (Logical AND).
    - Cú pháp: AND Rd, Rr;
    - Chức năng: thực hiện phép Logic AND giữa 2 thanh ghi Rd và Rr, kết quả đặt lại trong Rd.
    - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.
- **ORI** (Logical OR with Immediate).
    - Cú pháp: ORI Rd, K;
    - Chức năng: thực hiện phép Logic OR giữa thanh ghi Rd với hằng số K và kết quả đặt lại trong Rd.
    - Giới hạn: chỉ áp dụng cho các thanh ghi từ R16 đến R31.
- **OR** (Logical OR).
    - Cú pháp: OR Rd, Rr;
    - Chức năng: thực hiện phép Logic OR giữa 2 thanh ghi Rd và Rr , kết quả đặt lại trong Rd.
    - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.
- **LSL** (Logical Shift Left).
    - Cú pháp: LSL Rd;
    - Chức năng: dịch tất thanh ghi Rd sang trái 1 vị trí, Bit 7 (bit lớn nhất) của Rd sẽ được chứa trong cờ nhớ C, bit 0 của Rd bị xóa thành 0. Thực chất LSL tương đương với phép nhân thanh ghi Rd với 2.
    - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.
- **LSR** (Logical Shift Right).
    - Cú pháp: LSR Rd;
    - Chức năng: dịch tất thanh ghi Rd sang phải 1 vị trí, Bit 0 (bit nhỏ nhất) của Rd sẽ được chứa trong cờ nhớ C, bit 7 của Rd bị xóa thành 0. Thực chất LSR tương đương với phép chia thanh ghi Rd cho 2
    - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.
- **ADD** (ADD without Carry).
    - Cú pháp: ADD Rd, Rr;

- Chức năng: thực hiện phép cộng 2 thanh ghi Rd và Rr, kết quả đặt lại trong Rd. Cờ nhớ C không được sử dụng.
- Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.

- **INC** (INCrement).
  - Cú pháp: INC Rd;
  - Chức năng: tăng thanh ghi Rd 1 đơn vị và kết quả đặt lại trong Rd. Lệnh này đặc biệt thích hợp cho các ứng dụng lặp, kết hợp với BREQ hay BRNE có thể tạo thành 1 vòng lặp FOR.
  - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.

- **SUB** (SUBtract without Carry).
  - Cú pháp: SUB Rd, Rr;
  - Chức năng: thực hiện phép trừ 2 thanh ghi Rd - Rr, kết quả đặt lại trong Rd. Cờ nhớ C không được sử dụng.
  - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.

- **SUBI** (SUBtract Immediate).
  - Cú pháp: SUBI Rd, K
  - Chức năng: thực hiện phép trừ thanh ghi Rd với hằng số K, kết quả đặt lại trong Rd.
  - Giới hạn: chỉ áp dụng cho các thanh ghi từ R16 đến R31.

- **DEC** (DECrement).
  - Cú pháp: DEC Rd
  - Chức năng: giảm thanh ghi Rd 1 đơn vị và kết quả đặt lại trong Rd. Lệnh này đặc biệt thích hợp cho các ứng dụng lặp, kết hợp với BREQ hay BRNE có thể tạo thành 1 vòng lặp FOR.
  - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.

- **MUL** (MULtiply unsigned).
  - Cú pháp: MUL Rd, Rr
  - Chức năng: thực hiện phép nhân không dấu 2 thanh ghi 8 bits Rd, Rr, kết quả là 1 số 16 bits đặt trong 2 thanh ghi R1:R0. Chú ý nếu Rd và Rr là các thanh ghi R1 và R0 thì kết quả sau khi tính được sẽ được viết đè lên.
  - Giới hạn: áp dụng cho tất cả các thanh ghi trong RF.

2.2. Tập lệnh cho các thanh ghi I/O.

Bốn lệnh sau đây được thiết kế riêng để truy cập vùng nhớ I/O, các lệnh này sử dụng địa chỉ I/O của thanh ghi trong vùng nhớ này. Vì là thiết kế riêng cho vùng nhớ I/O, bạn không thể sử dụng các thanh ghi này để truy cập RF hay SRAM. Trong các cú pháp của instruction này, khái niệm địa chỉ A là địa chỉ I/O, $0 \leq A \leq 63$, nếu trong ví dụ A=0x00 thì đó là thanh ghi đầu tiên của vùng I/O, không phải là thanh ghi R0.

- **OUT** (OUTput Data).

- Cú pháp: OUT A, Rr;
- Chức năng: xuất giá trị từ thanh ghi Rr ra thanh ghi có địa chỉ A trong vùng nhớ I/O. đây là cách phổ biến nhất để xuất giá trị ra vùng I/O.
- Giới hạn: Rr là thanh ghi RF bất kỳ, A bị giới hạn từ 0 đến 63.

- **IN** (INPUT Data).
  - Cú pháp: IN Rr, A;
  - Chức năng: Load giá trị từ thanh ghi có địa chỉ A trong vùng nhớ I/O vào thanh ghi Rr. Đây là cách phổ biến nhất để nhận giá trị từ vùng I/O.
  - Giới hạn: Rr là thanh ghi RF bất kỳ, A bị giới hạn từ 0 đến 63.

- **SBI** (Set Bit in I/O Register).
  - Cú pháp: SBI A, b;
  - Chức năng: Set bit thứ b trong thanh ghi có địa chỉ A trong vùng nhớ I/O. Tuy nhiên lệnh này không có tác dụng trên toàn bộ vùng I/O mà chỉ có tác đối với 32 thanh ghi đầu (địa chỉ từ 0 đến 31).
  - Giới hạn: b là số thứ các bit trong thanh ghi, $0 \leq b \leq 7$; A bị giới hạn từ 0 đến 31.

- **CBI** (Clear Bit in I/O Register).
  - Cú pháp: CBI A, b
  - Chức năng: xóa bit thứ b trong thanh ghi có địa chỉ A trong vùng nhớ I/O. Tuy nhiên lệnh này không có tác dụng trên toàn bộ vùng I/O mà chỉ có tác đối với 32 thanh ghi đầu (địa chỉ từ 0 đến 31).
  - Giới hạn: b là số thứ các bit trong thanh ghi, $0 \leq b \leq 7$; A bị giới hạn từ 0 đến 31.

2.3. Các con trỏ X, Y, Z và cách truy cập toàn bộ không gian bộ nhớ.

Trong Register File của AVR, các thanh ghi từ R26 đến R31ngoài chứa năng thanh ghi thông thường còn có chức năng là con trỏ (Pointer) trong việc truy cập bộ nhớ (cả bộ nhớ data và bộ nhớ Program). Nếu được sử dụng như các Pointer, các thanh ghi trên được biết đến với tên gọi X, Y, Z. Định nghĩa như sau: X=R27:R26, Y=R29:R28, Z=R31:R30. Chúng là 3 thanh ghi 16 bits được định nghĩa trước cho tất cả các AVR. Ngoài ra trong các file định nghĩa cho chip chúng ta có thêm 6 định nghĩa khác là XL, XH, YL, YH, ZL, ZH cũng chính là tên gọi của R26-> R31. Phần này chúng ta khảo sát một số instruction dùng truy cập toàn bộ khối nhớ của AVR bằng cách sử dụng địa chỉ trực tiếp và bằng cách sử dụng Pointer.

- **LDS** (LoaD direct from data Space).
  - Cú pháp: LDS Rd, k;
  - Chức năng: load giá trị 1 byte từ thanh ghi có địa chỉ k trong SRAM vào thanh ghi Rd, k là dạng địa chỉ tuyệt đối có giới hạn từ 0 đến $(2^{16} - 1)$.
  - Giới hạn: Rd là thanh ghi bất kỳ trong RF nhưng giá trị lớn nhất của k là 65535, vì thế với lệnh này ta không thể truy cập vượt quá khoảng không gian 64KB. Nếu muốn truy cập vùng không gian lớn hơn 64KB chúng ta

cần một số hỗ trợ, tuy nhiên ở đây tôi giả sử bộ nhớ của chip (thường là bộ nhớ data) không vượt quá 64KB (thực tế chưa có chip AVR nào có SRAM hay EEPROM vượt quá 64KB).

- **STS** (STorage direct to data Space).
  - Cú pháp: STS k, Rr;
  - Chức năng: instruction này hoàn toàn giống LDS nhưng dùng để xuất dữ liệu từ thanh ghi Rr ra RAM, ngươi đọc có thể tham khảo phần giải thích cho LDS.

Một cách khác được dùng để truy cập bộ nhớ mà không dùng địa chỉ tuyệt đối là sử dụng sử dụng con trỏ. Có 2 instruction hỗ trợ con trỏ là LD LoaD indirec from data Space), và ST (STorage indirec to data Space), LD đọc dữ liệu từ SRAM vào thanh ghi còn ST lưu dữ liệu từ thanh ghi vào SRAM. Cả 3 con trỏ X, Y và Z đều có thể được dùng nhưng có một số điểm lưu ý: cả 3 đều dùng được trong trường hợp truy xuất thông thường nhưng với cách truy cập có offset, con trỏ X không sử dụng được. Để truy xuất bộ nhớ chương trình bằng con trỏ thì Z là giải pháp duy nhất…

2.4. Rẽ nhánh và vòng lặp.

Không giống như các ngôn ngữ cấp cao, khi lập trình bằng ASM bạn không được hỗ trợ các cấu trúc điều khiển như If, For, While…người lập trình ASM phải tự xây dựng cho mình các cấu trúc này từ những instruction cơ bản. Nếu bạn có trong tay tài liệu tra cứu instruction cho AVR bạn sẽ thấy có rất nhiều instruction có dạng BRxx, với BR là viết tắt của từ Branch (rẽ nhánh). Đây là các instruction cơ bản giúp bạn xây dựng các cấu trúc điều khiển tương đương If, For, While…cho riêng mình.

- **BREQ** (BRanch if EQual).
  - Cú pháp: BREQ LABEL;
  - Chức năng: Nhảy đến nhãn LABEL nếu cờ Z =1. Cờ Z chịu tác động của rất nhiều instruction như CP, CPI, SUB, SUBI…vì thế BREQ thường được sử dụng sau các instruction này.
- **BRLO** (BRanch if LOwer).
  - Cú pháp: BRLO LABEL;
  - Chức năng: bản chất của câu lệnh là nhảy đến nhãn LABEL nếu cờ C =1. Tuy nhiên, thông thường lệnh này sử dụng theo sau các instruction như CP, CPI, SUB, SUBI…khi đó việc rẽ nhánh sẽ xảy ra nếu thanh ghi Rd.
- **BRSH** (BRanch if Same or Higher).
  - Cú pháp: BRSH LABEL
  - Chức năng: bản chất của câu lệnh là nhảy đến nhãn LABEL nếu cờ C =0. Tuy nhiên, thông thường lệnh này sử dụng theo sau các instruction như CP, CPI, SUB, SUBI…khi đó việc rẽ nhánh sẽ xảy ra nếu thanh ghi Rd ≥Rr.

2.5. Một số Directive.

Một số từ bắt đầu bằng dấu chấm "." là các Directive (ví dụ .INCLUDE hay .ORG) là từ khóa mặc định của ASM AVR.

Chức năng: các directive không phải là mã lệnh mà chỉ là các chỉ dẫn về bộ nhớ, khởi động bộ nhớ, định nghĩa macro, … và không được dịch thành mã.

Dưới đây là bảng tóm tắt các directive và chức năng của chúng.

Directive	Description
BYTE	Reserve byte to a avarible
CSEG	Code Segment
CSEGSIZE	Program memory size
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data segment
DW	Define constant word(s)
ENDM, ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn Macro expansion in list file on
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

2.5. Cấu trúc cơ bản của chương trình ARM.

Có thể chia một đoạn code bằng hợp ngữ thành 4 phần:

- Phần 1: chứa các Directive và lệnh RJUMP dùng để xác định các địa chỉ bộ nhớ chương trình.
    - Chỉ thị .CSEG: báo cho trình biên dịch rằng phần code theo sau là phần chương trình thực thi, phần này sẽ được tải vào bộ nhớ chương trình của chip.
    - Chỉ thị .INCLUDE: báo cho trình biên dịch bắt đầu đọc 1 tệp đính kèm.
    - Chỉ thị .ORG: set vị trí trong bộ nhớ sẽ được tác động đến.
    - RJMP: nhảy không điều kiện đến 1 vị trí trong bộ nhớ.
- Phần 2: khởi tạo điều kiện đầu cho Stack.
- Phần 3: chương trình chính.
- Phần 4: chương trình con.

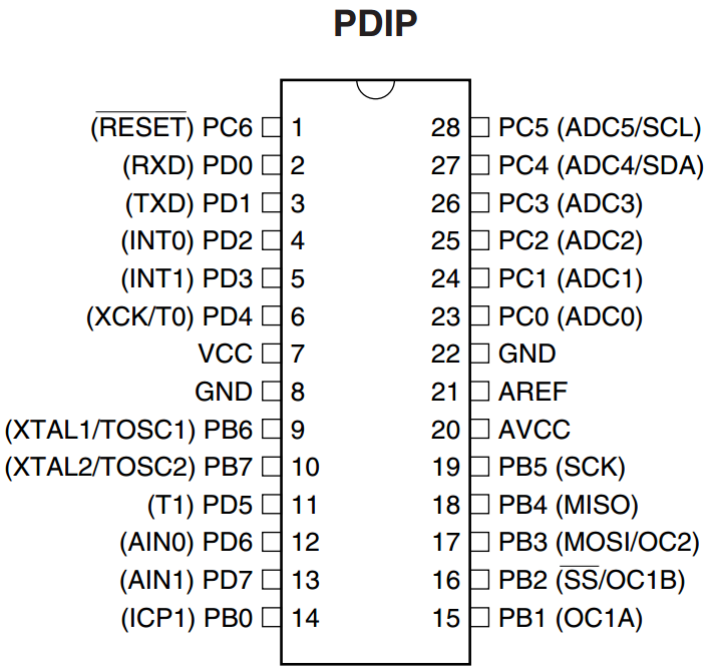
Chú ý: Đây chỉ là một cách để các bạn tiếp cận dễ hơn cách bố trí của một chương trình viết bằng hợp ngữ. Một khi đã quen thuộc, bạn có thể bố trí chương trình theo cách của bạn.

3. Tìm hiểu về ATmega8.

ATmega là một vi điều kiển 8 bits CMOS công suất thấp dựa trên kiến trúc AVR RISC. Bằng cách thực hiện mạnh mẽ các tập lệnh trong một chu kì xung nhịp, ATmega8 đạt được 1 MIPS trêm MHz, cho phép nhà thiết kế tối ưu hóa mức tiêu thụ năng lượng so với tốc độ xử lý.

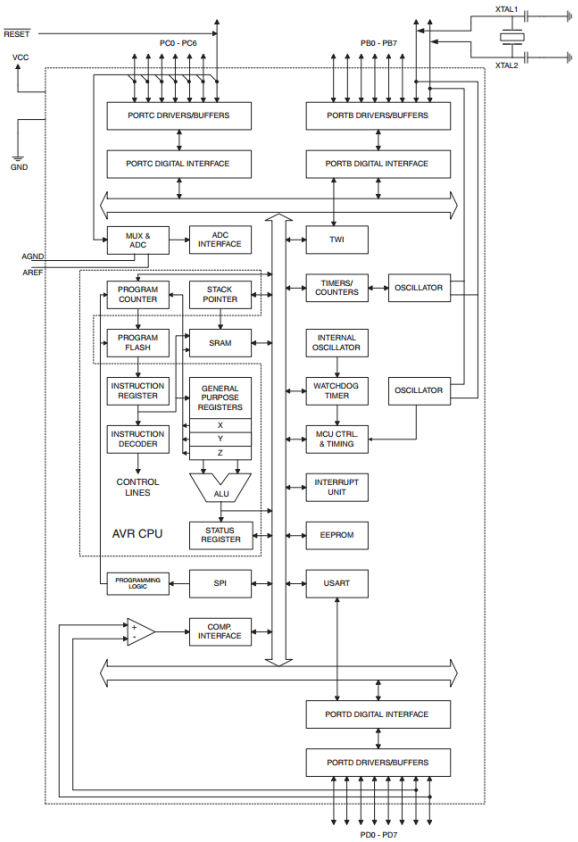
Hình 2: Hình ảnh thực tế ATmega8

3.1. Cấu hình chân (Pin configurations)



Hình 3: Cấu trúc chân của ATmega8

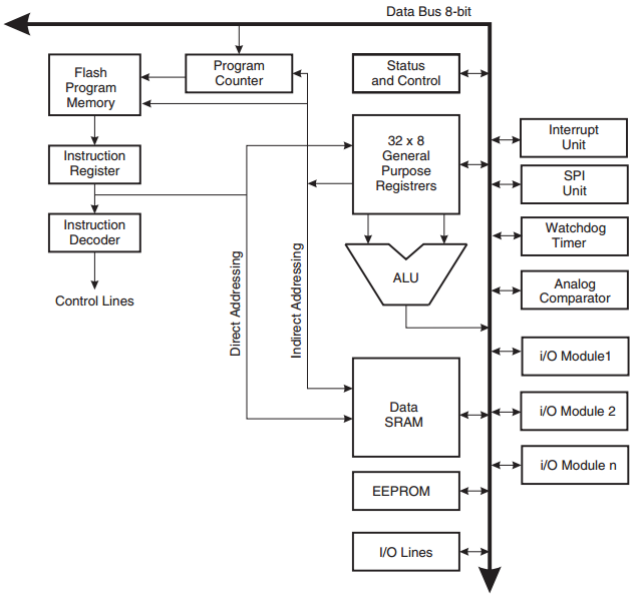
3.2. Sơ đồ khối.



Hình 4: Sơ đồ khối của ATmega8

AVR kết hợp một bộ tập lệnh với 32 thanh ghi (GPR – General Purpose Register) làm việc đa năng. Tất cả 32 thanh ghi được kết nối trực tiếp với ALU (Arithmetic Logic Unit), cho phép 2 thanh ghi độc lập được truy cập trong một lệnh duy nhất và thực hiện trong một chu kì xung nhịp. Kết quả đạt được của kiến trúc này là rất hiệu quả, trong khi đạt được thông lượng nhanh hơn gấp 10 lần so với bộ vi điều khiển CISC thông thường.

3.3. Kiến trúc tổng quan.



Hình 5: Sơ đồ cấu trúc CPU của ATmega8

AVR sử dụng kiến trúc Harvard, trong đó đường truyền cho bộ nhớ dữ liệu (Data memory bus) và đường truyền cho bộ nhớ chương trình (program memory bus) được tách riêng. Data memory bus chỉ có 8 bits và được kết nối với hầu hết các thiết bị ngoại vi, với register file. Trong khi đó program memory bus có độ rộng 16 bits và chỉ phục vụ cho instruction registers.

3.4. Đặc tính.

Được chế tạo theo kiến trúc RISC:

- Tập lệnh gồm 130 lệnh, hầu hết thực hiện trong một chu kì xung nhịp.
- 32 x 8 thanh ghi làm việc đa dụng.
- Tốc độ xử lý lệnh 16MIPS ở 16MHz.
- Vận hành tĩnh hoàn toàn.
- On – chip 2 – circle mutiplier

High Endurance Non-volatile Memory segments:

- Bộ nhớ lập trình 8Kbytes trong hệ thống.
- EEPROM 512Bytes.
- SRAM nội 1Kbyte
- Chu kì đọc ghi: 10000 Flash/100000 EEPROM.
- Lưu trữ dữ liệu: 20 năm ở 85oC/100 năm ở 25oC.
- Mã khởi động tùy chọn với bits khóa độc lập
- Lập trình trong hệ thống bằng chương trình khởi động trên chip
- Thực hiện đọc và ghi đúng

Đặc điểm ngoại vi:

- 2 bộ Timer/Counters 8 bits với bộ chia tần số dao động riêng biệt, một chế độ so sánh.
- 1 bộ Timer/Counter 16 bits với bộ chia tần số dao động riêng, chế độ so sánh và chế độ đếm sự kiện.
- Bộ đếm thời gian thực với bộ dao động riêng.
- 3 kênh tạo xung điều rộng (PMW – Pulse Width Modulation).
- 8 kênh ADC (Analog to Digital Converter) trong gói TQFP (Thin Quad Flat Package) và QFN/MLF (Quad Flat No – lead / Micro Lead Frame), 8 kênh với độ chính xác 10 bits.
- 6 kênh ADC trong gói PDIP (Program Development Increment Package), 6 kênh với độ chính xác 10 bits.
- Truyền thông nối tiếp đồng bộ TWI (Two-wire Serial Interface).
- Truyền nhận nối tiếp đồng bộ và không đồng bộ USART (Universal Synchronous & Asynchronous serial Reveiver and Transmitter).
- Truyền thông nối tiếp SPI (Serial Peripheral Bus).
- Bộ định thời Watdog lập trình được. Tự động reset khi treo máy.
- Bộ so sánh tín hiệu tương tự.

Tính năng đặc biệt của vi xử lý:

- Thiết lập lại nguồn và phát hiện nguồn chip bị suy giảm.
- Dao động RC hiệu chuẩn nội bộ.
- Ngắt ngoài và ngắt trong.
- 5 chế độ ngủ: chế độ rỗi - Idle, chế độ ADC Noise Reduction, chế độ tiết kiệm điện - Power-save, chế độ Power-down, và chế độ chờ - Standby.

I/O và các gói:

- 23 ngõ I/O lập trình được.
- 28-lead PDIP, 32-lead TQFP, and 32-pad QFN/MLF.

Điện áp hoạt động:

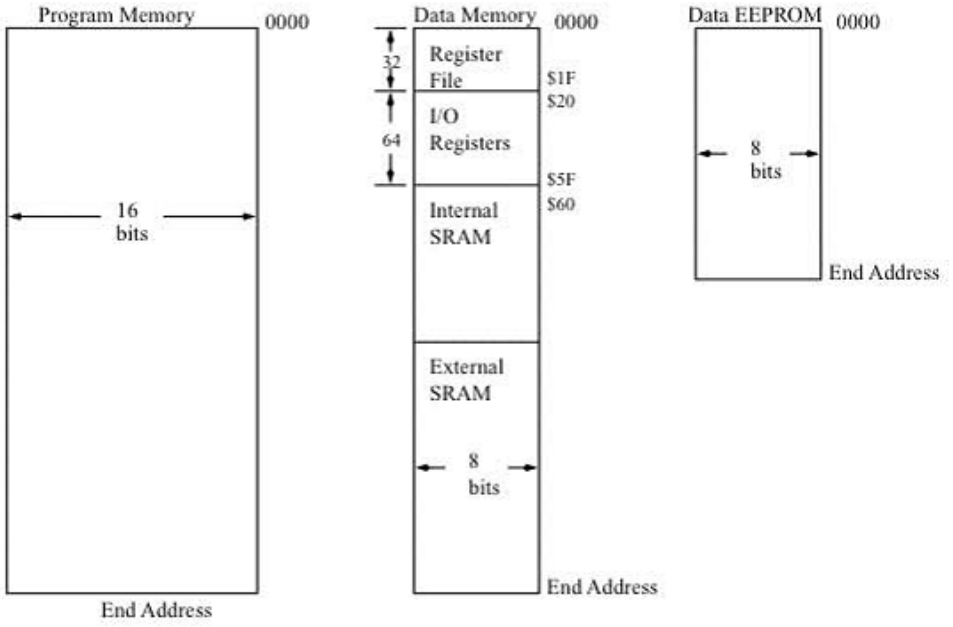
- 4,5 – 5,5V.

Tần số hoạt động:

- 0 - 16MHz.

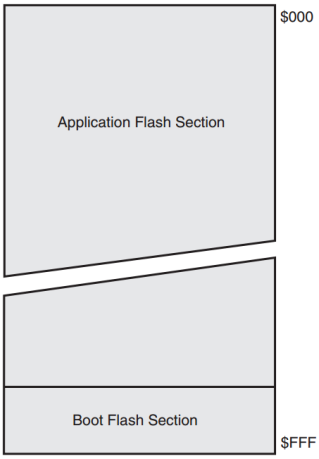
3.5. Bộ nhớ.

Kiến trúc của AVR gồm có 2 không gian bộ nhớ chính, đó là bộ nhớ chương trình (the Program Memory) và bộ nhớ dữ liệu (the Data Memory). Bên cạnh đó, ATmega8 có thêm EEPRROM để lưu trữ dữ liệu.



Hình 6: Tổ chức bộ nhớ AVR.

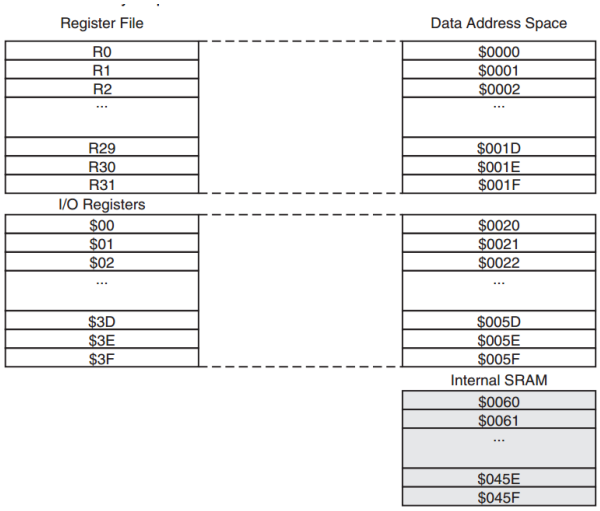
3.5.1. Bộ nhớ chương trình.



Hình 7: Sơ đồ bộ nhớ chương trình của ATmega8.

ATmega8 chứa 8Kbytes bộ nhớ Flash có thể lập trình lại nhiều lần cho lưu trữ chương trình. Do tất cả tập lệnh của AVR có độ rộng 16-bits hoặc 32-bits, nên bộ nhớ này được tổ chức thành 4K x 16 bits. Vì mục đích bảo mật phần mềm, không gian bộ nhớ chương trình được chia thành 2 phần: Boot Program section and Application Program section. Thực chất, Application section gồm 2 phần: phần chứa các instruction (mã lệnh cho hoạt đông của chip) và phần chứa vector ngắt (interrupt vectors).

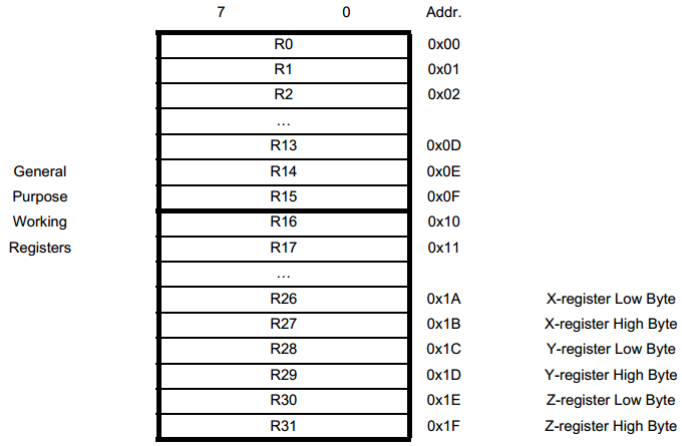
3.5.2. Bộ nhớ dữ liệu.



Hình 8:Sơ đồ bộ nhớ dữ liệu của ATmega8

Đây là phần chứa thanh ghi quan trọng nhất của chip, việc lập trình cho chip phần lớn là truy cập ở bộ nhớ này. Cơ bản được chia thành 5 phần:

- Phần 1:
    - Phần đầu tiên trong bộ nhớ dữ liệu gồm 32 thanh ghi có tên gọi register file (RF) hay General Purpose Register (GPR). Tất cả thanh ghi này đều có độ rộng 8 bits.



Hình 9:Tổ chức thanh ghi dữ liệu chung của ATmega8.

- Chúng được chia thành 2 phần, phần 1 bao gồm các thanh ghi từ R0 đến R15 và phần 2 là thanh ghi từ R16 đến R31. Chúng đều có đặc điểm:
- Được truy cập trực tiếp trong các instruction.
- Các toán tử, phép toán thực hiện trên các thanh ghi này chỉ cần 1 chu kỳ xung clock.
- Register File được kết nối trực tiếp với bộ xử lí trung tâm – CPU của chip.
- Chúng là nguồn chứa các số hạng trong các phép toán và cũng là đích chứa kết quả trả lại của phép toán.

- Phần 2:
  - Nằm ngay sau register file, phần này gồm 64 thanh ghi hay được gọi là 64 thanh ghi nhập/xuất hay còn gọi là vùng nhớ I/O.
  - Là cửa ngõ giao tiếp giữa CPU và thiết bị ngoại vi.
  - Tất cả thanh ghi điều khiển, trạng thái, … đều nằm ở đây.
  - Chú ý: vùng nhớ I/O có thể được truy cập như SRAM hay như các thiết bị I/O. Nếu sử dụng instruction truy xuất SRAM để truy xuất vùng nhớ này thì địa chỉ của chúng được tính từ 0x0020 đến 0x005F. Nhưng nếu truy xuất như các thanh ghi I/O thì địa chỉ của chúng đực tính từ 0x0000 đến 0x003F. Vì các thanh ghi trong vùng I/O không được hiểu theo tên như Register File, ta nên đính kèm 1 file Definition để dễ dàng trong việc gọi tên thanh ghi. Đối với ATmega8 thì ta sử dụng INCLUDE "M8DEF.INC".
  - Chứa thanh ghi quan trọng nhất của AVR, là thanh ghi SREG chứa 8 bit cờ (Flag) chỉ trạng thái của bộ xử lý, tất cả các bit này đều bị xóa sau khi reset, các bit này cũng có thể được đọc và ghi bởi chương trình. Chức năng từng bit được mô tả chi tiết như sau:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 0 – C (Carry Flag: Cờ nhớ)**:** là bit nhớ trong các phép đại số hoặc logic.
- Bit 1 – Z (Zero Flag: Cờ 0): cờ này được set nếu kết quả phép toán đại số hay phép Logic bằng 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.
- Bit 3 – V (Two's complement Overflow Flag: Cờ tràn của bù 2)
- Bit 4 – S (Sign Bit: Bit dấu)**:** Bit S là kết quả phép XOR giữa 1 cờ N và V, S=N xor V.
- Bit 5 – H (Half Carry Flag)**:** cờ H là cờ nhớ trong 1 vài phép toán đại số và phép Logic, 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).
- Bit 7 – I (Global Interrupt Enable): Cho phép ngắt toàn bộ): 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.
- Chú ý: tất cả các bit trong thanh ghi này đều có thể xóa thông qua các instruction không toán hạng CLx và set bởi Sex, trong đó x là tên của Bit.

- Phần 3:
  - Kế tiếp là 1024bytes RAM tĩnh, nội.
  - Là vùng không gian cho các biến (biến tạm thời hoặc toàn cục) trong luc thực thi chương trình.

- Phần 4:
  - RAM ngoại (external SRAM), chip cho phép người sử dụng gắn thêm bộ nhớ ngoài để chứa biến.
  - Vùng này chỉ tồn tại khi người sử dụng gắn thêm bộ nhớ vào chip.

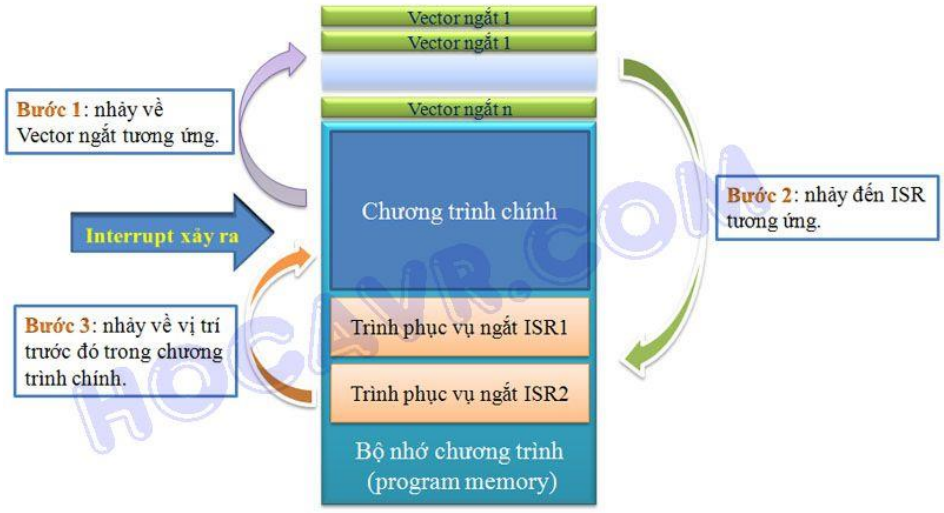
- Phần 5:
  - EEPROM (Electrically Ereasable Programmable ROM) là một phần quan trọng của các chip AVR mới.
  - Vì là ROM nên bộ nhớ này không bị xóa ngay cả khi không cung cấp nguồn nuôi cho chip, rất thích hợp cho các ứng dụng lưu trữ dữ liệu. Như trong hình 1, phần bộ nhớ EEPROM được tách riêng và có địa chỉ tính từ 0x0000.

3.6. Ngắt ngoài trên ATmega8.

3.6.1. Giới thiệu ngắt trên AVR.

Interrupt, thường được gọi là ngắt, là một tín hiệu khẩn cấp gửi đến bộ xử lý, yêu cầu bộ xử lí tạm ngừng tức khắc các hoạt động hiện tại để "nhảy" đến một nơi khác thực hiện một nhiệm vụ khẩn cấp nào đó, nhiệm vụ này gọi là trình phục vụ ngắt – ISR (Interrupt Service Routine). Sau khi kết thúc nhiệm vụ trong ISR, bộ đếm chương trình sẽ được trả về giá trị trước đó để bộ xử lí quay về thực hiện tiếp các nhiệm vụ còn dang dở.

Ngắt có mức độ ưu tiên xử lý cao nhất, ngắt thường được xử lý các sự kiện bất ngờ nhưng không tốn quá nhiều thời gian. Các tín hiệu dẫn đến ngắt có thể xuất phát từ các thiết bị bên trong chip (ngắt báo bộ đếm timer/counter tràn, ngắt báo quá trình gởi dữ liệu bằng RS232 kết thúc…) hay do các tác nhân bên ngoài (ngắt báo có 1 button được nhấn, ngắt báo có 1 gói dữ liệu đã được nhận…).



Hình 10: Tổ chức ngắt trên AVR.

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

Hình 11: Các vector ngắt và reset trên ATmega8.

3.6.2. Ngắt ngoài (External Interrupt) trên ATmega8.

Là loại ngắt duy nhất động lập với thiết bị của chip, hiệu quả trong việc giao tiếp người dùng và chip.

Trên ATmega8 có 2 ngắt ngoài có tên INT0 và INT1 tương ứng với 2 chân số 4 (PD2) và chân số 5 (PD3).

Có 3 thanh ghi liên quan đến ngắt ngoài đó là MCUCR, GICR và GIFR. Ta sẽ tìm hiểu cụ thể sau đây:

- Thanh ghi điều khiển MCU – MCUCR (MCU Control Register) là thanh ghi xác lập chế độ ngắt cho ngắt ngoài.

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

  - Đối với ngắt ngoài, ta chỉ cần quan tâm đến 4 bits thấp của nó dùng để thiết lập 4 modes khi sử dụng ngắt. Dưới đây là bảng "chân trị của 2 bits ISC11, ISC10; bảng cho ISC01, ISC00 hoàn toàn tương tự.

ISC11	ISC10	Mô tả
0	0	Mức thấp của chân INT1 tạo ra 1 yêu cầu ngắt – ngắt mức thấp.
0	1	Bất kì thay đổi nào trên chân INT1 tạo ra 1 yêu cầu ngắt.
1	0	Cạnh xuống trên chân INT1 tọa 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).

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	

  - Là thanh ghi 8 bits nhưng chủ có 2 bits cao được sử dụng cho điều khiển ngắt.
  - Bit 7 – INT1 gọi là bit cho phép ngắt (Interrupt Enable), set bit bằng 1 nghĩa là cho phép ngắt INT1 hoạt động.
  - Bit 6 – INT0 gọi là bit cho phép ngắt (Interrupt Enable), set bit bằng 1 nghĩa là cho phép ngắt INT0 hoạt động.

- Thanh ghi cờ ngắt chung – GIFR (General Interrupt Flag Register).

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

  - Là thanh ghi 8 bits nhưng có 2 bits cao là các bit trạng thái sử dụng cho ngắt INT1 và INT0.
  - Nếu có 1 sự kiện ngắt phù hợp xảy ra trên chân INT1, thì bit INTF1 sẽ tự động set 1. Tương tự với INT0.

Sau khi thiết lập các bit phù hợp cho ngắt ngoài, việc sau cùng chúng ta cần làm là set bit I, tức là cho phép ngắt toàn cục.

3.6.3. Chương trình đếm lên, xuống sử dụng ngắt ngoài.

3.6.3.1 Chương trình viết bằng ASM.

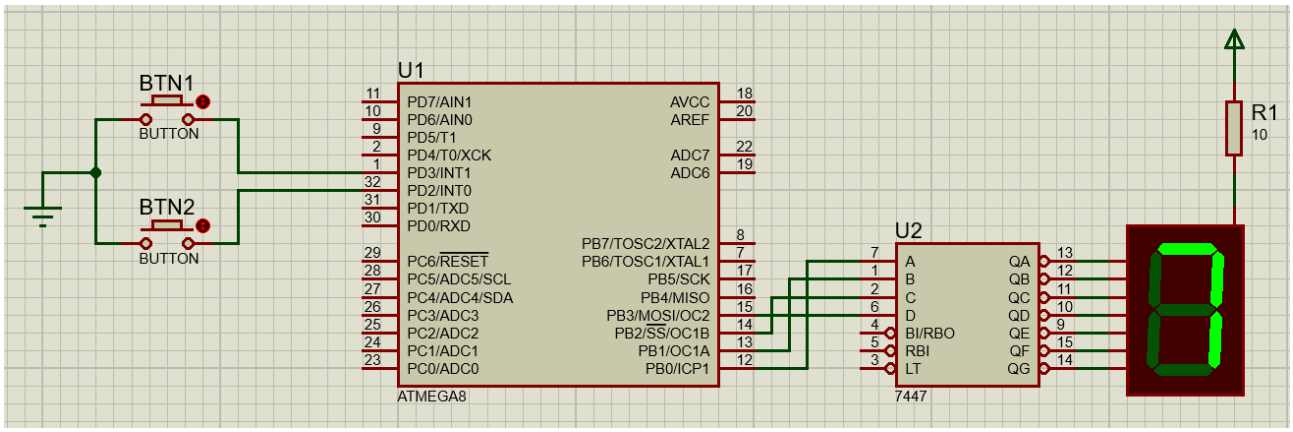
```
.CSEG
.ORG 0x0000
	RJMP BATDAU
.ORG 0x0020
BATDAU:
	;KHỞI ĐỘNG POINTER STACK
	LDI R17, HIGH(RAMEND)
	LDI R16, LOW(RAMEND)
	OUT SPL, R16
	OUT SPH, R17;
	;THIẾT LẬP CAC PORT
	CLR R16;XÓA R16, R16 = 0
	OUT DDRD, R16;DDRD=0, PORTB LÀ NGÕ  NHẬP
	LDI R16, 0xFF;SET ALL BIT R16 = 1
	OUT PORTD, R16;DDRD=0, PORTD=0xFF, ĐIỆN TRỞ KÉO LÊN - PULL UP RESISTOR
	OUT DDRB, R16;DDRD=0xFF, PORTD LÀ NGÕ XUẤT
	CLR R25; XÓA R25, R25 LÀ THANH GHI CHỨA SỐ ĐẾM
	CLR R20; R20 LÀ THANH GHI CHỨA GIÁ TRỊ TẠM THỜI TRƯỚC ĐÓ.
MAIN:
	IN R21, PINB;ĐỌC GIÁ TRỊ TỪ PINB, TỨC LÀ TỪ BUTTON
	RCALL SOSANH;GỌI CHƯƠNG TRÌNH CON SO SÁNH
	OUT PORTD, R25; XUẤT GIÁ TRỊ ĐẾM RA PORTC
	SBRS R21, 0 ; NẾU 0 R21(PB0=1) BỎ QUA DÒNG SAU
	RCALL TANG
	SBRS R21, 1;NẾU BIT R21 (PB1 = 1) THÌ BỎ QUA DÒNG SAU
	RCALL GIAM
	MOV R20, R21 ; LƯU LẠI TRẠNG THÁI PINB
	RJMP MAIN

;;CHƯƠNG TRÌNH CON KIỂM TRA SỐ TỪ 0 ĐẾN 9
SOSANH:
	CPI R25, 10
	BREQ RESET0;NẾU GIÁ TRỊ ĐẾM TẠM THỜI = 10 THÌ TRẢ VỀ 0
	CPI R25,255
	BREQ RESET9;NẾU GIÁ TRỊ ĐẾM TẠM THỜI = 255 THÌ TRẢ VỀ 9
	RJMP QUAYVE
RESET0:
	LDI R25,$0
	RJMP QUAYVE
RESET9:
	LDI R25,$9
QUAYVE:
	RET

;TĂNG SỐ ĐẾM LÊN 1 ĐƠN VỊ
TANG:
	SBRS R20,0
	RET
	INC R25
	RET

;GIẢM SỐ ĐẾM XUỐNG 1 ĐƠN VỊ.
GIAM:
	SBRS R20, 1
	RET
	DEC R25
```

```
        RET
```

3.6.3.2 Chương trình viết bằng C.

```
volatile int8_t val=0; //KHAI BÁO BIẾN VAL.

int main(void)
{
      DDRD = 0x00; //KHAI BÁO CỔNG D LÀ NGÕ NHẬP ĐỂ SỬ DỤNG CHO NGẮT.
       PORTD = 0xFF; //SỬ DỤNG ĐIỂN TRỞ NỘI KÉO LÊN.
       DDRB = 0xFF; //CỔNG C LÀ NGÕ XUẤT

       MCUCR |= (1<<ISC11)|(1<<ISC01); //THIẾT LẬP MODE: 2 NGẮT CẠNH XUỐNG.
       GICR |= (1<<INT1) | (1<<INT0); //THIẾT LẬP: CHO PHÉP NGẮT HOẠT ĐỘNG.
       sei();//THIẾT LẬP: NGẮT TOÀN CỤC.

    while (1)
    {
              PORTC++;
              _delay_loop_2(60000);
    }
       return 0;
}
```

3.6.3.3 Mô phỏng chương trình bằng Proteus.



Hình 12:Mạch đếm lên, xuống sử dụng ngắt ngoài bằng ATmega8.

3.7. Timer/Counter trên ATmega8.

3.7.1. Giới thiệu về Timer/Counter trên AVR.

Là module độc lập với CPU với chức năng chính là định thì (tạo ra một khoảng thời gian, đếm thời gian, …) và đếm sự kiện ngoài ra còn có chức năng tạo xung điều rộng PWM (Pulse Width Modulation).

Ở một số dòng AVR, một số Timer/Counter còn được dùng như các bộ canh chỉnh thời gian (calibration) trong các ứng dụng thời gian thực.

3.7.2. Timer/Counter trên chip ATmega8.

Trên chip ATmega8 có 2 bộ timer 8 bits (Timer/Counter0 và Timer/Counter2) và 1 bộ Timer 16 bits (Timer/Counter1).

- Timer/Counter0: là một bộ định thời, đếm đơn giản với 8 bits. Gọi là đơn giản vì bộ này chỉ có 1 chế độ hoạt động (mode) so với 5 chế độ của bộ

Timer/Counter1. Chế độ hoat động của Timer/Counter0 thực chất có thể coi như 2 chế độ nhỏ (và cũng là 2 chức năng cơ bản) đó là tạo ra một khoảng thời gian và đếm sự kiện.

- Timer/Counter1: là bộ định thời, đếm đa năng 16 bits. Bộ Timer/Counter này có 5 chế độ hoạt động chính. Ngoài các chức năng thông thường, Timer/Counter1 còn được dùng để tạo ra xung điều rộng PWM dùng cho các mục đích điều khiển. Có thể tạo 2 tín hiệu PWM độc lập trên các chân OC1A (chân 15) và OC1B (chân 16) bằng Timer/Counter1.
- Timer/Counter2: tuy là một module 8 bits như Timer/Counter0 nhưng Timer/Counter2 có đến 4 chế độ hoạt động như Timer/Counter1, ngoài ra nó nó còn được sử dụng như một module canh chỉnh thời gian cho các ứng dụng thời gian thực (chế độ asynchronous).

Một số định nghĩa:

- BOTTOM: giá trị thấp nhất mà một T/C có thể đạt được, giá trị luôn là 0.
- MAX: giá trị lớn nhất mà một T/C có thể đạt được, giá trị này quy định bởi độ rộng thanh ghi của T/C có thể chứa.
- TOP: giá trị mà khi T/C đạt đến nó sẽ thay đổi trạng thái, có thể thay đổi bằng cách điều khiển các bit điều khiển tương ứng hoặc nhập thông qua một số thanh ghi.

Thanh ghi dành cho hoạt động và điều khiển Timer/Counter0:

- TCNT0 (Timer/Counter Register):

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

  - Là 1 thanh ghi 8 bits chứa giá trị vận hành của T/C0. Thanh ghi này cho phép bạn đọc và ghi giá trị một cách trực tiếp.

- TCCR0 (Timer/Counter Control Register):

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	-	CS02	CS01	CS00	TCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Là thanh ghi điều khiển hoạt động của T/C0. Tuy là thanh ghi 8 bit nhưng chỉ có 3 bits tác dụng đó là CS00, CS01, CS02 là các bit chọn nguồn xung nhịp cho T/C0 (Clock Select) với chức năng các bit được mô tả như sau:

CS02	CS01	CS00	Description
0	0	0	No clock source (timer/counter stopped)
0	0	1	clkI/O/(No prescaling)
0	1	0	clkI/O/8 (From prescaling)
0	1	1	clkI/O/64 (From prescaling)
1	0	0	clkI/O/256 (From prescaling)

1	0	1	clkI/O/1024 (From prescaling)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

- TIMSK (Timer/Counter Interrupt Mask Register):

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

  - Là thanh ghi mặt nạ cho tất cả T/C trong ATmega8, trong đó chỉ có bit TOIE0 tức là bit số 0 liên quan đến T/C0.
  - TOIE0 là bit cho phép ngắt khi có tràn ở T/C0. Tràn là hiện tượng khi bộ giá trị trong thanh TCNT0 đạt MAX và đếm thêm 1 lần nữa.

- TIFR (Timer/Counter Interrupt Flag Register):

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

  - Là thanh ghi cờ nhớ cho tất cả bộ T/C, trong đó chỉ có bit TOV0 tức là bit số 0 liên quan đến T/C0.
  - TOV0 là cờ chỉ thị ngắt tràn của T/C0. Khi có ngắt tràn xảy ra, bit này tự động set 1.

Thanh ghi dành cho hoạt động và điều khiển Timer/Counter1: Vì là T/C 16 bits trong khi độ rộng bộ nhớ dữ liệu của ATmega8 là 8 bits nên đôi khi ta cần dùng cặp thanh ghi 8 bits tạo thành 1 thanh 16 bits.

- TCNT1H và TCNT1L (Timer/Counter Register):

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
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	

  - Là 2 thanh ghi 8 bits tạo thành thanh ghi 16 bits (TCNT1) chứa giá trị vận hành của T/C1. Cả 2 thanh ghi đều cho phép đọc và ghi trực tiếp.

- TCCR1A và TCCR1B (Timer/Counter Control Register):

  - Là 2 thanh ghi điều khiển hoạt động của T/C1. Tất cả các mode hoạt động của T/C1 đều được xác định thông qua các bit trong 2 thanh ghi này.
  - Tuy nhiên, đây không phải là 2 bytes cao và thấp của một thanh ghi mà là 2 thanh ghi hoàn toàn độc lập.

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Các bit này bao gồm bit chọn mode hay chọn dạng sóng (WGM – Waveform Generating Mode), quy định ngõ ra (COM – Compare Output Match), bit chọn giá trị chia prescaler cho xung nhịp (CS – Clock Slect).

- OCR1A và OCR1B (Ouput Compare Register A và B):

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

-

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

  - Chứa giá trị để so sánh.

- ICR1 (Input Capture Register 1):

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

  - Khi có sự kiện trên chân ICP1, thanh ghi ICR1 sẽ chụp lại giá trị của thanh ghi đếm TCNT1. Một ngắt có thể xảy ra, vì thể Input Capture có thể được dùng để cập nhật giá trị "TOP" của T/C1.

- TIMSK (Timer/Counter Interrupt Mask Register):
  - Bit 2 - TOIE1, là bit quy định ngắt tràn cho thanh T/C1.
  - Bit 3 - OCIE1b, là bit cho phép ngắt khi có 1 "Match" xảy ra trong việc so sánh TCNT1 và OCR1B.

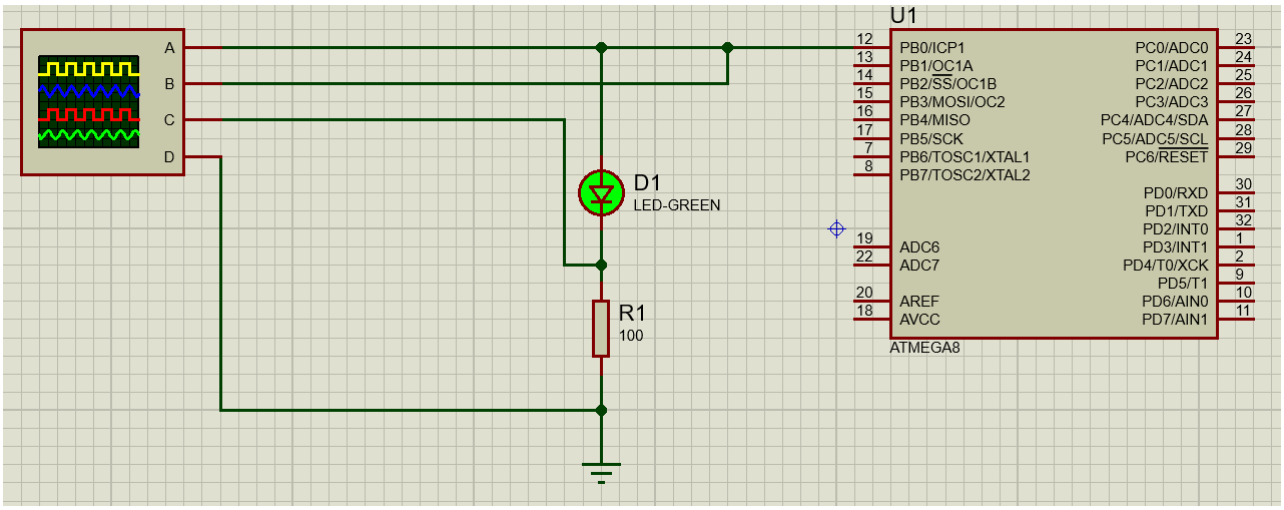
- Bit 4 - OCIE1A, là bit cho phép ngắt khi có 1 "Match" xảy ra trong việc so sánh TCNT1 với OCR1A.
- Bit 5 - TICIE1, là bit cho phép ngắt trong trường hợp Input Capture được dùng.

3.7.3. Chương trình nhấp nháy LED theo chu kì sử dụng Timer/Counter0 trên ATmega8.

3.7.3.1 Chương trình viết bằng C

```
int main(void){
	DDRB=0xFF;//CỔNG B LÀ CỔNG XUẤT
	PORTB=0x00;

	TCCR0=(1<<CS01);//THIẾT LẬP BỘ CHIA PRESCALER = 8
	TCNT0=131;//GÁN GIÁ TRỊ KHỞI TẠO CHO T/C0
	TIMSK=(1<<TOIE0);//CHO PHÉP NGẮT KHI CÓ TRÀN Ở T/C0
	sei();//CHO PHÉP NGẮT TOÀN CỤC
	while (1){
	}
	return 0;
}

//TRÌNH PHỤC VỤ NGẮT TRÀN Ở T/C0
ISR (TIMER0_OVF_vect ){
	TCNT0=131;//GÁN GIÁ TRỊ KHỞI TẠO CHO T/C0
	PORTB^=1;//ĐỔI TRẠNG THÁI Ở CỔNG B.
}
```

3.7.3.2 Mô phỏng chương trình bằng Proteus.



Hình 13: Mạch nhấp nháy led theo chu kì sử dụng Timer/Counter0 trên ATmega8.

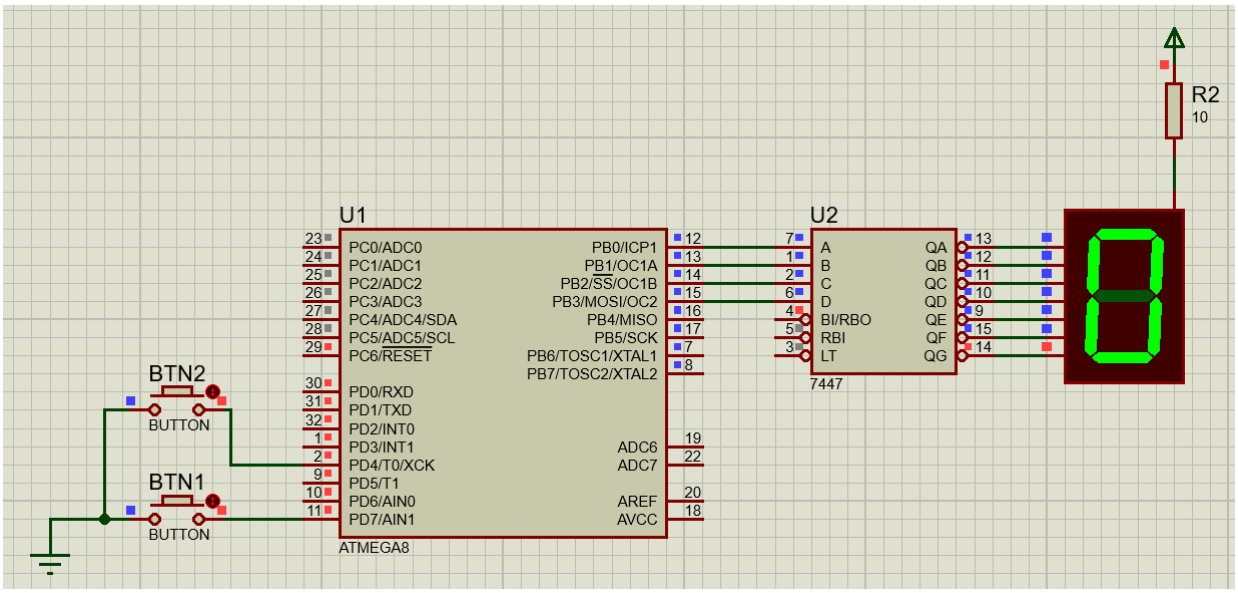
3.7.4. Chương trình đếm lên sử dụng Timer/Counter0 trên ATmega8.

3.7.4.1 Chương trình viết bằng C.

```
int main(void){
	DDRB=0xFF;//THIẾT LẬP CỔNG B LÀ CỔNG XUẤT.
	PORTB=0x00;
	DDRD=0x00;//THIẾT LẬP CỔNG D LÀ CỔNG VÀO ĐỂ NHẬN BUTTON
	PORTD=0xFF;//SỬ DỤNG ĐIỆN TRỞ KÉO LÊN Ở CỔNG B

	TCCR0=(1<<CS02)|(1<<CS01);//THIẾT LẬP MODE CLOCK ON FAILING EDGE.
```

```
        TCNT0=0;

        while (1){
              if (TCNT0==10) TCNT0=0;
              PORTB=TCNT0;//XUẤT GIÁ TRỊ RA LED 7 ĐOẠN
              if (bit_is_clear(PIND,7)) TCNT0=0;//ĐƯA VỀ GIÁ TRỊ 0 NẾU NHẤN BUTTON Ở
                                                CỔNG D, CỤ THỂ PD7.
        }
        return 0;
}
```

3.7.4.2 Mô phỏng chương trình bằng Proteus.



Hình 14: Mạch đếm lên sử dụng Timer/Counter0 trên ATmega8.

3.7.5. Chương trình nhấp nháy LED theo chu kì sử dụng Timer/Counter1 trên ATmega8.

3.7.5.1 Chương trình viết bằng C

```
int main(void)
{
    DDRB = 0xFF;//THIẾT LẬP CỔNG B LÀ CỔNG XUẤT
       PORTB = 0x00;//THIẾT LẬP GIÁ TRỊ ĐẦU

       TCCR1B =(1<<CS10)|(1<<CS11);//CHỌN MODE HOẠT ĐỘNG
       TCNT1 = 49910;//KHỞI TẠO GIÁ TRỊ ĐẦU CHO T/C1
       TIMSK = (1<<TOIE1);//CHO PHÉP NGẮT KHI CÓ TRÀN Ở T/C1
       sei();//CHO PHÉP NGẮT TOÀN CỤC
    while (1)
    {

    }
}
//CHƯƠNG TRÌNH PHỤC VỤ NGẮT T/C1
ISR(TIMER1_OVF_vect){
       TCNT1 = 49910;//KHỞI TẠO GIÁ TRỊ ĐẦU CHO T/C1
       PORTB ^= 1;//ĐẢO GIÁ TRỊ Ở CỔNG B.
}
```

3.7.5.2 Mô phỏng chương trình bằng Proteus.

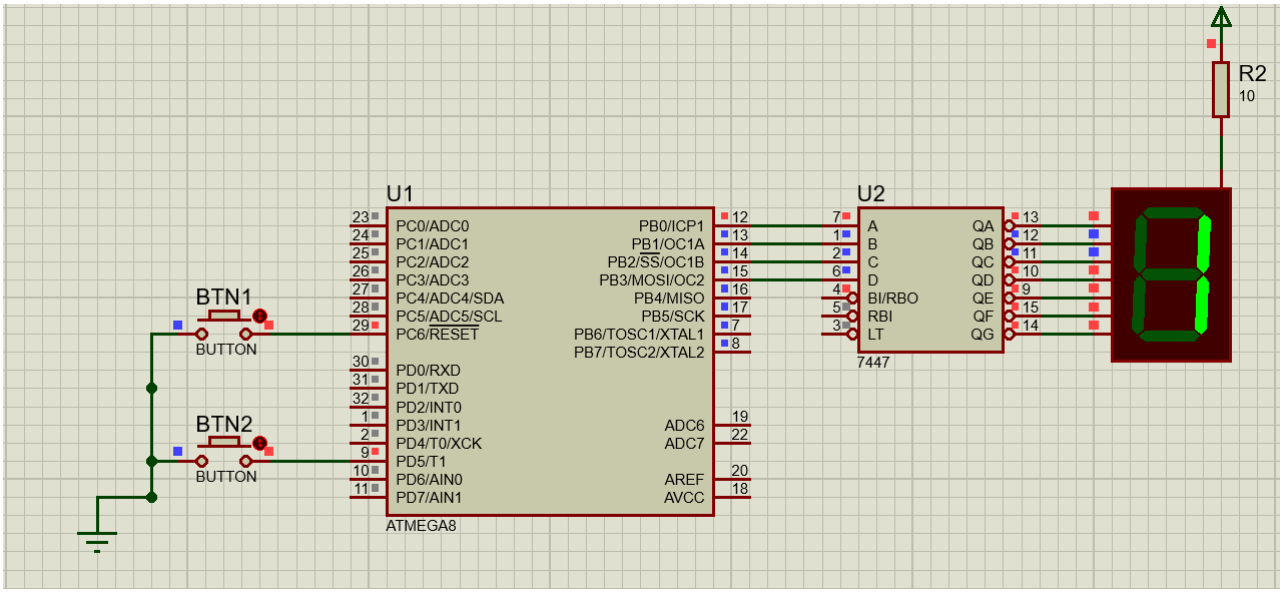
Note: giống mạch đã vẽ ở chương trình đếm lên sử dụng sử dụng Timer/Counter0 trên ATmega8.

3.7.7. Chương trình đếm lên khi "compare match" sử dụng Timer/Couter1 trên ATmega8.

3.7.7.1 Chương trình viết bằng C

```
volatile  unsigned char val=0;//KHAI BÁO BIẾN VAL ĐỂ GIỮ GIÁ TRỊ XUẤT RA CỔNG B
int main(void){
	DDRB=0xFF;//THIẾT LẬP CỔNG B LÀ CỔNG XUẤT.
	PORTB=0x00;//GIÁ TRỊ ĐẦU CỦA CỔNG = 0
	TCCR1B=(1<<WGM12)|(1<<CS12)|(1<<CS11);//CHỌN MODE CHO T/C 1 CỤ THỂ CTC4
	OCR1A=4;//GÁN GIÁ TRỊ CẦN SO SÁNH
	TIMSK=(1<<OCIE1A);//CHO PHÉP NGẮT KHI GIÁ TRỊ ĐẾM BẰNG OCR1A
	sei();//CHO PHÉP NGẮT TOÀN CỤC
	while (1){
		//do nothing
	}
	return 0;
}
//TRÌNH PHỤC VỤ NGẮT T/C1 KHI COMPARE MATCH.
ISR (TIMER1_COMPA_vect){
	val++;
	if (val==10) val=0;//GIỚI HẠN BIẾN VAL NẰM TRONG 0 ĐẾN 9
	PORTB =val;//XUẤT GIÁ TRỊ VAL RA CỔNG B.
}
```

3.7.7.2 Mô phỏng mạch bằng Proteus.



Hình 15:Mạch đếm khi "compare match" sử dụng Timer/Couter1 trên ATmega8.

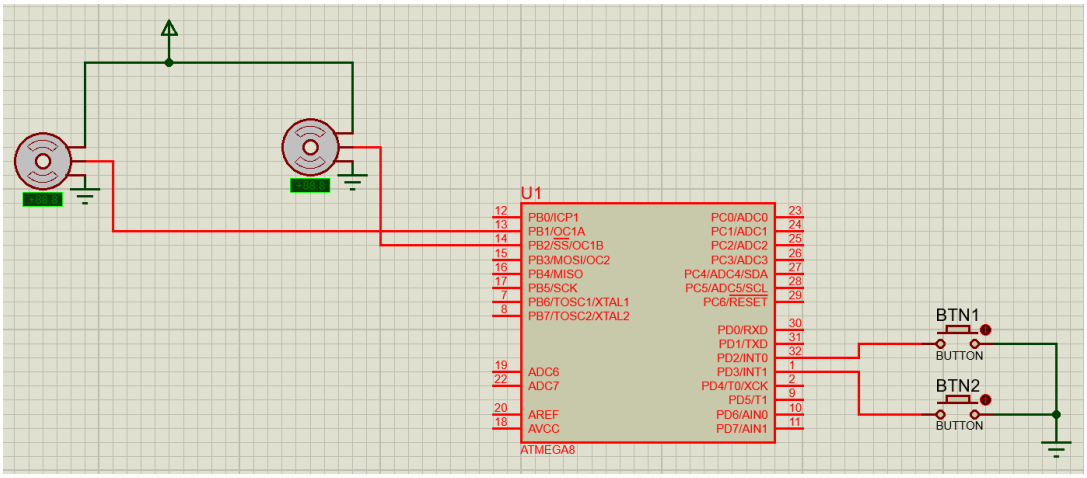
3.7.8. Chương trình điều khiển động cơ Servo bằng xung điều rộng sử dụng Timer/Counter1 trên ATmega8.

3.7.8.1 Chương trình viết bằng C.

```
int main(void)
{
	DDRB = 0xFF;//THIẾT LẬP CỔNG B LÀ CỔNG XUẤT
	PORTB = 0x00;//ĐẶT GIÁ TRỊ ĐẦU CỦA CỔNG B

	//THIẾT LẬP MODE CỦA T/C1
	MCUCR |= (1<<ISC11)|(1<<ISC01);
	GICR |= (1<<INT1)|(1<<INT0);

	TCCR1A |= (1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);
```

```
        TCCR1B |= (1<<WGM13)|(1<<WGM12)|(1<<CS10);
        //GÓC QUAY 90 ĐỘ
        OCR1A =  1000;
        OCR1B = 1500;
        ICR1 = 20000;

        sei();//CHO PHÉP NGẮT TOÀN CỤC
        while (1)
        {
        }

}
//CHƯƠNG TRÌNH PHỤC VỤ NGẮT
ISR(INT0_vect){
        if(OCR1A == 1000) OCR1A = 1500;
        else OCR1A = 1000;
}
ISR(INT1_vect){
        if(OCR1B == 1500) OCR1B = 2000;
        else OCR1B = 1500;
}
```

3.7.8.2 Mô phỏng chương trình bằng Proteus.



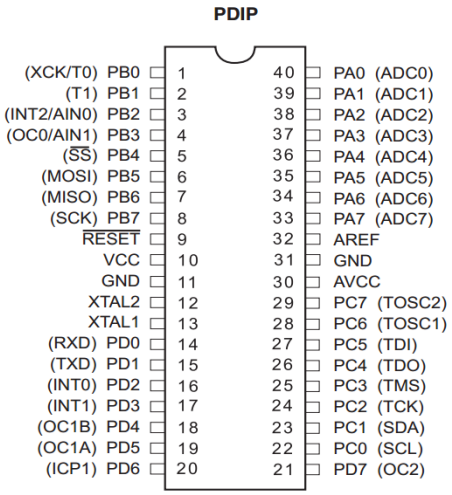
Hình 16:Mạch chương trình điều khiển động cơ Servo bằng xung điều rộng sử dụng Timer/Counter1 trên ATmega8.

4. Tìm hiểu về ATmega32.

Ở phần tiếp theo, chúng ta sẽ tìm hiểu về tổ chức và cách hoạt động của ATmega32. Về cơ bản, sự khác biệt của ATmega32 và ATmega8 không đáng kể.

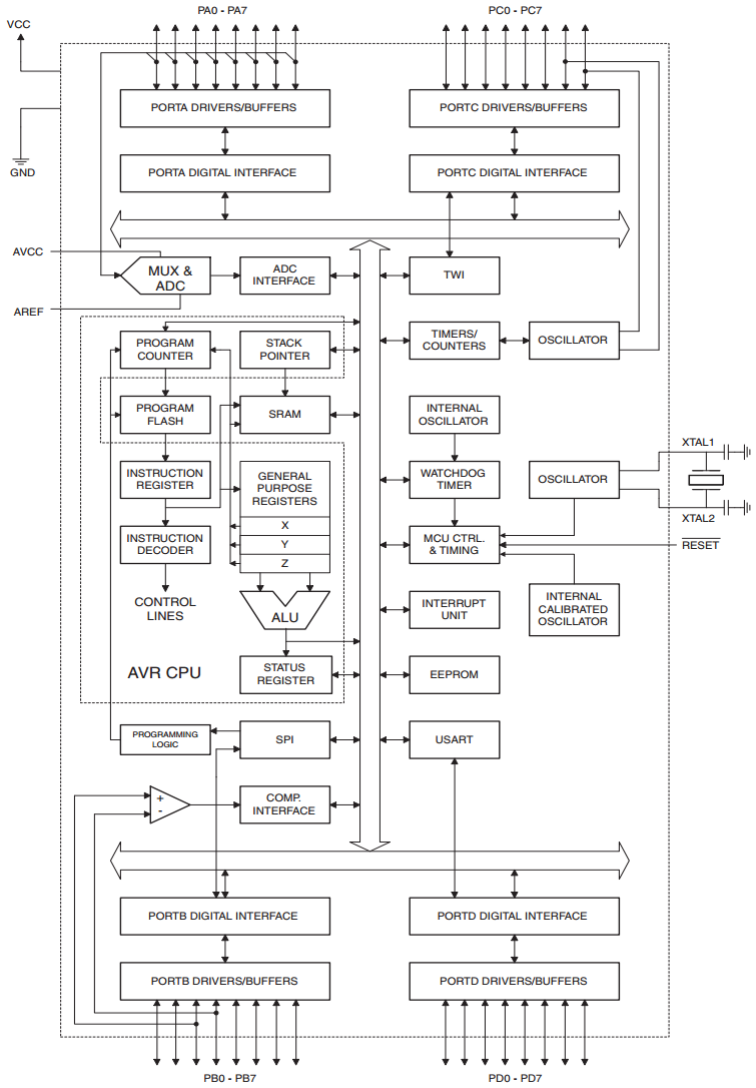
Hình 17: Hình ảnh thực tế ATmega32.

4.1. Cấu hình chân (Pin configurations).



Hình 18:Cấu trúc chân của ATmega32.

4.2. Sơ đồ khối.



Hình 19: Sơ đồ khối ATmega32.

4.3. Giao tiếp USART - Universal Synchronous & Asynchronous serial Reveiver and Transmitter.

4.3.1. Giới thiệu về USART.

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. Khái niệm USART thường để chỉ thiết bị phần cứng, không phải là môt chuẩn giao tiếp. USART cần phải kết hợp một thiết bị chuyển đổi mức điện áp để tạo một chuẩn giao tiếp nào đó.

Khái niệm "đồng bộ" để chị sự "báo trước" trong quá trình truyền, nó đòi hỏi ít nhất 2 đường truyền cho 1 quá trình.

Khái niệm "không đồng bộ" chỉ cần một đường truyền cho một quá trình. "Khung dữ liệu" đã được chuẩn hóa bởi thiết bị nên không cần đường xung nhịp báo trước dữ liệu đến. 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): quy định thời gian dành cho 1 bit truyền
- Frame (khung truyền): quy định về số bit trong mỗi lần truyền, các bit “báo” như bit Start và bit Stop, các bit kiểm tra như Parity, ngoài ra số lượng các bit trong một data cũng được quy định bởi khung truyền.
- Start bit: start 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.
- Data: data hay dữ liệu cần truyền là thông tin chính mà chúng ta cần gởi và nhận.
- Parity bit: parity là bit dùng kiểm tra dữ liệu truyền đúng không (một cách tương đối).
- Stop bits: stop bits là một hoặc các bit báo cho thiết bị nhận rằng một gói dữ liệu đã được gởi xong.

4.3.2. Truyền thông nối tiếp không đồng bộ trên ATmega32.

Có 3 chân chính liên quan đến module này là chân xung nhịp – XCK (chân số 1), chân truyền dữ liệu – TxD (Transmitted Data) và chân nhận dữ liệu – RxD (Received Data). Trong phần này ta chỉ cần quan tâm đến 2 chân TxD và RxD vì chân XCK chỉ được sử dụng như chân phát hoặc nhận xung dữ liệu trong chế độ truyền đồng bộ.

Có 5 thanh ghi thiết kế cho hoạt động và điều khiển USART:

- UDR (USART I/O Data Register):

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR
	TXB[7:0]								UDR
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	

  - Là một thanh ghi 8 bits chứa giá trị nhận 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.

- 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.

- UCSRA (USART Control and Status Register A):

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R/W	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Là một tromg 3 thanh ghi điều khiển hoạt động của USART.
  - 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.
  - Bit 1 - U2X là bit chỉ định gấp đôi tốc độ truyền.
  - Bit 0 - MPCM là bit chọn chế độ hoạt động đa xử lí (multi-processor).

- UCSRB (USART Control and Status Register B):

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
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 - RXCIE (Receive Complete Interrupt Enable) là bit cho phép ngắt khi quá trình nhận kết thúc.
  - Bit 6 - TXCIE (Transmit Complete Interrupt Enable) bit cho phép ngắt khi quá trình truyền kết thúc.
  - Bit 5 - UDRIE (USART Data Register Empty Interrupt Enable) là bit cho phép ngắt khi thanh ghi dữ liệu UDR trống.
  - Bit 4 - RXEN (Receiver Enable) là một bit điều khiển bộ nhận của USART, đề kích hoạt chức năng nhận dữ liệu bạn phải set bit này lên 1.
  - Bit 2 - TXEN (Transmitter Enable) là bit điều khiển bộ phát. Set bit này lên 1 bạn sẽ khởi động bộ phát của USART.
  - Bit 1 - RXB8 (Receive Data Bit 8) gọi là bit dữ liệu nhận thứ 8.
  - Bit 0 - TXB8 (Receive Data Bit 8) gọi là bit dữ liệu gửi thứ 8.

- UCSRC (USART Control and Status Register C):

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ1	TXB8	UCSRC
Read/Write	R/W	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Thanh ghi này có chung địa chỉ với thanh UBRRH (thanh ghi chứa byte cao dùng để xác lập tốc độ baud). Vì thế bit 7 trong thanh ghi này, tức bit URSEL là bit chọn thanh ghi. Khi URSEL=1, thanh ghi này được hiểu là thanh ghi điều khiển UCSRC, nếu bit URSEL=0 thì thanh ghi UBRRH sẽ được sử dụng.

- Bit 6 - UMSEL (USART Mode Select) là bit lựa chọn giữa 2 chế độ truyền thông đồng bộ và không đồng bộ.
- Bit 5 và 4 - UPM1 và UPM0 (Parity Mode) được dùng để quy định kiểm tra pariry.

UPM1	UPM2	Chế độ Parity
0	0	Không sử dụng
0	1	Không dùng
1	0	Parity chẵn
1	1	Parity lẻ

- Bit 2 và 1 - 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.

UCSZ2	UCSZ1	UCSZ0	Độ dài dữ liệu
0	0	0	5 bits
0	0	1	6 bits
0	1	0	7 bits
0	1	1	8 bits
1	0	0	Không dùng
1	0	1	Không dùng
1	1	0	Không dùng
1	1	1	9 bits

- Bit 0 - UCPOL (Clock Pority) là bit chỉ cực của xung kích trong chế độ truyền thông đồng bộ.

- UBRRL và UBRRH (USART Baud Rate Register):

Bit	15	14	13	12	11	10	9	8	
	URSEL	-	-	-	UBRR[11;8]				UBRRH
	UBRR[7;0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
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	

- Gán giá trị cho thanh UBRR, để USART dùng để tính tốc độ baud.

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

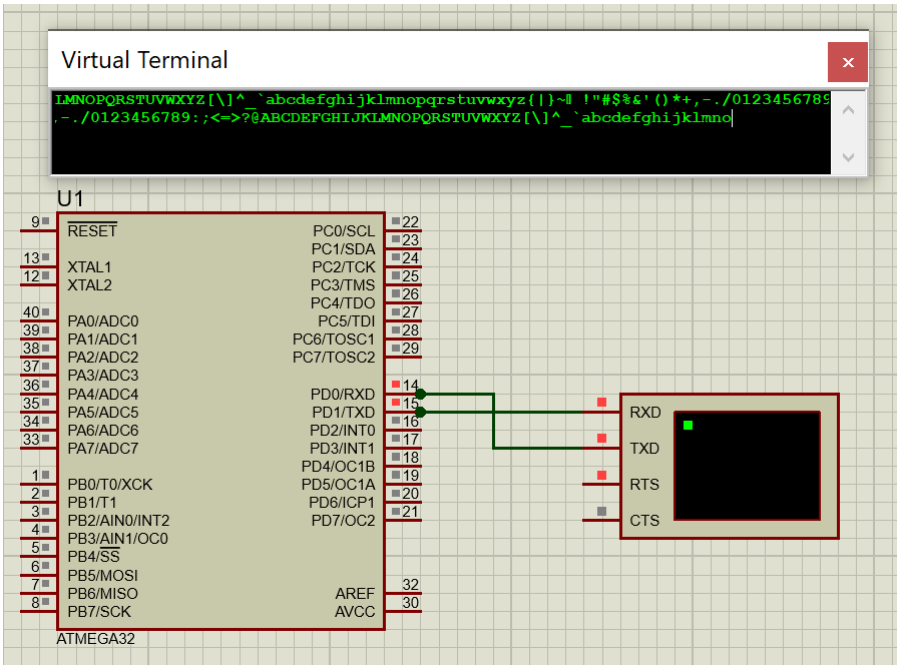
Thông thường, để sử dụng module USART trên AVR bạn phải thực hiện 3 việc quan trọng, đó là: cài đặt tốc độ baud (thanh ghi UBRR), định dạng khung truyền (UCSRB, UCSRC) và cuối cùng kích hoạt bộ truyền, bộ nhận, ngắt.

4.3.3. Chương trình truyền kí tự (truyền dữ liệu không đồng bộ USART) trên ATmega32.

4.3.3.1 Chương trình viết bằng C

```
//CHƯƠNG TRÌNH CON PHÁT DỮ LIỆU
void uart_char_tx(unsigned char chr){
	while (bit_is_clear(UCSRA,UDRE)) {};//LẶP CHO ĐẾN KHI UDRE != 1
	UDR=chr;//GHI DỮ LIỆU ĐỂ TRUYỀN ĐI
}

int main(void){
	//THIỆT LẬP TỐC ĐỘ TRUYỀN 57600K ỨNG VỚI F = 8MHZ
	UBRRH=0;
	UBRRL=8;

	//THIẾT LẬP KHUNG TRUYỀN VÀ KÍCH HOẠT BỘ TRUYỀN DỮ LIỆU
	UCSRA=0x00;
	UCSRC=(1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
	UCSRB=(1<<TXEN);

	while(1){
		for (char i=32; i<128; i++){
			uart_char_tx(i);//PHÁT
			_delay_ms(100);
		}
	}
}
```

4.3.3.2 Mô phỏng chương trình bằng Proteus.



Hình 20: Mạch chương trình truyền kí tự trên ATmega32.

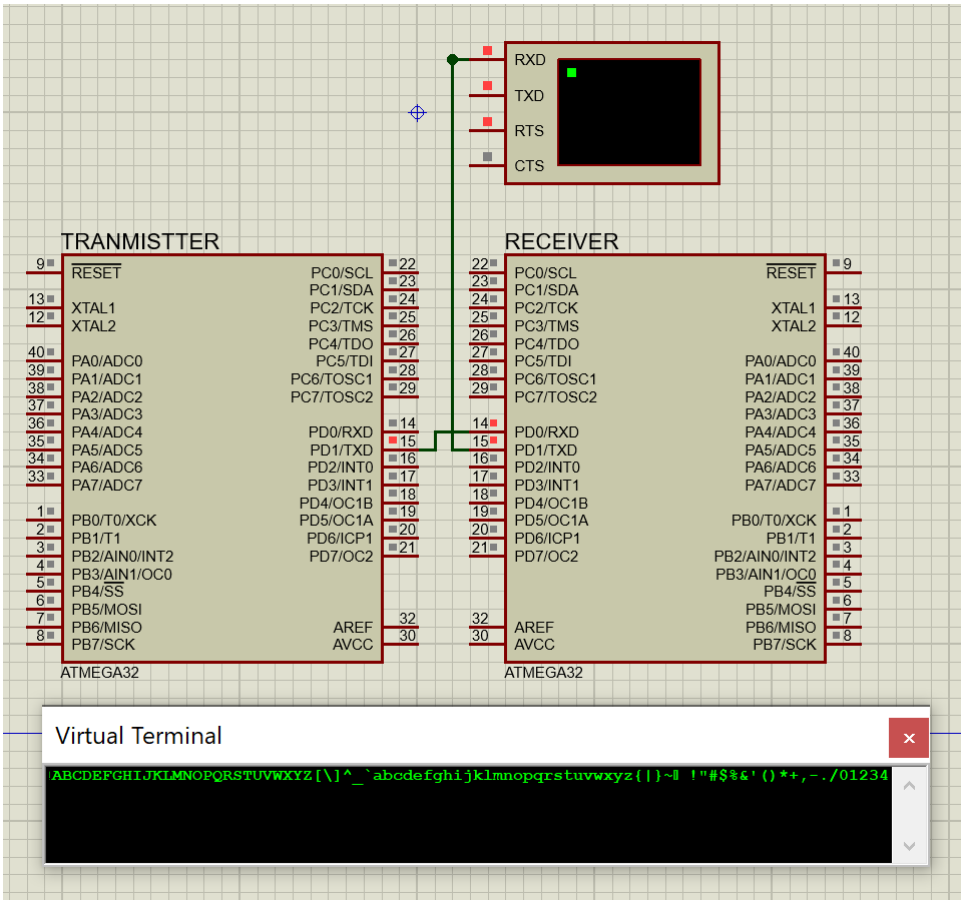
4.3.4. Chương trình truyền kí tự (truyền dữ liệu không đồng bộ USART) trên ATmega32.

4.3.4.1 Chương trình viết bằng C

```
//CHƯƠNG TRÌNH CON PHÁT DỮ LIỆU RA MÀN HÌNH
void uart_char_tx(unsigned char chr){
	while (bit_is_clear(UCSRA,UDRE)) {};//
	UDR=chr;
}
volatile unsigned char u_Data;//BIẾN LƯU GIÁ TRỊ NHẬN ĐƯỢC
```

```
int main(void){
       //THIẾT LẬP TỐC ĐỘ TRUYỀN 57600K ỨNG VỚI F = 8MHZ
      UBRRH=0;
      UBRRL=8;
        //THIẾT LẬP THÔNG SỐ KHÁC VÀ KÍCH HOẠT BỘ TRUYỀN, NHẬN DỮ LIỆU
      UCSRA=0x00;
      UCSRC=(1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
      UCSRB=(1<<RXEN)|(1<<TXEN)|(1<<RXCIE);//CHO PHÉP CẢ TRUYỀN VÀ NHẬN, NGẮT SAU KHI NHẬN
DỮ LIỆU
       sei(); ////CHO PHÉP NGẮT TOÀN CỤC
      while(1){      }
}
//TRÌNH PHỤC VỤ NGẮT USART
ISR(USART_RXC_vect){
       u_Data=UDR;
      uart_char_tx(u_Data);
}
```

4.3.4.2 Mô phỏng chương trình bằng Proteus.



Hình 21: Mạch chương trình nhận dữ liệu trên ATmeaga32.

4.4. Giao tiếp SPI - Serial Peripheral Bus

4.4.1. Giới thiệu SPI.

Giao tiếp SPI là một chuẩn truyền thông nối tiếp tốc độ cao do hang Motorola đề xuất.

Đây là kiểu truyền thông Master-Slave, trong đó có 1 chip Master điều phối quá trình tuyền thông và các chip Slaves được điều khiển bởi Master vì thế truyền thông chỉ xảy ra giữa Master và Slave. SPI là một cách truyền song công (full duplex).

SPI đôi khi được gọi là chuẩn truyền thông "4 dây" vì có 4 đường giao tiếp trong chuẩn này đó là SCK (Serial Clock), MISO (Master Input Slave Output), MOSI (Master Ouput Slave Input) và SS (Slave Select). Hình 1 thể hiện một kết SPI giữa một chip Master và 3 chip Slave thông qua 4 đường.

- SCK: Xung giữ nhịp cho giao tiếp SPI.
- MISO– Master Input / Slave Output: nếu là chip Master thì đây là đường Input còn nếu là chip Slave thì MISO lại là Output. MISO của Master và các Slaves được nối trực tiếp với nhau.
- MOSI – Master Output / Slave Input: nếu là chip Master thì đây là đường Output còn nếu là chip Slave thì MOSI là Input. MOSI của Master và các Slaves được nối trực tiếp với nhau.
- SS – Slave Select: SS là đường chọn Slave cần giao tiếp, trên các chip Slave đường SS sẽ ở mức cao khi không làm việc.

4.4.2. Truyền thông SPI trên ATmega32.

Module SPI trong các chip AVR hầu như hoàn toàn giống với chuẩn SPI mô tả trong phần trên. Phần bên dưới trình bày một số điểm quan trọng khi điều khiển SPI trên AVR.

Các chân SPI: Các chân giao tiếp SPI cũng chính là các chân PORT thông thường, vì thế nếu muốn sử dụng SPI chúng ta cần xác lập hướng cho các chân này. Trên chip ATmega32, các chân SPI như sau:

- SCK – PB7 (chân 8).
- MISO – PB6 (chân 7).
- MOSI – PB5 (chân 6).
- SS – PB4 (chân 5).

SPI được vận hành bởi 3 thanh ghi:

- SPCR (SPI Control Register):

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
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	

  - Là một thanh ghi 8 bits điều khiển tất cả hoạt động của SPI.
  - Bit 7- SPIE (SPI Interrupt Enable) bit cho phép ngắt SPI.
  - Bit 6 – SPE (SPI Enable). set bit này lên 1 để cho phép bộ SPI hoạt động. Nếu SPIE=0 thì module SPI dừng hoạt động.
  - Bit 5 – DORD (Data Order) bit này chỉ định thứ tự dữ liệu các bit được truyền và nhận trên các đường MISO và MOSI.
  - Bit 4 – MSTR (Master/Slave Select) nếu MSTR =1 thì chip được nhận diện là Master, ngược lại MSTR=0 thì chip là Slave.

- Bit 3 và 2 – CPOL và CPHA đây chính là 2 bits xác lập cực của xung giữ nhịp và cạnh sample dữ liệu tạo ra 4 Mode hoạt động.
- Bit 1:0 – CPR1:0 hai bit này kết hợp với bit SPI2X trong thanh ghi SPSR cho phép chọn tốc độ giao tiếp SPI, tốc độ này được xác lập dựa trên tốc độ nguồn xung clock chia cho một hệ số chia.

- SPSR (SPI Status Register):

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Là thanh ghi trạng thái của SPI.
- Bit 7 – SPIF là cờ báo SPI, khi một gói dữ liệu đã được truyền hoặc nhận từ SPI, bit SPIF sẽ tự động được set lên 1.
- Bit 6 – WCOL là bít báo va chạm dữ liệu (Write Colision), bit này được AVR set lên 1 nếu chúng ta cố tình viết 1 gói dữ liệu mới vào thanh ghi dữ liệu SPDR trong khi quá trình truyền nhận trước chưa kết thúc.
- Bit 0 – SPI2X gọi là bit nhân đôi tốc độ truyền, bit này kết hợp với 2 bits SPR1:0 trong thanh ghi điều khiển SPCR xác lập tốc độ cho SPI.

- SPDR (SPI Data Register):

Bit	7	6	5	4	3	2	1	0	
	MSB							LBS	SPSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	

- Là thanh ghi dư liệu của SPI.
- Trên chip Master, ghi giá trị vào thanh ghi SPDR sẽ kích quá trình tuyền thông SPI. Trên chip Slave, dữ liệu nhận được từ Master sẽ lưu trong thanh ghi SPDR, dữ liệu được lưu sẵn trong SPDR sẽ được truyền cho Master.

4.4.3. Chương trình truyền nhận một chiều với SPI trên ATmega32.

4.4.3.1 Chương trình viết bằng C

Chương trình dùng cho Master

```
#define cbi(port, bit) (port) &= ~(1<<(bit))
#define sbi(port, bit) (port) |= (1<<(bit))
//ĐỊNH NGHĨA CÁC ĐƯỜNG GIAO TIẾP SPI, PHỤ THUỘC TỪNG LOẠI CHIP
#define SPI_PORT      PORTB
#define SPI_DDR             DDRB
#define SCK_PIN             7
#define MISO_PIN      6
#define MOSI_PIN      5

#define ADDRESS_PORT  PORTB
#define ADDRESS_DDR         DDRB
#define Slave(i)            i
```

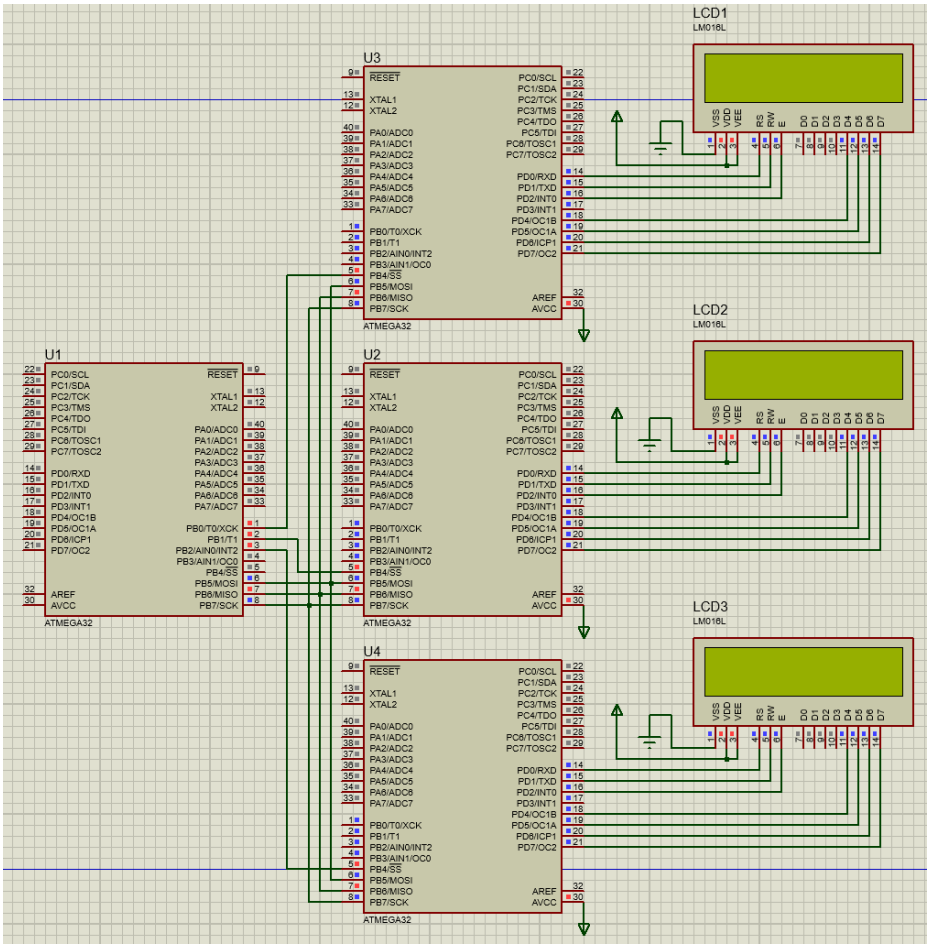
```
volatile uint8_t wData[3] = {0, 80, 160}, dis[];

//KHỞI ĐỘNG SPI Ở CHẾ ĐỘ MASTER
void SPI_MasterInit(void){
	SPI_DDR |= (1<<SCK_PIN)|(1<<MOSI_PIN);
	SPI_PORT |= (1<<MISO_PIN);//ĐIỆN TRỞ KÉO LÊN CHO CHÂN MISO.
	SPCR = (1<<SPIE)|(1<<SPE)|(1<<MSTR)|(1<<CPHA)|(1<<SPR1)|(1<<SPR0);

	//THIẾT LẬP CHÂN ĐIỀU KHIỂN CHO SLAVE.
	ADDRESS_DDR |= (1<<Slave(2)) | (1<<Slave(1)) | (1<<Slave(0));
	ADDRESS_PORT |= (1<<Slave(2)) | (1<<Slave(1)) | (1<<Slave(0));
}
//GỬI DỮ LIỆU 1 BYTES QUA SPI
void SPI_Transmit(uint8_t i, uint8_t data){
	cbi(ADDRESS_PORT, Slave(i));
	SPDR = data;
	while(bit_is_clear(SPSR, SPIF)){};
	sbi(ADDRESS_PORT, Slave(i));
}

int main(void)
{
    SPI_MasterInit();
       _delay_ms(100);
       while (1)
    {
		SPI_Transmit(0, wData[0]++);
		if (wData[0] > 80)
		{
			wData[0] = 0;
		}
		_delay_ms(10);

		SPI_Transmit(1, wData[1]++);
		if (wData[1] > 160)
		{
			wData[0] = 80;
		}
		_delay_ms(10);

		SPI_Transmit(0, wData[0]++);
		if (wData[2] > 240)
		{
			wData[2] = 160;
		}
		_delay_ms(500);
    }
}
```

Chương trình dùng cho Slave.

```
//ĐỊNH NGHĨA CÁC ĐƯỜNG GIAO TIẾP SPI, PHỤ THUỘC CẤU TRÚC CHÂN TỪNG CHIP
#define SPI_PORT PORTB
#define SPI_DDR DDRB
#define  SCK_PIN 7
#define MISO_PIN 6
#define MOSI_PIN 5
#define SS_PIN 4

volatile unsigned char wData = 16, rData;
volatile      uint8_t dis[5];;

void SPI_SlaveInit(void){
	SPI_DDR |= (1 <<MISO_PIN);
```

```
        SPI_PORT |= (1<<MOSI_PIN)|(1<<SS_PIN);
        SPCR = (1<<SPIE)|(1<<SPE)|(1<<CPHA)|(1<<SPR1)|(1<<SPR0);
}

int main(void)
{
        sei();
        SPI_SlaveInit();
        DDRD = 0xFF;
        init_LCD();
        clr_LCD();

    while (1)
    {
    }
}
//CHƯƠNG TRÌNH PHỤC VỤ NGẮT TRÊN SPI
ISR(SPI_STC_vect){
        rData = SPDR;
        clr_LCD();

        sprintf(dis, "%i", rData);
        move_LCD(1,1);
        print_LCD(dis);
}
```

4.4.3.2 Mô phỏng chương trình bằng Proteus.



Hình 22: Mạch chương trình truyền nhận một triều SPI trên ATmega32

4.5. Giao tiếp TWI tương thích chuẩn I^2C.

4.5.1. Giới thiệu truyền thông TWI.

TWI (Two-Wire Serial Intereafce) là một module truyền thông nối tiếp đồng bộ trên các chip AVR dựa trên chuẩn truyền thông I^2C. I^2C là viết tắc của từ Inter-Integrated Circuit là một chuẩn truyền thông do hãng điện tử Philips Semiconductor sáng lập và xây dựng thành chuẩn năm 1990.

TWI (I2C) là một truyền thông nối tiếp đa chip chủ (tạm dịch của cụm từ multi-master serial computer bus), được hiểu là trên cùng một bus có thể có nhiều hơn một thiết bị làm Master, đồng thời một Slave có thể trở thành Master nếu có khả năng.

TWI (I2C) được thực hiện trên 2 đường SDA (Serial DATA) và SCL (Serial Clock) trong đó SDA là đường truyền/nhận dữ liệu và SCL là đường xung nhịp.

Một số khái niệm đặc điểm của TWI:

- Master: là chip khởi động quá trình truyền nhận, phát đi địa chỉ của thiết bị cần giao tiếp và tạo xung giữ nhịp trên đường SCL.
- Slave: là chip có một địa chỉ cố định, được gọi bởi Master và phục vụ Master.
- SDA- Serial Data: là đường dữ liệu nối tiếp, tất cả các thông tin về địa chỉ hay dữ liệu đều được truyền trên đường này theo thứ tự từng bit một.
  Chú ý: trong chuẩn I2C, bit có trọng số lớn nhất (MSB) được truyền trước nhất, đặc điểm này ngược lại với chuẩn UART.
- SCL –Serial Clock: là đường giữ nhịp nối tiếp. TWI (I2C) là chuẩn truyền thông nối tiếp đồng bộ, cần có 1 đường tạo xung giữ nhịp cho quá trình truyền/nhận, cứ mỗi xung trên đường giữ nhịp SCL, một bit dữ liệu trên đường SDA sẽ được lấy mẫu (sample).

4.5.2. Giới thiệu truyền thông TWI trên ATmega32.

TWI trên ATmega32 được vận hành hởi 5 thanh ghi:

- TWBR (TWI Bit Rate Register)**:**

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	SPSR
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	

  - Là thanh ghi 8 bits quy định tốc độ phát xung giữ nhịp trên đường SCL của chip master, được tính theo công thức:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

- TWCR (TWI Control Register)**:**

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7- TWINT (TWI Interrupt Flag): là một cờ báo rất quan trọng. TWINT được tự động set lên 1 khi TWI kết thúc một quá trình bất kỳ nào đó.
- Bit 6 – TWEA (TWI Enable Acknowledge Bit): tạm hiểu là bit kích hoạt tín hiệu xác nhận.
- Bit 5 – TWSTA (TWI START Condition Bit): là bit tạo START condition.
- Bit 4 – TWSTO (TWI STOP Condition Bit): là bit tạo STOP condition cho TWI.
- Bit 3 – TWWC (TWI Write Collision Flag): khi cờ TWINT đang ở mức thấp tức TWI đang bận, nếu chúng ta viết dữ liệu vào thanh ghi dữ liệu (TWDR) thì một lỗi xảy ra, khi đó bit TWWC tự động được set lên 1.
- Bit 2 – TWEN (TWI Enable Bit): bit kích hoạt TWI trên AVR, khi TWEN được set lên 1, TWI sẵn sàng hoạt động.
- Bit 0 – TWIE (TWI Interrupt Enable Bit): bit cho phép ngắt TWI.

- TWSR (TWI Status Register)**:**

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

- Là thanh ghi 8 bits chứa 5 bits trạng thái của TWI và 2 bits chọn precaler

- TWDR (TWI Data Register)**:**

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

- Là thanh ghi dữ liệu chính của TWI. Trong quá trình nhận, dữ liệu được lưu trong TWDR. Trong quá trình gửi, dữ liệu chứa trong TWDR sẽ chuyển ra đường SDA.

- TWAR (TWI Address Register)**:**

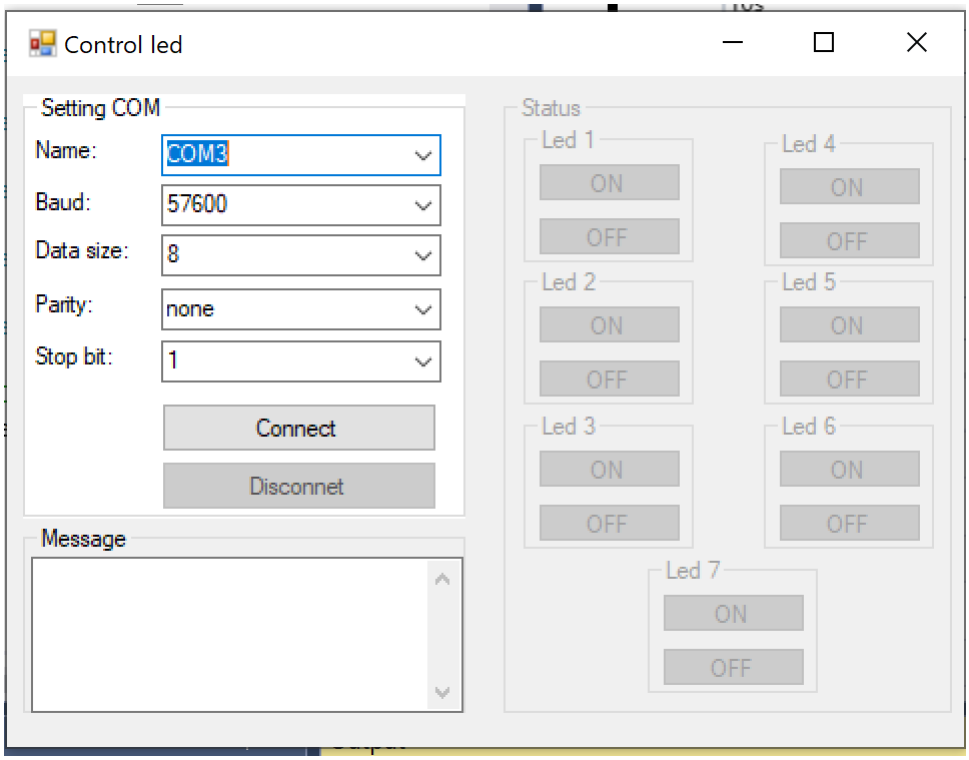
Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

- Là thanh ghi chứa device address của Slave.

Chương III: Kết quả - chương trình điều khiển mạch LED bằng ngôn ngữ C# thông qua cổng COM.

1. Chương trình điều khiển viết bằng ngôn ngữ C#.

Giao diện



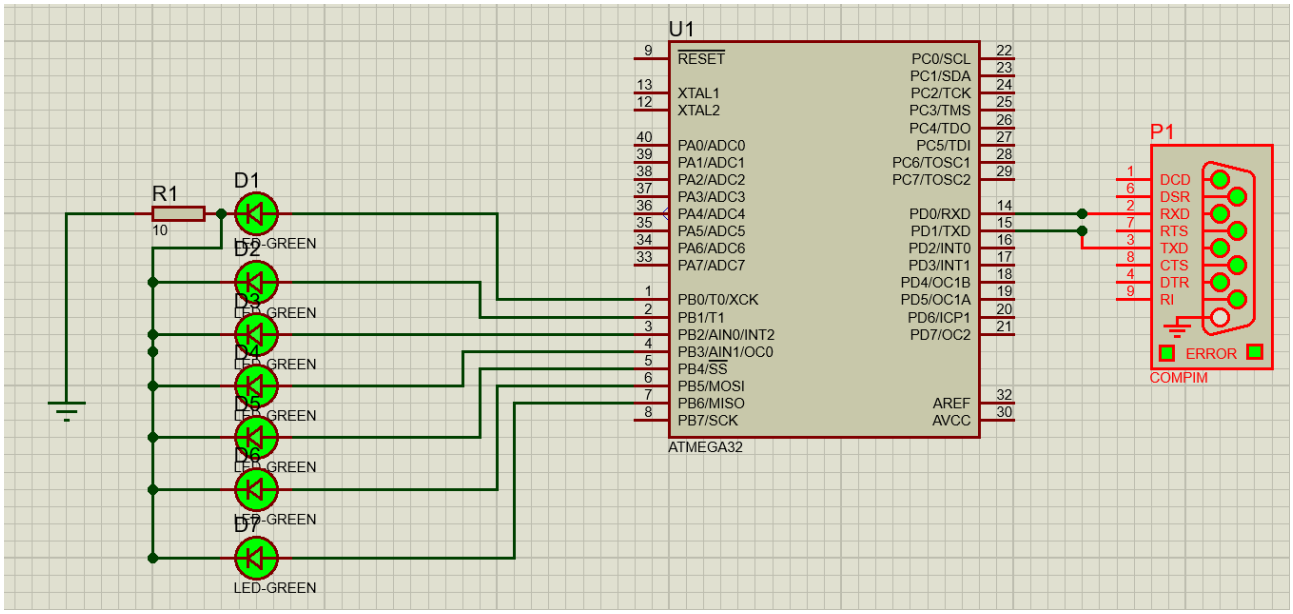
Hình 23: Giao diện chương trình điều khiển

2. Chương trình nhúng vào ATmega32.

```
//CHUONG TRINH CON TRUYEN DU LIEU
void led_control(unsigned char chr)
{
	PORTB = chr;
}

void sent_data(unsigned char chr)
{
	//while (bit_is_clear(UCSRA,UDRE)) {}; //cho den khi bit UDRE=1
	while(!(UCSRA & (1 << UDRE)));
	UDR=chr;
}

volatile unsigned char u_Data;
//BIẾN LƯU GIÁ TRỊ PINB CŨ VÀ NHẬN GIÁ TRỊ KIỂM TRA BAN ĐẦU
volatile unsigned char data_sent = 0x80;

int main(void)
{
    //THIẾT LẬP KHUNG TRUYỀN 57600K VỚI F = 8MHZ
	UBRRH = 0;
	UBRRL =8;
	//THIET LAP CONG B LÀ CỔNG XUẤT
	DDRB = 0xFF;

	//THIẾT LẬP THANH GHI GIAO TIẾP QUA UART
	UCSRA = 0x00;
	UCSRC = (1<<URSEL)|(1<<UCSZ1) | (1<<UCSZ0);
	UCSRB = (1<<RXCIE)|(1<<RXEN) |(1<<TXEN);
	sei();


    while (1)
```

```
        {
                if(data_sent != (char)PINB)
                {
                        sent_data(PINB);
                        //_delay_ms(500);
                        data_sent = (char)PINB;
                }
        }
}
//CHƯƠNG TRÌNH CON PHỤC VỤ NGẮT USART
ISR(USART_RXC_vect){
        u_Data = UDR;
        if(u_Data == 0x80)
        {
                sent_data(PINB);
        }
        else
        {
                PORTB = u_Data;
        }
}
```

3. Mạch mô phỏng



Hình 24: Mạch chương trình điều khiển 7 LED bằng ngôn ngữ C#

4. Ưu, nhược điểm và hướng phát triển.

Ưu điểm:

- Dữ liệu ở ATmega32 không cần gửi liên tục lên chương trình, mà nó chỉ gửi khi có thay đổi ở chân xuất dữ liệu (cụ thể là PORTB) hoặc khi người dùng gửi xuống yêu cầu ATmega32 gửi dữ liệu trả về (cụ thể là khi nhấn Connect).

Nhược điểm:

- Nếu giao tiếp với khung truyền dữ liệu n bits thì chỉ điều khiển tối đa cho (n – 1) LED (vì 1 bit dành cho yêu cầu phía chip gửi dữ liệu trả về).

Hướng phát triển:

- Tìm ra cách khắc phục nhược điểm trên.
- Bổ sung đầy đủ chức năng (ví dụ như thêm các khung truyền, Baud rate, …)
- Không chỉ đơn giản điều khiển LED mà có thể điều khiển các thiết bị khác có cùng cách điều khiển hoặc tương tự.

Chương IV: Kết luận.

Qua việc tìm hiểu về đề tài, em đã biết được kiến trúc và một số ứng dụng cơ bản của họ vi xử lý ARM. Giúp em có thêm kiến thức về lập trình nhúng và đã phát triển được một chương trình điều khiển LED thông qua cổng COM viết bằng ngôn ngữ C#.

Em xin chân thày cảm ơn thầy đã định hướng giúp em có thể hoàn thành bài báo cáo này.

Tài liệu tham khảo

https://en.wikipedia.org/wiki/AVR_microcontrollers
http://www.hocavr.com/
https://www.alldatasheet.com/