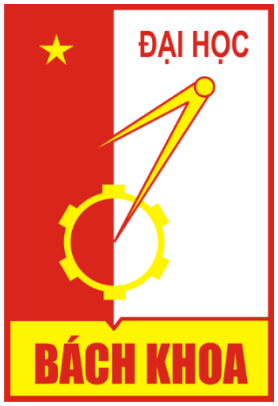
ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

****



BÁO CÁO THỰC HÀNH

HỌC PHẦN: HỆ NHÚNG

BÀI THỰC HÀNH SỐ 1

Lập trình vi điều khiển

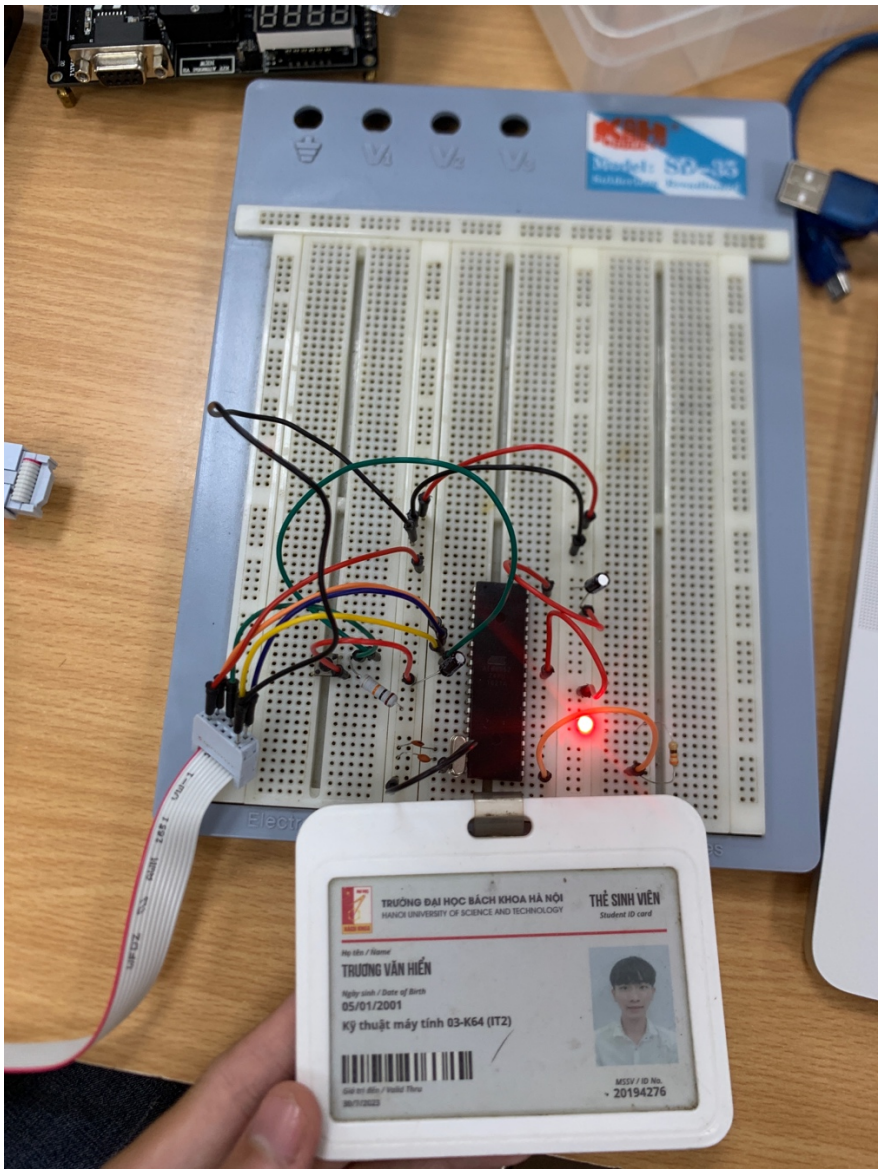
Họ và tên	:	Trương Văn Hiển
Mã số sinh viên	:	20194276
Lớp	:	727602
Giảng viên hướng dẫn	:	TS. Ngô Lam Trung

Hà Nội, tháng 05 năm 2023

MỤC LỤC

Phần 1: Lắp ráp một hệ vi điều khiển đơn giản

Mạch hoàn chỉnh cuối cùng:



Phần 2: Bài tập tự làm

Bài 1. Viết chương trình C thực hiện các hiệu ứng LED như sau (chạy lần lượt các hiệu ứng, mỗi hiệu ứng 2 lần, tạo trễ thích hợp):

(∘: LED off, O: LED on)

```c
#define led P1

int i, j, k;
int delayms = 100;  // Thời gian trễ giữa các lần sáng led

// Tạo độ trễ chính xác (ms) sử dụng Timer 0
void delay_ms(long ms) {
    ms *= 1000;  // Tính số chu kì máy (1000ckm = 1ms)
    while (ms > 0) {
        // Xóa thiết lập cũ của Timer0
        TMOD = TMOD & 0xF0;
        // Thiết lập Mode 1, 16-bit Timer/Counter
        TMOD = TMOD | 0x01;
        ET0 = 0;  // Che ngắt Timer 0
        // Kiểm tra xem số ckm còn lại có lớn hơn 65536
        if (ms >= 65536) {
            // Nếu lớn hơn thiết lập giá trị thanh ghi về 0
            TH0 = 0;
            TL0 = 0;
            ms -= 65536;
        } else {
            // Nếu nhỏ hơn thiết lập giá trị thanh ghi về 65536 - ms
            ms = 65536 - ms;
            TH0 = ms / 256;
            TL0 = ms % 256;
            ms = 0;
        }
        TF0 = 0;  // Xóa cờ tràn Timer 0
        TR0 = 1;  // Khởi động Timer 0
        while (TF0 == 0)
            ;     // Chờ đến khi tràn số
        TR0 = 0;  // Dừng Timer 0
    }
}
```

Hàm **delay_ms** được sử dụng để tạo độ trễ giữa các lần sáng LED. Nó sử dụng Timer 0 để đạt được độ chính xác theo ms.

1) Bật lần lượt từng cặp LED đơn từ trái qua phải và từ phải qua trái.

ooooooooo → OOooooooo → oOOooooo → ooOOoooo

→ oooOOooo → ... → ooooooOO → oooooooo

→ ooooooOOo → oooooOOoo → ... → oooooooo

```
// Hiệu ứng 1
void hieu_ung_1() {
    led = 0xFF;  // Tắt tất cả led
    delay_ms(delayms);

    unsigned char hieu_ung_1 = 0x03;  // Cho 2 led đầu sáng
    // Cho 2 led sáng chạy từ đầu đến cuối
    for (j = 0; j < 7; j++) {
        led = ~hieu_ung_1;
        delay_ms(delayms);
        hieu_ung_1 = hieu_ung_1 << 1;
    }

    // Cho 2 led sáng chạy từ cuối đến đầu
    hieu_ung_1 = 0xC0;  // Cho 2 led cuối sáng
    for (j = 0; j < 7; j++) {
        led = ~hieu_ung_1;
        delay_ms(delayms);
        hieu_ung_1 = hieu_ung_1 >> 1;
    }
}
```

Hàm **hieu_ung_1** thực hiện hiệu ứng LED như mô tả. Đầu tiên, tất cả các LED được tắt bằng cách gán giá trị 0xFF cho biến led. Giải thích code thực hiện hiệu ứng LED:

- Hai LED đầu tiên được bật sáng bằng cách gán giá trị 0x03 cho biến *hieu_ung_1*
- Hai LED sáng chạy từ đầu đến cuối bằng cách dịch trái *hieu_ung_1* và hiển thị giá trị đảo bit của *hieu_ung_1* trên LED
- Hai LED sáng chạy từ cuối đến đầu bằng cách dịch phải *hieu_ung_1* và hiển thị giá trị đảo bit của *hieu_ung_1* trên LED

2) Bật lần lượt từng LED từ trái và dồn qua phải rồi tắt lần lượt từ phải qua trái.

∘∘∘∘∘∘∘∘ → O∘∘∘∘∘∘∘ → ... → ∘∘∘∘∘∘∘O → O∘∘∘∘∘∘O

→ ... → ∘∘∘∘∘∘OO → ... → OOOOOOOO → OOOOOOO∘

→ OOOOOO∘∘ → ... → O∘∘∘∘∘∘∘ → ∘∘∘∘∘∘∘∘

```
// Hiệu ứng 2
void hieu_ung_2() {
    led = 0xFF;  // Tắt tất cả led
    delay_ms(delayms);

    // giu_hieu_ung lưu trữ trạng thái sáng của các led đã được dịch đến cuối
    unsigned char giu_hieu_ung = 0x00;
    // countled là số vòng lặp để dịch led (giảm dần)
    int countled = 8;

    // Hiệu ứng sáng dần
    for (j = 0; j < 8; j++) {
        unsigned char hieu_ung_2 = 0x01;  // bật sáng led đầu tiên
        // Bắt đầu vòng lặp dịch led đưa led về cuối
        for (k = 0; k < countled; k++) {
            // Kết hợp trạng thái sáng của led đang bị dịch và các led đã ở cuối
            led = ~(hieu_ung_2 | giu_hieu_ung);
            delay_ms(delayms);
            // Kiểm tra xem đã đến cuối vòng dịch led chưa
            // Nếu chưa thì tiếp tục dịch led
            if (k != countled - 1) hieu_ung_2 = hieu_ung_2 << 1;
        }
        // Khi led đã ở cuối lưu trữ lại trạng thái sáng của các led ở cuối
        giu_hieu_ung = hieu_ung_2 | giu_hieu_ung;
        countled--;  // Giảm vòng lặp dịch led
    }

    // Hiệu ứng tắt dần
    for (j = 0; j < 8; j++) {
        // Lúc này tất cả các led đã sáng nên giu_hieu_ung = 0xFF
        led = ~giu_hieu_ung;
        delay_ms(delayms);
        // Dịch từng bit của giu_hieu_ung sang phải để tắt led
        giu_hieu_ung = giu_hieu_ung >> 1;
    }
}
```

Hàm **hieu_ung_2** thực hiện hiệu ứng LED bật lần lượt từng LED từ trái và dồn qua phải, sau đó tắt lần lượt từ phải qua trái. Giải thích code:

- Hai trạng thái *LED_OFF* và *LED_ON* đã được định nghĩa bên trên để đại diện cho trạng thái tắt và bật của LED.
- Hàm **hieu_ung_2** bắt đầu bằng việc tắt tất cả các LED bằng cách gán giá trị 0xFF cho biến *led*. Sau đó, hiệu ứng LED được thực hiện như sau:
    1. Hiệu ứng sáng dần:
        - Biến *giu_hieu_ung* lưu trữ trạng thái sáng của các LED đã được dịch đến cuối.
        - Biến *countled* đếm số lần dịch LED và được khởi tạo là 8

- Vòng lặp đầu tiên j từ 0 đến 7 thực hiện dịch LED từ trái qua phải
- Biến *hieu_ung_2* được khởi tạo là 0x01 để bật sáng LED đầu tiên
- Vòng lặp k từ 0 đến countled - 1 dịch LED và hiển thị trạng thái sáng của LED đang bị dịch và các LED đã ở cuối
- LED được hiển thị bằng cách gán giá trị đảo bit của *hieu_ung_2* kết hợp với giu_hieu_ung trên biến led
- Sau mỗi vòng lặp, biến *hieu_ung_2* được dịch trái một bit (nếu chưa đến cuối vòng dịch LED)
- Khi vòng lặp kết thúc, trạng thái sáng của các LED đã đến cuối được lưu trữ vào *giu_hieu_ung*, bằng cách kết hợp giá trị *hieu_ung_2* với *giu_hieu_ung*
- Biến *countled* được giảm đi 1 để giảm số lần dịch LED

2. Hiệu ứng tắt dần:
    - Vòng lặp tiếp theo j từ 0 đến 7 thực hiện tắt LED từ phải qua trái
    - Toàn bộ các LED đã sáng trong hiệu ứng trước đó, nên *giu_hieu_ung* có giá trị là 0xFF
    - LED được tắt bằng cách gán giá trị đảo bit của *giu_hieu_ung* trên biến led
    - Sau mỗi vòng lặp, biến *giu_hieu_ung* được dịch phải một bit để tắt từng LED

Thông qua 2 vòng lặp này, hiệu ứng LED được tạo ra theo yêu cầu: bật lần lượt từng LED từ trái và dồn qua phải, sau đó tắt lần lượt từ phải qua trái.

3) Bật lần lượt 2 LED đối xứng từ ngoài vào trong rồi từ trong ra ngoài
∘∘∘∘∘∘∘∘ → O∘∘∘∘∘∘O → ∘O∘∘∘∘O∘ → ∘∘OO∘∘OO∘∘ →

∘∘∘OO∘∘∘ → ∘∘∘OO∘∘∘ → ∘∘O∘∘O∘∘ → ∘O∘∘∘∘O∘ →

O∘∘∘∘∘∘O → ∘∘∘∘∘∘∘∘

```c
// Hiệu ứng 3
void hieu_ung_3() {
    led = 0xFF;  // Tắt tất cả các led
    delay_ms(delayms);

    unsigned char hieu_ung_3_1 = 0x01;  // Bật led đầu tiên
    unsigned char hieu_ung_3_2 = 0x80;  // Bật led cuối cùng
    for (j = 0; j < 8; j++) {
        led = ~(hieu_ung_3_1 | hieu_ung_3_2);
        // Kiểm tra led 4, 5 có đang bị sáng lặp lại do dịch bit đối xứng không
        // Nếu không thì cho phép trễ
        if (hieu_ung_3_1 != 0x08) delay_ms(delayms);
        // Dịch 2 led dần dần sáng vào trong và rồi sáng lại ra phía ngoài
        hieu_ung_3_1 = hieu_ung_3_1 << 1;
        hieu_ung_3_2 = hieu_ung_3_2 >> 1;
    }
}
```

Hàm **hieu_ung_3** thực hiện hiệu ứng LED bật lần lượt 2 LED đối xứng từ ngoài vào trong, sau đó từ trong ra ngoài. Giải thích code:

- Hai trạng thái *LED_OFF* và *LED_ON* đã được định nghĩa bên trên để đại diện cho trạng thái tắt và bật của LED.
- Hàm **hieu_ung_3** bắt đầu bằng việc tắt tất cả các LED bằng cách gán giá trị 0xFF cho biến *led*. Sau đó, hiệu ứng LED được thực hiện như sau:
    - Biến *hieu_ung_3_1* được khởi tạo là 0x01 để bật sáng LED đầu tiên
    - Biến *hieu_ung_3_2* được khởi tạo là 0x80 để bật sáng LED cuối cùng
    - Vòng lặp j từ 0 đến 7 thực hiện hiệu ứng bật và tắt các LED đối xứng
    - LED được hiển thị bằng cách gán giá trị đảo bit của *hieu_ung_3_1* kết hợp với *hieu_ung_3_2* trên biến led
    - Sau mỗi vòng lặp, hai LED đối xứng được dịch vào trong bằng cách dịch trái *hieu_ung_3_1* và dịch phải *hieu_ung_3_2*
    - Trong quá trình dịch LED vào trong, chỉ có LED 4 và 5 không bị sáng lặp lại do dịch bit đối xứng, nên sau khi hiển thị LED 4 và 5, có một đợt trễ trước khi hiển thị LED tiếp theo

Thông qua vòng lặp này, hiệu ứng LED được tạo ra theo yêu cầu: bật lần lượt 2 LED đối xứng từ ngoài vào trong, sau đó từ trong ra ngoài.

Bài 2. Viết chương trình C sử dụng ngắt ngoài INT0, INT1 để thay đổi lần lượt 3 hiệu ứng LED trong bài 1. Bấm K4 sẽ chuyển sang hiệu ứng tiếp theo, bấm K3 để quay lại hiệu ứng trước.

Giải thích code:

1. Các biến:
    - *led*: biến đại diện cho thanh ghi P1, nơi điều khiển các LED
    - *i, j, k*: biến đếm trong vòng lặp
    - *hieu_ung_hien_tai*: biến lưu trữ hiệu ứng hiện tại của LED
    - *delayms*: thời gian trễ giữa các lần sáng LED
2. Các hàm:
    - ***delay_ms(long ms)***: hàm tạo độ trễ chính xác theo số milliseconds sử dụng Timer 0
    - ***hieu_ung_1()***: hàm thực hiện hiệu ứng 1 của LED
    - ***hieu_ung_2()***: hàm thực hiện hiệu ứng 2 của LED
    - ***hieu_ung_3()***: hàm thực hiện hiệu ứng 3 của LED
    - **i*nitExtInterrupt()***: hàm khởi tạo ngắt ngoài 0 và 1 (INT0 và INT1)
    - ***ISR0(), ISR1()***: hàm xử lý ngắt ngoài 0 và 1 (INT0 và INT1)
    - ***main()***: hàm chính của chương trình, trong đó sử dụng vòng lặp vô hạn để kiểm tra hiệu ứng hiện tại và chạy hiệu ứng đó

Các hiệu ứng LED được thực hiện trong các hàm **hieu_ung_1()**, **hieu_ung_2()** và **hieu_ung_3()**. Mỗi hiệu ứng được thực hiện bằng cách thay đổi giá trị của biến led để bật hoặc tắt các LED theo một cấu trúc và thời gian nhất định. Trong mỗi hiệu ứng, có các điều kiện kiểm tra *hieu_ung_hien_tai* để xác định xem hiệu ứng có thay đổi hay không. Nếu có sự thay đổi, hiệu ứng sẽ dừng lại và không tiếp tục thực hiện.

Bài 3. Viết chương trình C để mô phỏng đồng hồ đếm ngược. Sử dụng 2 LED 7 thanh bên trái để hiển thị số phút và 2 LED 7 thanh bên phải để hiển thị số giây. Ban đầu đồng hồ hiển thị giá trị 00.00. Sử dụng 1 nút bấm đề cài đặt thời gian đếm ngược theo phút (ví dụ: 05.00), và 1 nút bấm khác để bắt đầu đếm ngược. Khi đếm ngược về 00.00 thì còi (*buzzer*) sẽ phát 3 tiếng, mỗi tiếng 1s. ***Lưu ý:*** *sử dụng timer để tạo trễ chính xác.*

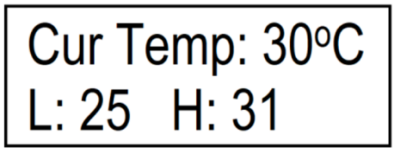
Minh họa đồng hồ đếm ngược

Giải thích code:

1. Khai báo biến và hằng số:
   - *x50ms*: Số vòng lặp đếm của Timer 1 tương ứng với 50ms.
   - *startCountDown*: Biến để kiểm tra xem đang trong quá trình đếm ngược hay không.
   - *minutes, seconds*: Biến lưu trữ giá trị số phút và số giây của đồng hồ.
   - *countDownLoop*: Biến lưu trữ số vòng lặp còn lại của Timer 1.
2. Các hàm:
   - **delay_ms**: tạo độ trễ chính xác (ms) sử dụng Timer 0.
   - **turnOnBuzzer_1s**: bật còi trong 1s
   - **initilize_timer1**: khởi tạo Timer 1 để đếm thời gian
   - **stop_timer1**: dừng Timer 1
   - **timer1_interrupt**: xử lý ngắt Timer 1. Được gọi khi Timer 1 tràn số (mỗi 50ms).
     - Kiểm tra xem đã đếm đủ 1s hay chưa
     - Nếu đã đếm đủ 1s, giảm thời gian hiển thị đi 1s
     - Nếu số giây và số phút đều bằng 0, dừng đếm ngược và kêu còi 3 lần
     - Thiết lập lại số vòng lặp của Timer 1 để tiếp tục đếm 1s
   - **initilize_int0, external0_interrupt**: xử lý ngắt ngoài 0. Được gọi khi nút bấm để cài đặt số phút được nhấn. Khi xảy ra ngắt, số phút đếm ngược sẽ tăng lên
   - **initilize_int1, external1_interrupt**: hàm khởi tạo và xử lý ngắt ngoài 1. Khi xảy ra ngắt, đồng hồ đếm ngược sẽ bắt đầu hoặc dừng lại
   - **output_7seg**: hiển thị giá trị lên LED 7 đoạn
   - **display_number**: quét lần lượt các số của LED 7 đoạn để hiển thị số phút và số giây

3. Hàm **main**: kiểm tra trạng thái đếm ngược và khởi tạo hoặc dừng Timer 1 tương ứng. Sau đó, số phút và số giây sẽ được hiển thị lên LED 7 đoạn. Vòng lặp chính sẽ tiếp tục lặp lại việc kiểm tra và hiển thị thời gian.

Bài 4. Viết chương trình C để viết ứng dụng theo dõi nhiệt độ theo thời gian thực. Cụ thể, nhiệt độ môi trường được nhận biết thông qua cảm biến nhiệt (DS18B20) và được cập nhật sau mỗi 6s. Thông tin nhiệt độ được hiển thị trên một LCD 1602. Cụ thể, dòng trên để hiển thị nhiệt độ tại thời điểm hiện tại và dòng dưới hiển thị nhiệt độ cao nhất và thấp nhất trong khoảng thời gian 1 phút theo dõi trước đó. Sử dụng 1 nút bấm (tùy chọn) để chuyển đổi giá trị nhiệt độ giữa °C và °F trên LCD.



Minh họa hiển thị nhiệt độ trên LCD1602

Giải thích code:

1. Khai báo biến và chân I/O:
   - *display_port*: Chân dữ liệu ra cho LCD
   - *DQ*: Chân dữ liệu của cảm biến nhiệt DS18B20
   - *tempArr*: Mảng lưu trữ giá trị nhiệt độ trong khoảng thời gian 1 phút theo dõi trước đó
   - *currentTempIndex*: Chỉ số của phần tử nhiệt độ hiện tại trong mảng tempArr
   - *temp*: Biến lưu trữ giá trị nhiệt độ đọc từ cảm biến DS18B20
   - *isFahrenheit*: Biến để chuyển đổi giữa đơn vị đo Celsius và Fahrenheit trên LCD
2. Các hàm:
   - **delay_ms**: Tạo độ trễ chính xác (theo miliseconds) sử dụng Timer 0
   - **initilize_int0, external0_interrupt**: Khởi tạo và xử lý ngắt ngoài 0 để chuyển đổi giữa đơn vị đo Celsius và Fahrenheit trên LCD
   - Các hàm làm việc với LCD:
     - **Wait_For_LCD**: Hàm đợi cho LCD
     - **lcd_cmd**: Gửi lệnh tới LCD.
     - **lcd_bytea**: Gửi dữ liệu tới LCD
     - **lcd_init**: Khởi tạo LCD
     - **display_str_lcd1602**: Hiển thị chuỗi ký tự lên LCD

- **display_char_lcd1602:** Hiển thị ký tự lên LCD

- Các hàm làm việc với cảm biến nhiệt độ DS18B20:
  - **delay_us_DS18B20**: Hàm trễ microsecond dành cho DS18B20
  - **Init_DS18B20**: Khởi tạo cảm biến DS18B20
  - **ReadByteFromScratchpad**: Đọc 1 byte từ Scratchpad
  - **WriteByteToScratchpad**: Ghi 1 byte vào Scratchpad
  - **ReadTemperature**: Đọc nhiệt độ từ cảm biến DS18B20
- Các hàm hỗ trợ chuyển đổi:
  - **celsius_to_fahrenheit**: Chuyển đổi độ Celsius sang độ Fahrenheit
  - **convert**: Chuyển đổi số nguyên thành chuỗi

3. Hàm **main** là hàm chính của chương trình, thực hiện các công việc sau:
   - Khởi tạo biến và cấu hình chân I/O
   - Khởi tạo LCD
   - Trong vòng lặp vô hạn, chương trình sẽ:
     - Đọc giá trị nhiệt độ từ cảm biến DS18B20
     - Hiển thị giá trị nhiệt độ lên màn hình LCD
     - Chuyển đổi đơn vị đo nhiệt độ nếu cần
     - Hiển thị giá trị nhiệt độ đã chuyển đổi lên màn hình LCD
   - Chương trình không thoát và tiếp tục chạy vòng lặp vô hạn cho đến khi bị ngắt nguồn hoặc kết thúc bằng lệnh break