

Bài 6 : Bàn phím

Nội dung:

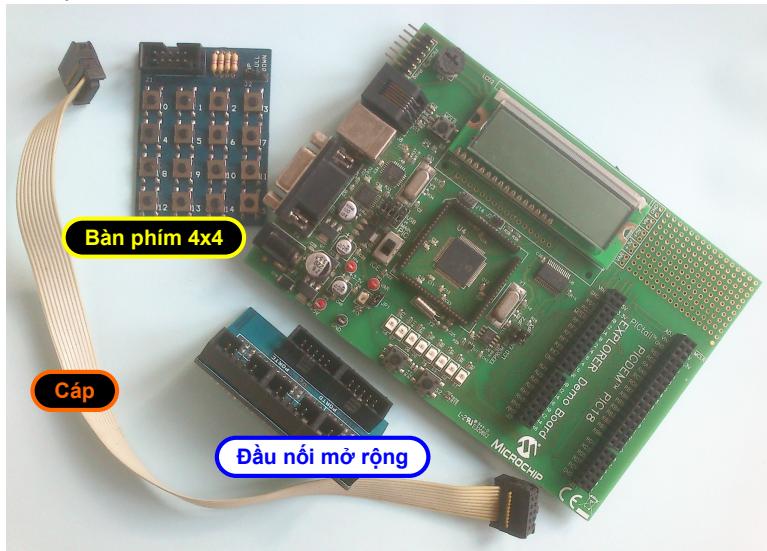
- Nd1. Khảo sát cấu tạo, hoạt động của ma trận phím.
- Nd2. Tìm hiểu kỹ thuật "quét hàng / đọc cột", định vị phím nhấn.
- Nd3. Chống rung phím bằng phần mềm (bài làm thêm).

Yêu cầu:

- Yc1. Kiểm tra phím trên từng hàng.
- Yc2. Viết chương trình con kiểm tra quét hàng theo chỉ số.
- Yc3. Kết hợp với ngắt thời gian 10ms gọi chương trình con kiểm tra quét hàng, định vị phím, xuất số thứ tự phím ra LED 8-bit (hoặc LCD).

6.1 Các bước hiện thực yêu cầu 1

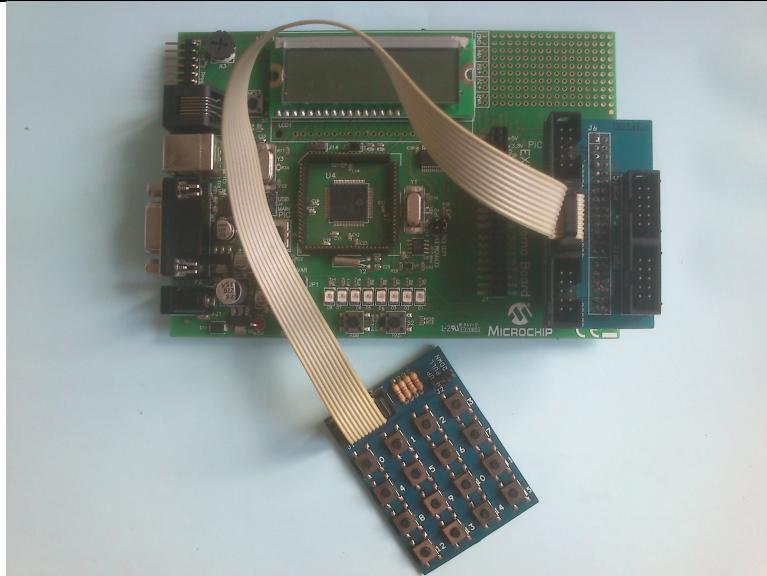
- Bước 1. Tạo dự án mới Tn05, tập tin nguồn Tn05.c.
- Bước 2. Khởi động: port LED 8-bit, Timer0, portB (xem yêu cầu bên dưới).
- Bước 3. Kết nối mạch:



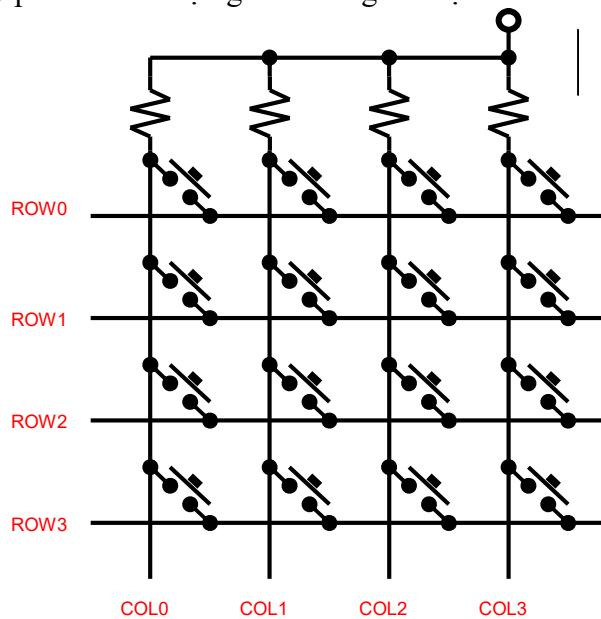
- Cắm connector mở rộng vào card PICDEM PIC18 như trong hình.



- Kết nối : bàn phím - dây cáp - portB của connector mở rộng như hình sau:



Bước 4. Kết nối hàng cột với PortB.
Ma trận 16 phím có cấu tạo gồm 4 hàng x 4 cột như hình bên dưới.



Cột có trở kéo lên để đảm bảo mức logic 1 khi phím không được nhấn.
Sử dụng PortB để kết nối với ma trận 16 phím (4 hàng x 4 cột) :

- Cột COL0 - COL3 : cổng nhập, nối vào các bit RB₀₋₃.
- Hàng ROW0 - ROW3 : cổng xuất, nối vào RB₄₋₇.

6.2 Kiểm tra bàn phím

Bước 5. Cấu hình PortB cho 4 bit cao là xuất, 4 bit thấp là nhập (sinh viên phải xác định giá trị cho thanh ghi TRISB).

Các định nghĩa hàng

```
#define X4N4 0x0F
#define ROW0 0b11101111
#define ROW1 0b11011111
#define ROW2 0b10111111
#define ROW3 0b01111111
```

```

volatile unsigned char keyout @ 0xF8A; //LATB
volatile unsigned char keyin @ 0xF81; //PORTB
volatile unsigned char key_io @ 0xF93; //TRISB
volatile unsigned char led @ 0xF8C; //LATD
volatile unsigned char led_io @ 0xF95; //TRISB

```

Hàm **init()**

```

void init()
{
    ADCON1=0x0F; //nhap digital
    key_io=X4N4; //B7-B4:hang xuat, B3-B0:cot nhap
    led_io=XUAT; //port LED xuat
    led=0;
}

```

Bước 6. Thực hiện kiểm tra phím trên 1 hàng. Làm cho 1 hàng xuống mức 0, các hàng khác lên mức 1. Sử dụng các dữ liệu ROW0 - ROW3 để xuất ra cổng chọn hàng (keyout). Sau đó, đọc dữ liệu về từ cột, chuyển sang port LED để quan sát kết quả (chủ yếu là 4 bit thấp).

```

void main(void)
{
    init();
    keyout=ROW1;
    while(1) led=keyin;
}

```

Bước 7. Dịch, chạy, nhấn phím trên hàng 1 và quan sát port LED.

Bước 8. Thực hiện tiếp kiểm tra trên các hàng còn lại.

6.3 Quét toàn bàn phím:

Bước 9. Thêm vào module sử dụng Timer0 tạo thời gian 20ms. Trong **timer_process()** gọi hàm **quetphim()** thực hiện kiểm tra phím theo các bước kế tiếp.

Bước 10. Sử dụng một biến **idx** (giá trị = 0, 1, 2, 3) để ghi nhớ vị trí hàng đang được kiểm tra.

```

#define MAXIDX 4
#define ROW0 0b11101111

```

Như vậy, dữ liệu xuất ra cổng **keyout** (còn được gọi là mã quét - scan code) sẽ thay đổi tùy theo chỉ số hàng. Việc xác định mã quét có thể được thực hiện bằng nhiều cách như:

- Tra bảng các dữ liệu từ ROW0 đến ROW3 (bước 5).
- Kiểm tra giá trị **idx** rồi gán giá trị từ ROW0 đến ROW3.
- Tạo scan_code từ ROW0 bằng phép dịch trái theo **idx**.

```

unsigned char dl_quet;
keyout=0xFF; //chon hang theo idx
dl_quet=ROW0<<idx;
keyout=dl_quet;

```

Bước 11. Hàm **quetphim()** như sau:

```

void quetphim()
{
    unsigned char dl_quet;
    keyout=0xFF; //chon hang theo idx
    dl_quet=ROW0<<idx;
    keyout=dl_quet;
    getkey(); //doc cot, phat hien phim
    idx=(++idx)%MAXIDX;
    if (keyreg!=NOKEY) xulyphim();
}

```

Sau khi gọi scancode ra cổng keyout, gọi hàm **getkey()** để xác định mã phím (keycode) đã nhấn và xuất ra LED để kiểm tra lại.

Hàng 1 (ROW0): keycode từ 0 đến 3.

Hàng 2 (ROW1): keycode từ 4 đến 7.

Hàng 3 (ROW2): keycode từ 8 đến 11.

Hàng 4 (ROW3): keycode từ 12 đến 15.

Có thể sử dụng công thức : **keycode = idx * 4 + chỉ số cột** (chỉ số cột = 0÷3)

Như vậy, cần xác định chỉ số cột (0-3) của phím được nhấn từ dữ liệu đọc được từ cổng **keyin** và tạo mã phím như trong hàm **getkey()** sau:

```
#define COL1      0b00001101
#define COL2      0b00001011
#define COL3      0b00000111
#define NOKEY     0xFF

void getkey()
{
    unsigned char col;
    col=keyin & 0x0F;           //RB3-RB0
    switch (col)
    {
        case COL0: keyreg=0; break;
        case COL1: keyreg=1; break;
        case COL2: keyreg=2; break;
        case COL3: keyreg=3; break;
        default: keyreg=NOKEY;
    }
    //doi ra so thu tu (keycode)
    if (keyreg!=NOKEY) keyreg+=(idx*4);
}
```

Chú ý, do việc tạo ra mã phím có sử dụng chỉ số **idx** nên tạo mã quét xong mới thực hiện tăng chỉ số **idx** (hàm **quetphim()**).

Bước 12. Cuối cùng, nếu keycode nhận được khác với NOKEY thì gọi hàm **xulyphim()** thực hiện chức năng mềm được gán cho phím. Trong bài thí nghiệm này, hàm **xulyphim()** chỉ hiển thị mã phím ra LED. Tuy nhiên, sinh viên có thể chọn một vài phím nào đó để làm các công việc mà ta đã làm ở các bài thí nghiệm trước.

6.4 Bài tập

- Hiện mã phím hoặc một câu thông báo ra màn hình LCD khi nhấn phím.
- Xử lý chống rung phím bằng phần mềm. Cách làm là so sánh keycode hiện tại với old code (là keycode lưu lại ở lần bấm trước) và với NOKEY để xác nhận nhấn/nhả phím.