

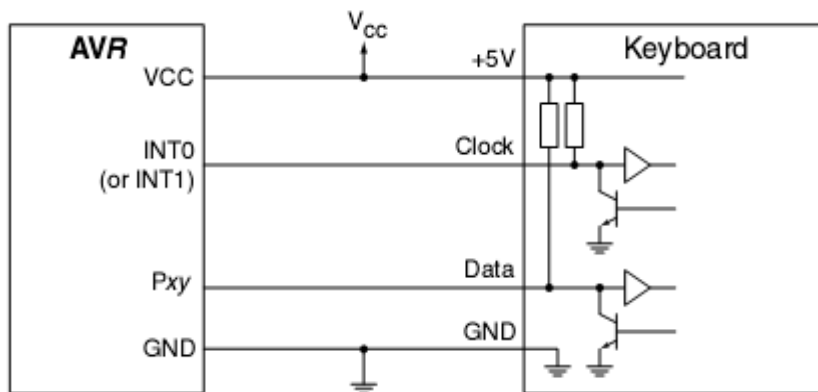
KEYBOARD INTERFACE WITH MICROCONTROLLER

Most microcontrollers require some kind of a human interface. This tutorial describes one way of doing this i.e. interfacing a standard PS2 keyboard with Atmega (microcontroller) to display text on a LCD.



Two connector types are available, the 5-pin DIN connector of "5D" type, and the smaller six-pin mini-DIN. The pin assignments are shown in the figure.

The signal lines are open connector, with pull-up resistors (internal) located in the keyboard.

CONNECTIONS



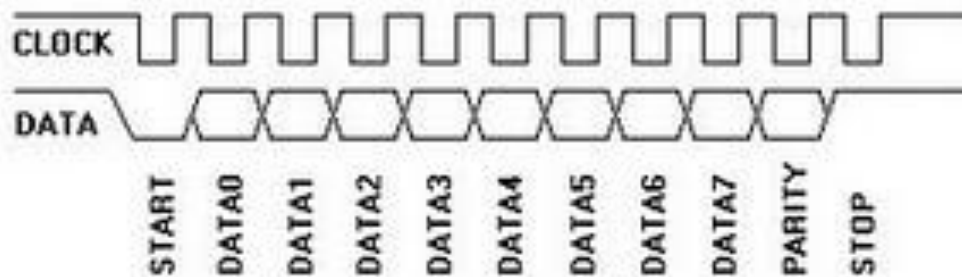
Keyboard connector pin arrangement.

AT Computer		
Signals	DIN41524, Female at Computer, 5-pin DIN 180°	6-pin Mini DIN PS2 Style Female at Computer
Clock	1	5
Data	2	1
nc	3	2,6
GND	4	3
+5V	5	4
Shield	Shell	Shell

PS/2 ALGORITHM

TIMING

The protocol is: one start bit (always 0), eight data bits, one odd parity bit and one stop bit (always 1). The data is valid during the low period (falling edge) of the clock pulse. The keyboard is generating the clock signal, and the clock pulses are typically 30-50 μ s low and 30-50 μ s high.



The host system can send commands to the keyboard by forcing the clock line low. It then pulls the data line low (the start bit). Now, the clock line must be released. The keyboard will count 10 clock pulses. The data line must be set up to the right level by the host before the trailing edge of the clock pulse. **After the tenth bit, the keyboard checks for a high level on the data line (the stop bit), and if it is high, it forces it low.** This tells the host that the data is received by the keyboard. The software in this design note will not send any commands to the keyboard

SCAN CODES

The AT keyboard has a scan code associated with each key. When a key is pressed, this code is transmitted. If a key is held down for a while, it starts repeating. The repeat rate is typically 10 per second. When a key is released, a "break" code ($\$F0$) is transmitted followed by the key scan code. For most of the keys, the scan code is one byte. Some keys like the *Home*, *Insert* and *Delete* keys have an extended scan code, from two to five bytes. The first byte is always $\$E0$. This is also true for the "break" sequence, e.g., $E0 F0 xx...$

The code supplied with this tutorial is a simple keyboard to RS-232 interface. The scan codes received from the keyboard are translated into appropriate ASCII characters and transmitted by the UART. **The source code is written in C.**

Keyboard reception is handled by the interrupt function **INT0_interrupt**. The reception will operate independent of the rest of the program.

The algorithm is quite simple: Store the value of the data line at the leading edge of the clock pulse. This is easily handled if the clock line is connected to the INT0 or INT1 pin. The interrupt function will be executed at every edge of the clock cycle, and data will be stored at the falling edge. After all bits are received, the data can be decoded. This is done by calling the **decode** function. For character keys, this function will store an ASCII character in a buffer. It will take into account if the shift key is held down when a key is pressed. Other keys like function keys, navigation keys (arrow keys, page up/down keys etc.) and modifier keys like Ctrl and Alt are ignored.

C CODE FOR KEYBOARD INTERFACING

```
#include <mega16.h>
```

```

// Alphanumeric LCD Module functions
#asm
.equ __lcd_port=0x18 ;PORTB //PORT B has been used for lcd display
#endasm
#include <lcd.h>
#include <delay.h>

#ifndef __KB_INCLUDED
#define __KB_INCLUDED

#include<mega16.h>
#define CLOCK 2 //PIND.2 to be connected to clock
#define DATAPIN 3 //PIND.3 has been used as data pin.

#define ISC00 0
#define ISC01 1

void InitKeyboard(void); //function declaration
void decode(unsigned char sc);
void put_kbbuff(unsigned char c);
int getchar_kb(void);
void print_hexbyte(unsigned char i);
void keyboard_mode();
void number(int []);
void capital(int []);
#endif
// External Interrupt 0 service routine
#define BUFF_SIZE 64

unsigned char edge, bitcount; // 0 = neg. 1 = pos.

unsigned char kb_buffer[BUFF_SIZE];
unsigned char *inpt, *outpt;
unsigned char buffcnt;
flash unsigned char unshifted[68][2] = {
0x0d,9,
0x0e,'|',
0x15,'q',
0x16,'1',
0x1a,'z',
0x1b,'s',
0x1c,'a',
0x1d,'w',
0x1e,'2',
0x21,'c',
0x22,'x',
0x23,'d',
0x24,'e',
0x25,'4',
0x26,'3',
0x29,' ',
0x2a,'v',
0x2b,'f',
0x2c,'t',
0x2d,'r',
0x2e,'5',
0x31,'n',
0x32,'b',

```

```
0x33, 'h',
0x34, 'g',
0x35, 'y',
0x36, '6',
0x39, ',',
0x3a, 'm',
0x3b, 'j',
0x3c, 'u',
0x3d, '7',
0x3e, '8',
0x41, ', ',
0x42, 'k',
0x43, 'i',
0x44, 'o',
0x45, '0',
0x46, '9',
0x49, '. ',
0x4a, '- ',
0x4b, 'l',
0x4c, 'ø',
0x4d, 'p',
0x4e, '+',
0x52, 'æ',
0x54, 'ã',
0x55, '\\',
0x5a, 13,
0x5b, '...',
0x5d, '\\',
0x61, '<',
0x66, 8,
0x69, '1',
0x6b, '4',
0x6c, '7',
0x70, '0',
0x71, ', ',
0x72, '2',
0x73, '5',
0x74, '6',
0x75, '8',
0x79, '+',
0x7a, '3',
0x7b, '- ',
0x7c, '*',
0x7d, '9',
0, 0
};
```

```
// Shifted characters
flash unsigned char shifted[68][2] = {
0x0d, 9,
0x0e, '§',
0x15, 'Q',
0x16, '!',
0x1a, 'Z',
0x1b, 'S',
0x1c, 'A',
0x1d, 'W',
0x1e, '"',
0x21, 'C',
```

```
0x22, 'X',
0x23, 'D',
0x24, 'E',
0x25, 'x',
0x26, '#',
0x29, ' ',
0x2a, 'V',
0x2b, 'F',
0x2c, 'T',
0x2d, 'R',
0x2e, '%',
0x31, 'N',
0x32, 'B',
0x33, 'H',
0x34, 'G',
0x35, 'Y',
0x36, '&',
0x39, 'L',
0x3a, 'M',
0x3b, 'J',
0x3c, 'U',
0x3d, '/',
0x3e, '(',
0x41, ';',
0x42, 'K',
0x43, 'I',
0x44, 'O',
0x45, '=',
0x46, ')',
0x49, ':',
0x4a, '_',
0x4b, 'L',
0x4c, 'Ø',
0x4d, 'P',
0x4e, '?',
0x52, 'Æ',
0x54, 'Å',
0x55, '`',
0x5a, 13,
0x5b, '^',
0x5d, '*',
0x61, '>',
0x66, 8,
0x69, '1',
0x6b, '4',
0x6c, '7',
0x70, '0',
0x71, ',',
0x72, '2',
0x73, '5',
0x74, '6',
0x75, '8',
0x79, '+',
0x7a, '3',
0x7b, '-',
0x7c, '*',
0x7d, '9',
0,0
};
```

```

void InitKeyBoard(void)
{
    inpt = kb_buffer;           // Initialize buffer
    outpt = kb_buffer;
    buffcnt = 0;

    MCUCR = 2;                 // INT0 interrupt on falling edge
    edge = 0;                 // 0 = falling edge 1 = rising edge
    bitcount = 11;
    #asm("sei")                // interrupt enable
}

```

```

interrupt [EXT_INT0] void ext_int0_isr(void) //external interrupt function
{
    static unsigned char data;           // Holds the received scan code

    if(bitcount < 11 && bitcount > 2) // Bit 3 to 10 is data. Parity bit,
    {                                     // start and stop bits are ignored.
        data = (data >> 1);
        if(PIND & 8)
            data = data | 0x80;         // Store a '1'
    }

    if(--bitcount == 0)                // All bits received
    {
        decode(data);
        bitcount = 11;
    }
}

```

```

void decode(unsigned char sc)
{
    static unsigned char is_up=0, shift = 0, mode = 0;
    unsigned char i;

    if (!is_up)                        // Last data received was the up-key identifier
    {
        switch (sc)
        {
            case 0xF0 :                // The up-key identifier
                is_up = 1;
                break;

            case 0x12 :                // Left SHIFT
                shift = 1;
                break;

            case 0x59 :                // Right SHIFT
                shift = 1;
                break;

            case 0x05 :                // F1
                if(mode == 0)
                    mode = 1;         // Enter scan code mode
                if(mode == 2)
                    mode = 3;         // Leave scan code mode
                break;
        }
    }
}

```

```

default:
    if(mode == 0 || mode == 3)          // If ASCII mode
    {
        if(!shift)                     // If shift not pressed,
        {                               // do a table look-up
            for(i = 0; unshifted[i][0]!=sc && unshifted[i][0]; i++);
            if (unshifted[i][0] == sc) {
                put_kbbuff(unshifted[i][1]);
            }
        } else {                       // If shift pressed
            for(i = 0; shifted[i][0]!=sc && shifted[i][0]; i++);
            if (shifted[i][0] == sc) {
                put_kbbuff(shifted[i][1]);
            }
        }
    } else {                            // Scan code mode
        print_hexbyte(sc);              // Print scan code
        put_kbbuff(' ');
    }
    break;
}
} else {
    is_up = 0;                          // Two 0xF0 in a row not allowed
    switch (sc)
    {
        case 0x12 :                     // Left SHIFT
            shift = 0;
            break;

        case 0x59 :                     // Right SHIFT
            shift = 0;
            break;

        case 0x05 :                     // F1
            if(mode == 1)
                mode = 2;
            if(mode == 3)
                mode = 0;
            break;

        case 0x06 :                     // F2
            // clr();
            break;
    }
}
}

```

```

void put_kbbuff(unsigned char c)
{
    if (buffcnt<BUFF_SIZE)              // If buffer not full
    {
        *inpt = c;                      // Put character into buffer
        inpt++;                          // Increment pointer

        buffcnt++;

        if (inpt >= kb_buffer + BUFF_SIZE) // Pointer wrapping

```

```

        inpt = kb_buffer;
    }
}

int getchar_kb(void)
{
    int byte;
    while(buffcnt == 0 );           // Wait for data

    byte = *outpt;                  // Get byte
    outpt++;                         // Increment pointer

    if (outpt >= kb_buffer + BUFF_SIZE) // Pointer wrapping
        outpt = kb_buffer;

    buffcnt--;                       // Decrement buffer count

    return byte;
}

void print_hexbyte(unsigned char i) //function to convert into hexadecimal
code
{
    unsigned char h, l;

    h = i & 0xF0;                    // High nibble
    h = h>>4;
    h = h + '0';

    if (h > '9')
        h = h + 7;

    l = (i & 0x0F)+'0';              // Low nibble
    if (l > '9')
        l = l + 7;

    put_kbbuff(h);
    put_kbbuff(l);
}

void main(void)
{

    InitKeyBoard();
    // LCD module initialization
    lcd_init(16);

    // Global enable interrupts
    #asm("sei")
    lcd_clear();

    unsigned char key,i,k;
    i=0; k=0;
    while (1)
    {
        lcd_gotoxy(i,k);
        i++;
        if (i==15) {
            i=0; //to move to next line

```



```
        k++;
    }
    key = getchar_kb();
    if (key=='!') { lcd_clear(); i=k=0; } //!' has been used to clear screen
    else if (key==8)
    {
        if (i!=0) i=i-2;
        else {
            k--; i=15;
        }
        lcd_gotoxy(i,k);
        lcd_putchar(0x20);
    }
    else(lcd_putchar(key))

};
}
```