Enhanced Keyboard with Multimedia capability and Application Launch Function

Raj Parihar

Introduction

This application note explains the implementation of Universal Serial Bus (USB) keyboard with enhanced features using microchip's USB firmware. Firmware incorporates QWERTY keyboard, multimedia volume control, system power management functionalities along with Application Launch (AL) functions. Design has been implemented using Human Interface Device (HID) class in a composite USB device manner with consumer control application.

Features

- Simple keyboard with Alpha-Numeric keys
- Power management function support i.e. System Wake UP, SLEEP etc.
- Multimedia functionalities with Volume control
- Application Launch capability
- Compatible with Windows and Linux operating system
- No Custom Driver Required, uses generic drivers
- Firmware is portable across all PIC with USB part

USB Protocol: Overview

USB is a simple, reliable, and versatile interface. It is a cost effective and fast means of communication between devices and Host with many other advantages. Universal Bus connectors and 4-wire simple connection makes it a suitable interface for compact devices as well. The bus allows peripherals to be attached, configured, used and detached while the host and other peripherals are in operation.

Application Space in USB

According to USB 2.0 specification

Specification	Bit Rate	Typical Application
LOW - SPEED	1.5 Mbps	Interactive Devices i.e. Keyboard, Mouse, joysticks etc.
FULL - SPEED	12 Mbps	Phone, Audio i.e. Broadband, Microphone etc.
HIGH - SPEED	480 Mbps	Video, Storage i.e. Video application, Imaging etc.

USB Device Terminology

USB device is a collection of end points which can be addressed from USB host controller uniquely. According to USB 2.0 specification Endpoint is "a uniquely addressable portion of USB device that is the source or sink of information in a communication flow between the host and device". The unique address required for each endpoint consists of an endpoint number (which may range from 0 to 15) and direction (IN or OUT). The endpoint direction is always from the host's perspective; IN is towards the host and OUT is away from the host. An endpoint configured to do control transfers must transfer data in both directions, so a control endpoint actually consists of a pair of IN and OUT endpoints that share an endpoint number. All USB devices must have Endpoint 0 configured as a control endpoint.

USB Data Transfer Type

USB 2.0 supports four types of data transfers: Control, Bulk, Interrupt and Isochronous.

- **Control** transfer is used to configure a device at the time of plug-in and can be used for other device specific purposes, including control of other pipes on the device.
- **Bulk** data transfers are used when the data is generated or consumed in relatively large, bulky quantities.
- **Interrupt** data transfers are used for timely, and reliable, delivery of data. For example, characters or coordinates with human perceptible echo or feedback response characteristics.
- **Isochronous** data transfers occupy a pre-negotiated amount of USB bandwidth with pre-negotiated delivery latency (also called streaming real-time transfers).

Human Interface Device Class

The human interface device (HID) class was one of the first USB classes to be supported under Windows and other operating Systems. On PCs running Windows, applications can communicate with HID using the generic HID drivers built into the operating system such as kbdhid.sys and mouhid.sys.

The **HID** class consists of devices that are used by humans to control the operation of computer systems. Typical examples of **HID** class devices include:

- Keyboards and pointing devices—for example, standard mouse devices, trackballs, and joysticks.
- Front-panel controls—for example: knobs, switches, buttons, and sliders.
- Controls that might be found on devices such as telephones, VCR remote controls, games or simulation devices—for example: data gloves, throttles, steering wheels, and rudder pedals.

Microchip's USB Solution

Microchip USB Firmware Framework can be used to create new USB applications. It can be thought of as a reference design project, containing the necessary firmware for USB operation and provides a placeholder for the user's code.

The USB Firmware Framework also provides a set of modular firmware interfaces that handle most of the work for implementing USB communications. Each firmware reference project is written to have a cooperative multitasking environment; thus, no blocking functions should be implemented in user code. The files are tightly interdependent, and pass information between themselves during compile time to create the complete USB configuration.

The PIC18 with USB peripheral device families contains a full-speed and low-speed compatible USB Serial Interface Engine (SIE) that allows fast communication between any USB host and PIC microcontroller. The SIE can be interfaced directly to the USB, utilizing the internal transceiver.

For complete information of functions and stack please refer PICDEM FS USB demo board's user guide and PIC18 Data sheet available on Microchip's website.

Overview: Design

For demonstration purpose a 4 by 4 keypad has been interfaced in current version. Any other type of keyboard with different numbers of rows and columns can be interfaced by writing the appropriate scanning routines for the same. The keypad works in two different modes as following.

Normal Mode:

In normal mode keypad works as simple keyboard. It sends alpha-numeric keys to host whenever a key is pressed. Pressing 'F' (Select) sends the keypad in "Multimedia Mode". Again pressing 'F' brings keypad back in normal mode.

1	2	3	C
4	5	6	D
7	8	9	E
A	0	B	F **Select**

Figure: Alpha-Numeric Keyboard

Multi Media Mode:

In this mode keypad supports following types of tasks:

1. Volume Control (UP, DOWN and MUTE): [first 3 keys]
2. System Control (System SLEEP and WAKE UP): [2 keys towards RHS]
3. AL function (Media player, Calculator, Browser, Email): [3 keys in 3rd row and 1st in 2nd row]
4. Browser Control (Forward, Backward, Refresh): [3 keys in last row]
5. launch 'Search' and 'Favorite': [2 keys in 2nd row]

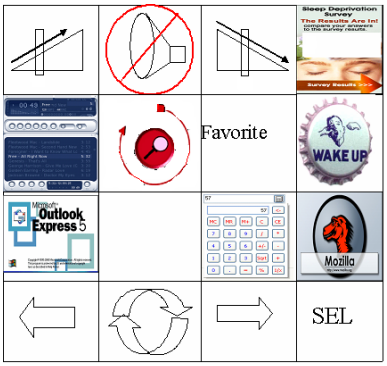


Figure: Multimedia functions

Implementation

In this implementation of USB/HID keyboards with enhanced features the following design guidelines have been followed.

1. Two HID interfaces has been implemented in a USB composite device fashion, where one interface is used for a standard QWERTY keyboard report that's identical to the keyboard boot protocol report, and the other interface is used for the new, enhanced functionality.
2. Second interface is a consumer control HID usage that implements Volume Up, Volume Down, Mute, and WWW Home buttons using Report ID 1.
3. Keyboard is equipped with Power Management buttons (Sleep, Wake-up, or Power Down buttons), which is implemented in Report ID 2 of second interface.

Interfaces and Endpoint Diagram

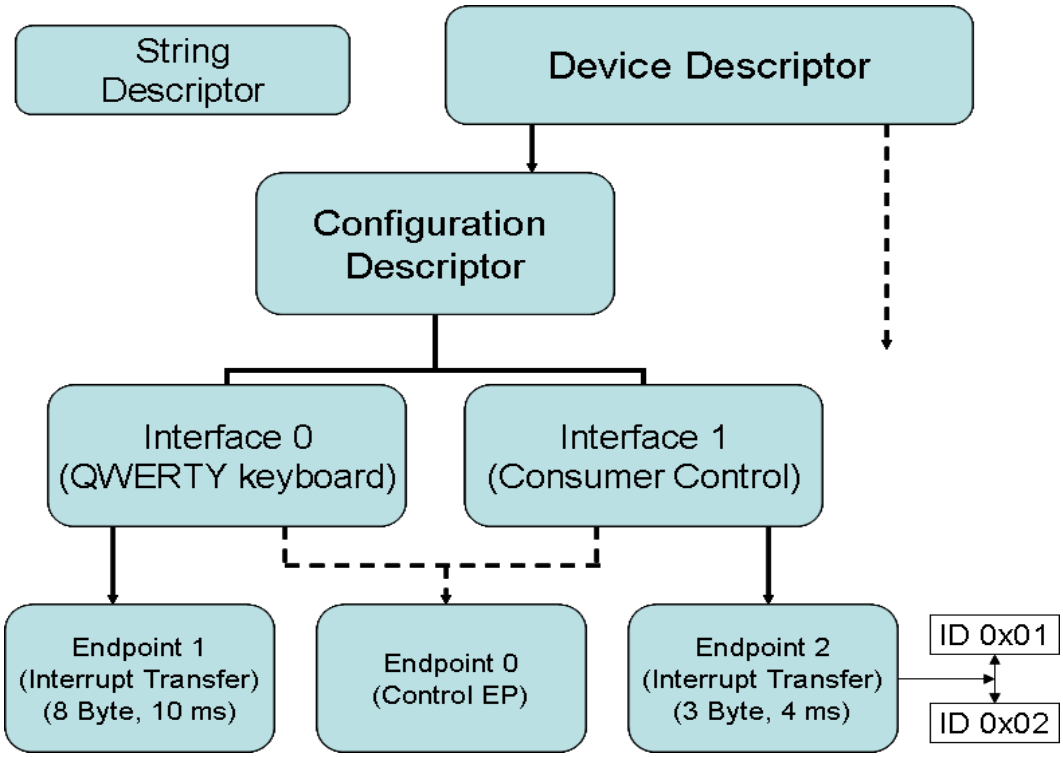


Figure: Available Endpoints and Interfaces in device

USB Descriptors

1. Device Descriptor:

Device descriptor is very first data structure which includes information such as what USB revision the device complies with, the Product and Vendor IDs used to load the appropriate drivers and the number of possible configurations the device can have

```
/* Device Descriptor */
rom USB_DEV_DSC device_dsc=
{
    Sizeof (USB_DEV_DSC),   // Size of this descriptor in bytes: 18
    DSC_DEV,                // DEVICE descriptor type: 01
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code; Defined in Interface DSC
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    EP0_BUFF_SIZE,          // Max packet size for EP0: 0x08
    0x04D8,                 // Vendor ID: Microchip
    0x0111,                 // Product ID: for this product
    0x0001,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x00,                   // Device serial number string index
    0x01                    // Number of possible configurations
};
```

2. Configuration Descriptor:

Configuration descriptor specifies enabled configuration incase the device has more than one configurations available. Amount of power this particular configuration uses, if the device is self or bus powered and the number of interfaces it has.

A typical configuration descriptor consists of at least one configuration descriptor, one or more interface descriptors and one or more endpoint descriptors.

Here is an example of configuration descriptor's definition with two interfaces for enhanced keyboard implementation

```
#define CFG01 rom struct            \
{   USB_CFG_DSC     cd01;           \
    USB_INTF_DSC    i00a00;         \
    USB_HID_DSC     hid_i00a00;     \
    USB_EP_DSC      ep01i_i00a00;   \
    USB_INTF_DSC    i01a00;         \
    USB_HID_DSC     hid_i01a00;     \
    USB_EP_DSC      ep02i_i01a00;   \
} cfg01
```

The current implementation of configuration descriptor is as following.

```
/* Configuration Descriptor */
    Sizeof (USB_CFG_DSC),    // Size of this descriptor in bytes: 09
    DSC_CFG,                 // CONFIGURATION descriptor type: 02
    Sizeof (cfg01),          // Total length of data for this cfg
    2,                       // Number of interfaces in this cfg: 02
    1,                       // Index value of this configuration
    0,                       // Configuration string index
    _DEFAULT|_RWU,           // Attributes, see usbdefs_std_dsc.h
    50,                      // Max power consumption (2X mA): 100 mA
```

3. Interface Descriptor:

A configuration's interface descriptor contains information about the endpoints the interface supports. Each configuration must support at least one interface. In the current implementation two interfaces has been implemented in following manner.

Keyboard functionality is supported by interface 1.

```
/* Interface 1 Descriptor: QWERTY Keyboard Function */
    sizeof(USB_INTF_DSC),    // Size of this descriptor in bytes: 09
    DSC_INTF,                // INTERFACE descriptor type: 02
    0,                       // Interface Number
    0,                       // Alternate Setting Number
    1,                       // Number of endpoints in this interface
    HID_INTF,                // Class code: 0x03
    BOOT_INTF_SUBCLASS,      // Subclass code: 0x01
    HID_PROTOCOL_KEYBOAD,    // Protocol code: 0x01
    0,                       // Interface string index
```

Multimedia and AL function are supported by interface 2, which supports two report formats.

```
   /* Interface 2 Descriptor: Multimedia and AL support */
    sizeof(USB_INTF_DSC),   // Size of this descriptor in bytes: 09
    DSC_INTF,               // INTERFACE descriptor type: 02
    1,                      // Interface Number
    0,                      // Alternate Setting Number
    1,                      // Number of endpoints in this interface
    HID_INTF,               // Class code: 0x03
    NO_INTF_SUBCLASS,       // Subclass code should come here, if any
    HID_PROTOCOL_NONE,      // Protocol code should come here, if any
    1,                      // Interface string index
```

4. Class Specific Descriptor:

A class specific descriptor tells host about the current USB class specs version, country code, size of input and output report which would be used during the data transfer from device to host or vice versa. The descriptor below implements HID class.

```
/* HID Class-Specific Descriptor */
    sizeof(USB_HID_DSC),    // Size of this descriptor in bytes
    DSC_HID,                // HID descriptor type
    0x0101,                 // HID Spec Release Number in BCD format
    0x00,                   // Country Code (0x00 for Not supported)
    HID_NUM_OF_DSC,         // Number of class descriptors,
    DSC_RPT,                // Report descriptor type
    sizeof(hid_rpt01),      // Size of the report 1 descriptor
    sizeof(hid_rpt02),      // Size of the report 2 descriptor
```

5. End point Descriptor:

The endpoint descriptor identifies the transfer type and direction, as well as some other specifics for the endpoint. There may be many endpoints in a device and endpoints may be shared in different configurations.

Keyboard endpoint uses interrupt transfer mechanism to transfer the data from device to host. Direction of this endpoint is IN and size is 8 Bytes. The descriptor also specifies the time interval which is used by host to poll the endpoint. In this case the polling interval is 10 ms.

```
/* Endpoint 1 Descriptor: QWERTY Function Endpoint */

sizeof(USB_EP_DSC),DSC_EP,_EP01_IN,_INT,HID_INT_IN_EP01_SIZE,0x0A,
```

Multimedia function endpoint also uses interrupt transfer mechanism to transfer the data from device to host. Direction of this endpoint is IN and size is 3 Bytes. The descriptor also specifies the time interval which is used by host to poll the endpoint. In this case the polling interval is 4 ms.

```
/* Endpoint 2 Descriptor: Multimedia Function Endpoint */

sizeof(USB_EP_DSC),DSC_EP,_EP02_IN,_INT,HID_INT_IN_EP02_SIZE,0x04,
```

Sample Report Descriptors for USB HID keyboard

First HID Interface Report Descriptor (keyboard report identical to the boot protocol):

```
rom struct{byte report[HID_RPT01_SIZE];}hid_rpt01={
    0x05, 0x01,    /* Usage Page (Generic Desktop)              */
    0x09, 0x06,    /* Usage (Keyboard)                          */
    0xA1, 0x01,    /* Collection (Application)                  */
    0x05, 0x07,    /*      Usage page (Key Codes)               */
    0x19, 0xE0,    /*      Usage minimum (224)                  */
    0x29, 0xE7,    /*      Usage maximum (231)                  */
    0x15, 0x00,    /*      Logical minimum (0)                  */
    0x25, 0x01,    /*      Logical maximum (1)                  */
    0x75, 0x01,    /*      Report size (1)                      */
    0x95, 0x08,    /*      Report count (8)                     */
    0x81, 0x02,    /*      Input (data, variable, absolute)     */
    0x95, 0x01,    /*      Report count (1)                     */
    0x75, 0x08,    /*      Report size (8)                      */
    0x81, 0x01,    /*      Input (constant)                     */
    0x95, 0x06,    /*      Report count (6)                     */
    0x75, 0x08,    /*      Report size (8)                      */
    0x15, 0x00,    /*      Logical minimum (0)                  */
    0x25, 0x65,    /*      Logical maximum (101)                */
    0x05, 0x07,    /*      Usage page (key codes)               */
    0x19, 0x00,    /*      Usage minimum (0)                    */
    0x29, 0x65,    /*      Usage maximum (101)                  */
    0x81, 0x00,    /*      Input (data, array)                  */
    0xC0};         /* End Collection                            */
```

The second HID interface will issue one of two possible input reports, each distinguished by a unique Report ID, depending on what control the user is manipulating.

```
rom struct{byte report[HID_RPT02_SIZE];}hid_rpt02={
    0x05, 0x0c,                    // USAGE_PAGE (Consumer Devices)
    0x09, 0x01,                    // USAGE (Consumer Control)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x85, 0x01,                    //   REPORT_ID (1)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x01,                    //   LOGICAL_MAXIMUM (1)
    0x75, 0x01,                    //   REPORT_SIZE (1)
    0x95, 0x10,                    //   REPORT_COUNT (16)
    0x09, 0xe2,                    //   USAGE (Mute) 0x01
    0x09, 0xe9,                    //   USAGE (Volume Up) 0x02
    0x09, 0xea,                    //   USAGE (Volume Down) 0x03
    0x09, 0xcd,                    //   USAGE (Play/Pause) 0x04
    0x09, 0xb7,                    //   USAGE (Stop) 0x05
    0x09, 0xb6,                    //  USAGE (Scan Previous Track) 0x06
    0x09, 0xb5,                    //   USAGE (Scan Next Track) 0x07
    0x0a, 0x8a, 0x01,              //   USAGE (Mail) 0x08
    0x0a, 0x92, 0x01,              //   USAGE (Calculator) 0x09
    0x0a, 0x21, 0x02,              //   USAGE (www search) 0x0a
    0x0a, 0x23, 0x02,              //   USAGE (www home) 0x0b
    0x0a, 0x2a, 0x02,              //   USAGE (www favorites) 0x0c
    0x0a, 0x27, 0x02,              //   USAGE (www refresh) 0x0d
```

```
    0x0a, 0x26, 0x02,                   //   USAGE (www stop) 0x0e
    0x0a, 0x25, 0x02,                   //   USAGE (www forward) 0x0f
    0x0a, 0x24, 0x02,                   //   USAGE (www back) 0x10
    0x81, 0x62,                         //   INPUT (Data,Var,Abs,NPrf,Null)
    0xc0,
// System Control Descriptor
    0x05, 0x01,    /* Usage Page (Generic Desktop)                  */
    0x09, 0x80,    /* Usage (System Control)                        */
    0xA1, 0x01,    /* Collection (Application)                      */
    0x85, 0x02,    /* Report ID 0x02 [SYSTEM CTRL]                  */
    0x19, 0x82,    /*      Usage minimum (System Sleep)             */
    0x29, 0x83,    /*      Usage maximum (System Wake up)           */
    0x95, 0x02,    /*      Report count (2)                         */
    0x81, 0x06,    /*Input (data, variable, relative, Preferred)   */
    0x95, 0x06,    /*      Report count (6)                         */
    0x81, 0x01,    /*Input (Constant)                              */
    0xC0           /*End Collection                                */
};
```

Keyboard Data format

First HID interface will issue 8-byte input reports that are identical to the standard keyboard boot protocol report as documented in the HID Class Version 1.11 specification.

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Right GUI	Right ALT	Right Shift	Right CTRL	Left GUI	Left ALT	Left Shift	Left CTRL
1	Reserved							
2	Keyboard Page Usage							
3	Keyboard Page Usage							
4	Keyboard Page Usage							
5	Keyboard Page Usage							
6	Keyboard Page Usage							
7	Keyboard Page Usage							

Second HID interface will issue one of two possible input reports, each distinguished by a unique Report ID, depending on what control the user is manipulating.

Report ID 1

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Report ID = 0x01							
1	AL Email				AL Media Player	VOL Down	VOL UP	VOL Mute
2	Browser Back	Forward		Refresh	AL Favorite	AL Browser	AL Search	AL Calc

Report ID 2

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Report ID = 0x02							
1							System WAKE UP	System SLEEP
2								

Firmware Templates

Below is a generic template of main routine. USB driver service and User application should be written in co-operative manner.

Main Routine:

```
Void main (void)
{
    InitializeSystem(); //USB device and user initialization
    while(1)
    {
        USBTasks();     // USB Driver Service Routine

        KeypadScan();   // Keypad scanning routine for key press

      /*Load the keyboard buffer with the keyboard HID scan code*/
      if(SendFlag == FALSE)
            {
            if(FunctionFlag == FALSE)
                  kb_buffer[3] = keyval;
            }
      else
            {
            if(FunctionFlag == FALSE)
                  kb_buffer[3] = 0x00;
            }

      if(FunctionFlag == FALSE)
            Send2Kboard();    //Send as keyboard data
      else
            {
                  if(ctrl_buffer[0] == ALCtrlID)
                        Send2ALCtrl();  // send Multimedia and AL data
                  if(ctrl_buffer[0] == SysCtrlID)
                        Send2SysCtrl(); // send System ctrl data
            }

    }//end while
}//end main
```

Routine for alpha-numeric data

```
void Send2Kboard(void)
{
      if(!mHIDTxIsBusy())
        {
            HIDTxReport(kb_buffer,8);
                  SendFlag = TRUE;
                  kb_buffer[3] = 0x00;
        }
}
```

Routine for multimedia function and AL function

```
void Send2ALCtrl(void)
{
        if(!mHIDTx2IsBusy())
         {
             HIDTx2Report(ctrl_buffer,3);
             SendFlag = TRUE;
                   ctrl_buffer[1] = 0x00;
                   ctrl_buffer[2] = 0x00;
         }
}
```

Routine for system control function

```
void Send2SysCtrl(void)
{
        if(!mHIDTx2IsBusy())
         {
             HIDTx2Report(ctrl_buffer,2);
             SendFlag = TRUE;
                   ctrl_buffer[1] = 0x00;
                   ctrl_buffer[2] = 0x00;
         }
}
```

Enhanced Keyboard: Firmware Directory structure

The Multimedia Keyboard project source code is broken up into following sub folders under MultimediaKboard as shown below.

Microchip stack application source files in the Root directory **MultimediaKboard**:

MCHPUSB.mcp	MPLab project file for enhanced keyboard
Main.c	The main file for enhanced keyboard
Io_cfg.h	Contains application specific port configuration information
18f4550.lkr	Linker script for PIC18F4550. Specific to application
autofiles	Subfolder - Microchip USB descriptors and configurations
system	Subfolder - Microchip USB stack chapter 9 specifications
user	Subfolder – User specific routines and interfacing code
Docs	Subfolder – Related documents
_output	Subfolder – compiler output goes here

Microchip USB stack's source files in the **System** Subdirectory:

Hid.c, .h	USB HID communication and interface code
TypeDefs.h	Constants and type definitions used for USB application
usb.h	USB header file
Usb9.c, .h	Performs USB Protocol and command handling
Usb_compile_time_val.h	USB end point size validation file.
Usbctrltrf.c, .h	USB transfer control handler services
usbdefs_ep0_buff.h	USB end point buffer and structure definition file
usbdefs_std_dsc.h	USB standard descriptor structure file
Usbdrv.c, .h	USB operation handing code and definitions
Usbdsc.c. .h	contains the USB descriptor information and defination
usbmap.c, .h	USB memory manager; compile-time memory allocator for the USB endpoints

Microchip USB stack's configurations and descriptors in the **autofiles** Subdirectory:

usbcfg.h	USB End point configuration file
Usbdsc.c. .h	contains the USB descriptor information and definition

User Specific routines and interfacing code in **user** subdirectory:

Multimedia_kboard.c,h	Specific routines and interfacing code

Resource Utilization and Future direction:

Here is a summary of resource utilization. The values are approximate for a typical implementation however it can vary depending on application and the way of implementation.

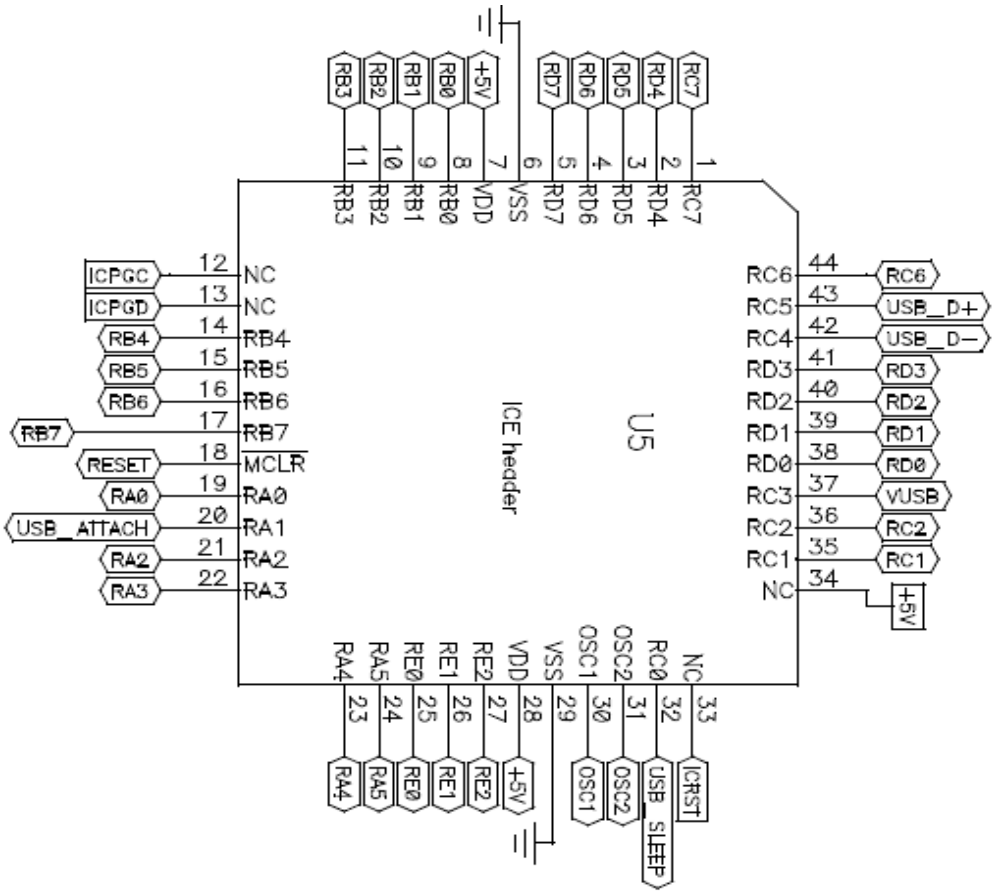
Device Firmware → Resources	USB Multimedia keyboard
Data RAM	350 Bytes
Program Memory	4.5 KB
I/O s	Implementation Specific
Power Consumption (Maximum)	< 500 mW

To incorporate additional functionalities one should consider implementation of more interfaces thus more endpoints in order to meat the requirement. If the power drawn by the device is likely to exceed more than 100 mA, self powered option should be used to power up the device instead of Bus powered option.

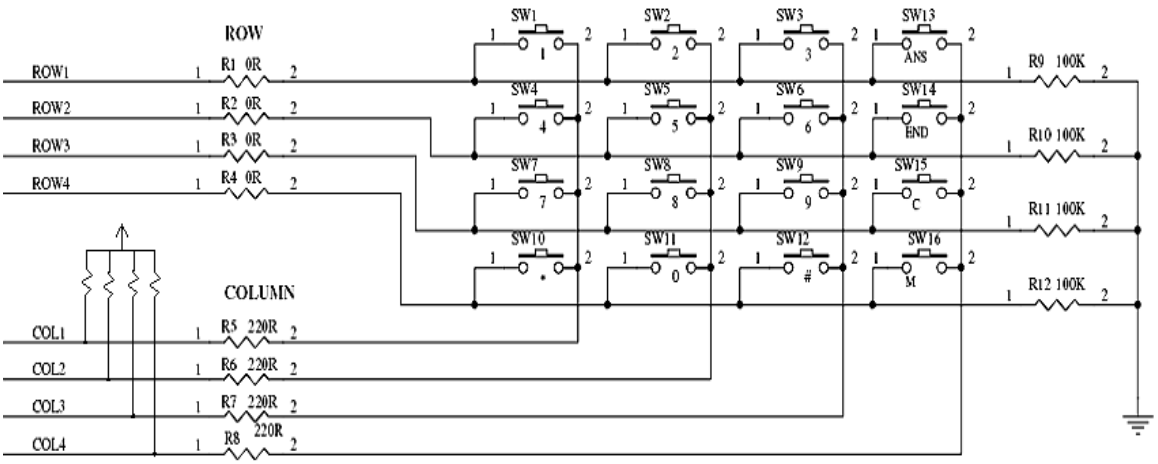
Appendix:

A. Schematic and Diagrams.

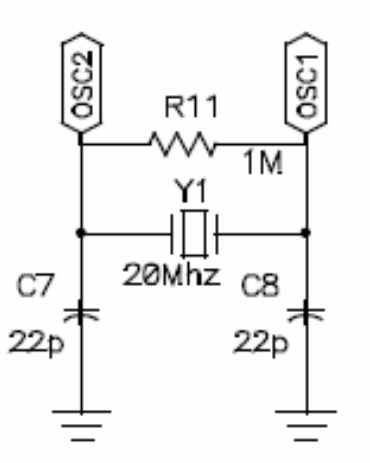
1. Microcontroller's Pin out diagram



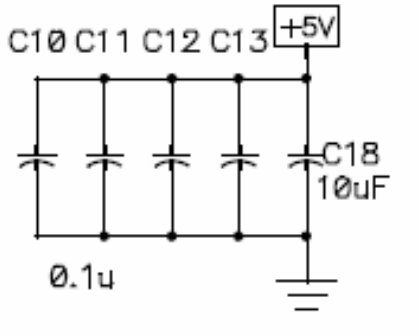
2. 4 by 4 keypad interfacing



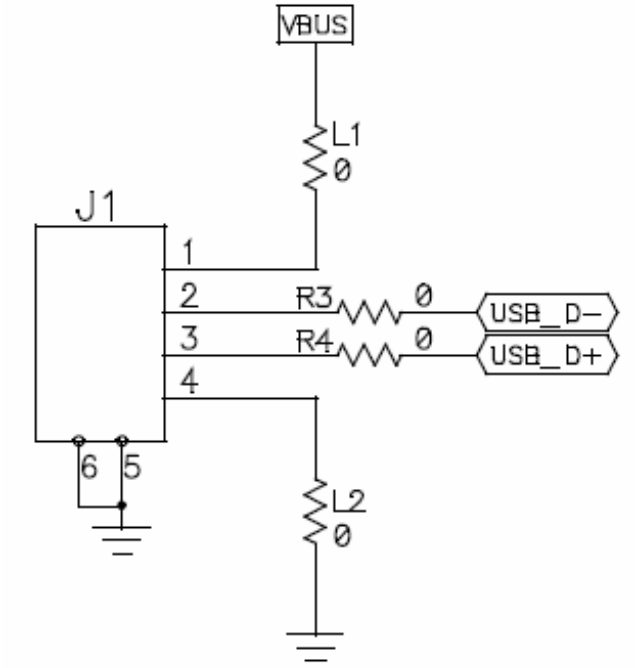
3. XTAL oscillator (20 MHz)



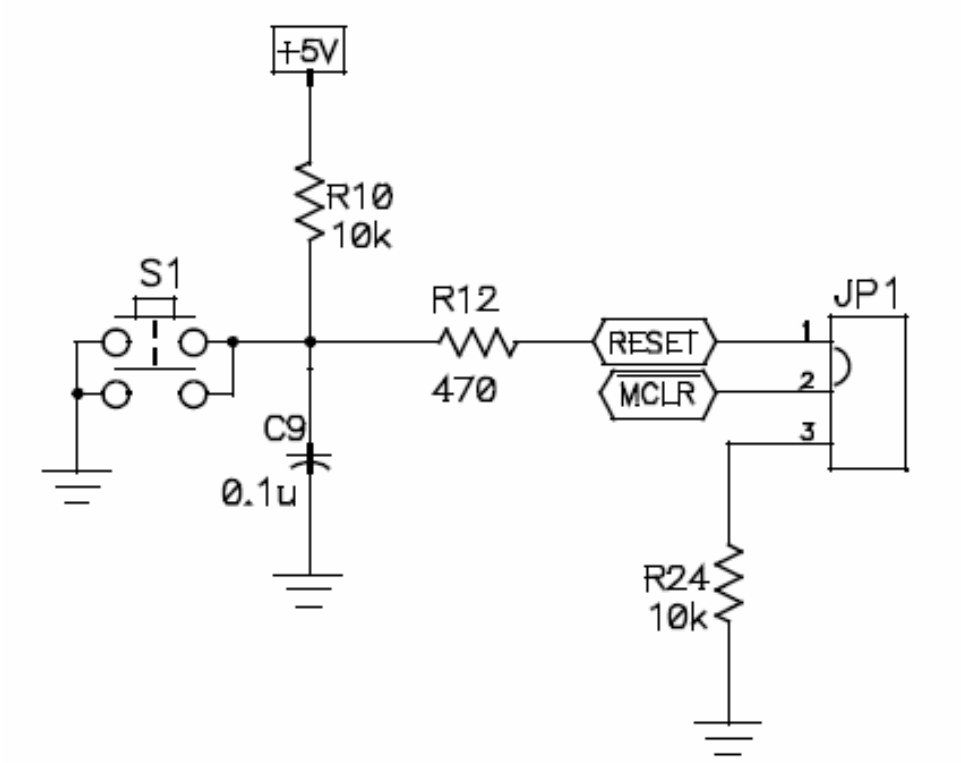
4. Power Supply Filter (Decoupling capacitor)



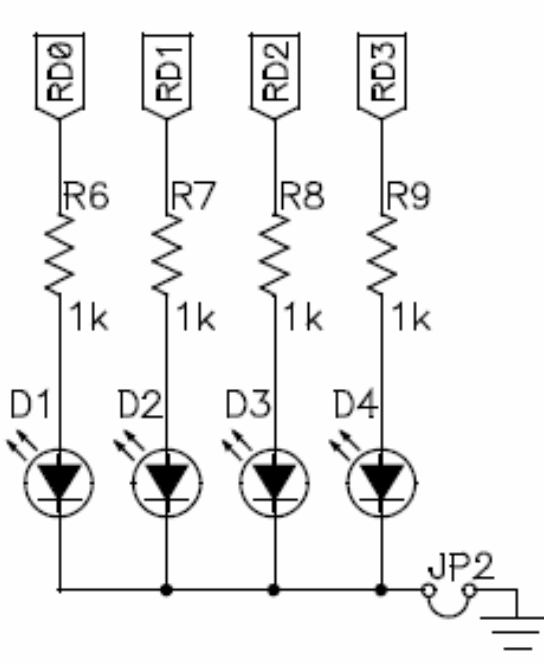
5. USB Connector



6. Reset Switches



7. LED indicators on board



B. Enumeration of Device under Windows OS

Under window when devices gets properly enumerated it gets listed under following sections in device manager.

Under **Human Interface Devices** (HID):

HID-Compliant Consumer Control Device HID-Compliant device USB Human Interface Device USB Human Interface Device

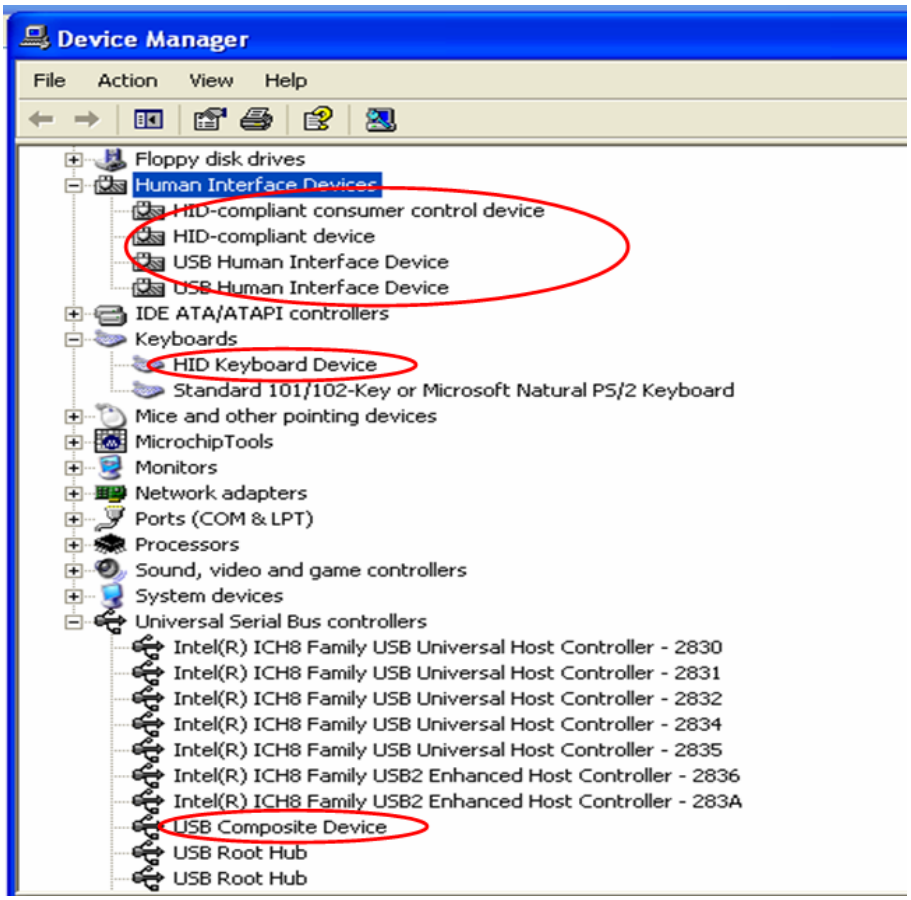
Under **Keyboards**:

HID Keyboard Device

Under **Universal Serial Bus Controllers**:

USB Composite Device

Example:



References

- "Universal Serial Bus Revision 2.0 Specification" http://www.usb.org
- USB Complete: Everything You Need to Develop Custom USB Peripherals by Jan Axelson
- "USB Device Class Definition for Human Interface Devices (HID) Version 1.11" http://www.usb.org
- 'Microchip Stack for the USB HID Protocol" (AN***) http://www.microchip.com/usb
- "PIC18F2455/2550/4455/4550 full-speed USB 2.0 PIC® microcontrollers Data Sheet" (DS39632C) http://www.microchip.com/datasheet
- "PICDEM™ FS USB Demonstration Kit User's Guide" (DS51526A) http://www.microchip.com/picdemfs
- "Enhanced keyboard and windows" from windows hardware development central by Microsoft Inc. http://www.microsoft.com/whdc/device/input/w2kbd.mspx