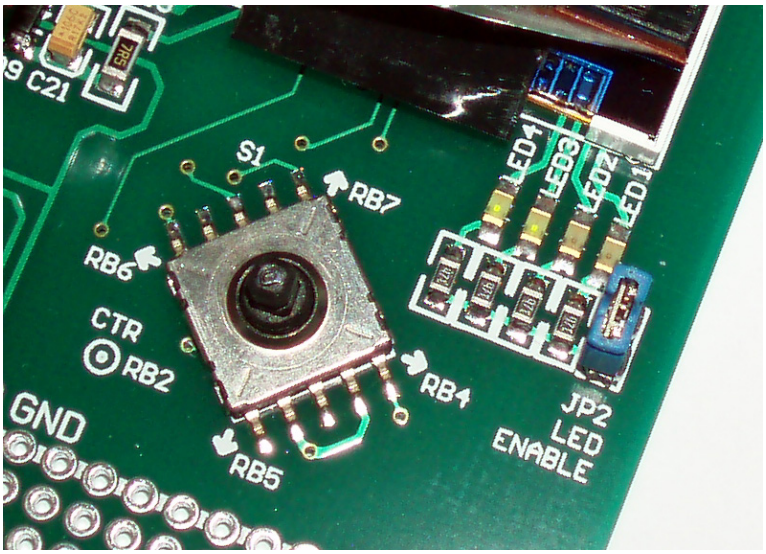
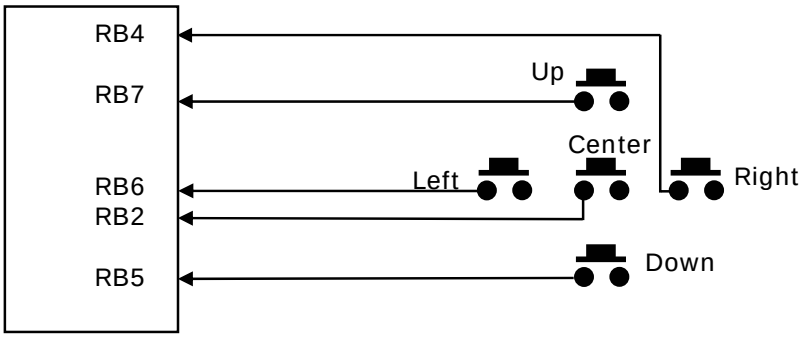
Joystick: API for 5-way navigator joystick

Program in this application note demonstrates the API for joystick S1. It is a 5-way navigator joystick in the sense that, it is possible to click it up, down, right, left, and center. Its position is highlighted in the picture below.



Its simplified wiring diagram is shown here. The actual schematic is more complicated though, but the idea behind is the same.



Remarks: keys' pins short to ground to read 0 when pressed

There are many references from the Internet on this topic. However, the best keyboard/keypad driver I've seen is from Jean J. Labrosse's book, Embedded System Building Blocks, 2nd edition. The keypad module presented on Jean's book concentrates on matrix keyboard. The following features are available[1]:

[1] Chapter 3, Keyboards, Embedded System Building Blocks, 2nd edition, by Jean J. Labrosse

- Scan any keyboard arrangement from 3x3 to an 8x8 key matrix
- Provides buffering with user configurable buffer size
- Supports auto-repeat
- Keeps track of how long a key has been pressed
- Allows up to three Shift keys

Because our navigator joystick has been wired in direct I/O configuration, a simplified version would be presented here with the following features:

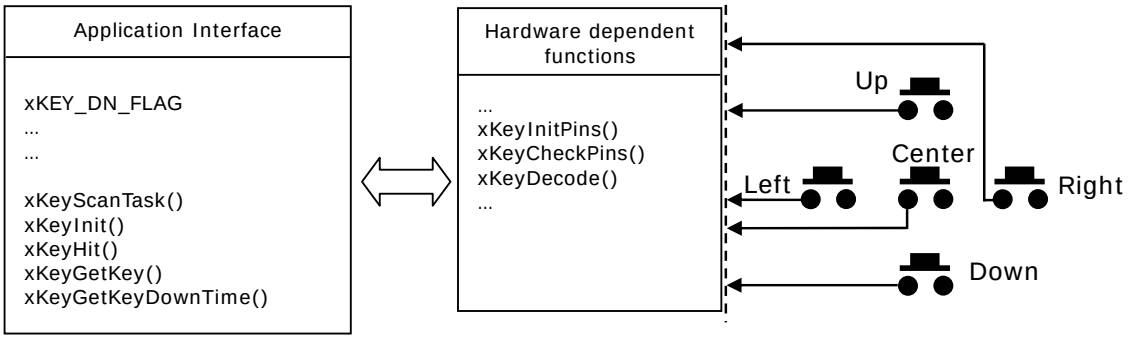
- Scan individual switch
- Provides buffering with user configuration buffer size
- Supports auto-repeat
- Keeps track of how long a key has been pressed

The source code of this program is found under our web site with title JoystickDemo
http://www.techtoys.com.hk/PIC_boards/PIC18-4455-STK1/pic18-4550-stk1.htm

Joystick functions have been included in form of a software module.

Listing 1 shows the xKey.h file. It would be enough for us to make use of all functions provided by the xKey.c module by learning just the constants and function prototypes defined under xKey.h, instead of going through the details of xKey.c.

Module flow diagram is shown in Figure below.



```
/* Pin-out definition for PIC18LF-4550-STK1 board */

typedef     unsigned char     BOOLEAN;                    (1)

#define     xKEY_DN           PORTBbits.RB5               (2)
#define     xKEY_RT           PORTBbits.RB4               (3)
#define     xKEY_LT           PORTBbits.RB6               (4)
#define     xKEY_UP           PORTBbits.RB7               (5)
#define     xKEY_CTR          PORTBbits.RB2               (6)

#define     xKEY_DIR_DN       TRISBbits.TRISB5            (7)
#define     xKEY_DIR_RT       TRISBbits.TRISB4            (8)
#define     xKEY_DIR_LT       TRISBbits.TRISB6            (9)
#define     xKEY_DIR_UP       TRISBbits.TRISB7            (10)
#define     xKEY_DIR_CTR      TRISBbits.TRISB2            (11)

/* General definition */
#define  FALSE                      0                    (12)
#define  TRUE                       1                    (13)

/* Key definitions */
#define xKEY_DN_FLAG                1                    (14)
#define xKEY_RT_FLAG                2                    (15)
#define xKEY_LT_FLAG                3                    (16)
#define xKEY_UP_FLAG                4                    (17)
#define xKEY_CTR_FLAG               5                    (18)

/* Buffer size, delay constants etc */
#define  xKEY_BUF_SIZE              3                    (19)
#define  xKEY_RPT_DLY               5                    (20)
#define  xKEY_RPT_START_DLY         30                   (21)
#define  xKEY_SCAN_TASK_DLY         20                   (22)

/* extern definition */
#ifdef  xKEY_GLOBALS                                     (23)
#define xKEY_EXT                                         (24)
#else
#define xKEY_EXT extern                                  (25)
#endif

/* Auto-repeat enable bit */
xKEY_EXT    BOOLEAN     xKeyRptEn;                       (26)

/* API functions */
void                xKeyScanTask(void);                  (27)
void                xKeyInit(void);                      (28)
BOOLEAN             xKeyHit(void);                       (29)
unsigned char       xKeyGetKey (void);                   (30)
unsigned int        xKeyGetKeyDownTime(void);            (31)

void                xKeyInitPins(void);                  (32)
BOOLEAN             xKeyCheckPins(void);                 (33)
unsigned char       xKeyDecode(void);                    (34)
```

Listing 1 **xKey.h file**

Line (1) is the type definition for BOOLEAN for C18 compiler. Other compilers may have a 'bit' data type for small microcontroller. However, C18 only has the smallest 8-bit data for variables so we 'typedef' it BOOLEAN for TRUE/FALSE variable. If there is another compiler with a data type 'bit', we just need to modify the code

typedef bit BOOLEAN

to maintain the code so there is no need to inspect every line for BOOLEAN.

Line (2) – (6) provides hardware definition matching the development board. Other compilers may have a different way of definition/syntax. For example, 'sbit' is used for Keil C for 8051 families to define a pin name. The same joystick driver has been ported to 8051 for KEIL C compiler as well. Interested parties may search at our web site under 8051 for details. If you want to use other pins for your own hardware, this is the place to modify.

Line (7) – (11) defines the pin direction registers. These are specific to Microchip PIC and C18 compiler as well. These should match the definitions from line (2) to (6) above.

Line (12) & (13) defines the keyword FALSE and TRUE for portability.

Line (14) – (18) defines arbitrary constants for individual key. Constant xKEY_DN_FLAG (say) would be used in the main() program for detection of the DOWN key press.

Line (19) defines the buffer size of the Joystick buffer. A cyclic buffer is implemented inside xKey.c. The direct consequence of using a buffer is that, it is possible to postpone reading the joystick without losing keystrokes if the microcontroller is handling other tasks. The size of the buffer depends on the application requirement. For our simple demonstration, a buffer size of 3 is good enough. It is possible to assign a large buffer of 50; unfortunately, key buffer is occupying RAM space so it fights for RAM resources with other tasks as well.

Line (20) defines the number of scan times before auto-repeat executes again, i.e. it is the rate of auto-repeat. Scan time measured in units of xKEY_SCAN_TASK_DLY defined in Line (22).

Therefore, the actual time for auto-repeat in our case is 100 ms.

xKEY_RPT_DLY * xKEY_SCAN_TASK_DLY ms, which is 5*20ms

Line (21) defines the number of scan times before auto-repeat function started. Again, its unit is measured in xKEY_SCAN_TASK_DLY.

Remarks: The pre-requisite of using auto-repeat is to set xKeyRptEn to TRUE during run time.

Line (22) is the number of milliseconds between keyboard scans. It is an important parameter for key debounce. We need debounce because switches are not perfect. They do not generate a clear-cut 1 or 0 when they are pressed or released. A normal person presses a key longer than 20ms, so a key debounce delay of 20ms is usually good enough. This value is also dependent on the quality of switch as well.

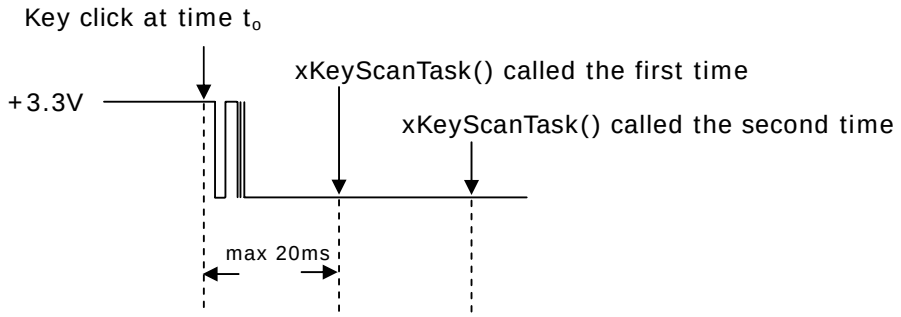
Line (23) - (25) takes care of extern keyword for xKey.c and the application program (Joystick_main.c). If we need a global variable that would be accessed by the application program, we need to prefix the variable with xKEY_EXT, like the case in xKeyRptEn.

Line (26) defines a global variable for enable/disable the auto-repeat feature. xKeyRptEn defaults to FALSE on xKeyInit() which is the module initialization.

Here come the API functions:

Line (27)

xKeyScanTask() void xKeyScanTask(void);
This is a direct modification from the original KeyScanTask(void *data) created by Jean J. Labrosse in his book Embedded System Building Blocks, 2nd edition. Because we do not have RTOS yet, this function has been made public, and the function OSTimeDlyHMSM() in the original version has also been removed. This function is the heart of the keyboard module. It should be called in a periodic manner with the rate defined by xKEY_SCAN_TASK_DLY. In this case, key debounce is taken care of during the xKEY_SCAN_TASK_DLY period. The principle is illustrated below. Suppose at any time t_o , the key UP is clicked, after some time (max 20ms defined by xKEY_SCAN_TASK_DLY), xKeyScanTask() would be called for the first time. xKeyScanTask() would record the state of the key and just exit. A normal key press would last longer than 20 ms. When xKeyScanTask() is called the second time, algorithm inside xKeyScanTask() will decide if that key is still pressed. If “yes”, the key code would be decoded and placed inside the Joystick buffer for future read; otherwise, no key code would be inserted as it must have been noise or not a real key stroke. It has been assumed that no key bounce will last longer than 20ms! If you’ve got a key like that, just through it away.
Example : Please refer to Joystick_main.c



Key Debounce (drawing not to scale, key bounce should be short)

Line (28)

xKeyInit() void xKeyInit(void);
xKeyInit() is the initialization code for the module. It must be called before using any of the other functions. xKeyInit() is responsible for initializing internal variables used by the module and hardware ports as well.
Example : Please refer to Joystick_main.c

Line (29)

xKeyHit() BOOLEAN xKeyHit(void);
xKeyHit() allows your application to determine if a key has been pressed. It return TRUE if a key was pressed, and FALSE otherwise. **Arguments:** none

Return Value: TRUE or FALSE

Example : Please refer to Joystick_main.c

Line (30)

xKeyGetKey()
unsigned char xKeyGetKey(void);

xKeyGetKey() is called by the application to obtain a scan code from the Joystick buffer, in our case, scan code has been defined by:

```
/* Key definitions */
#define xKEY_DN_FLAG        1
#define xKEY_RT_FLAG        2
#define xKEY_LT_FLAG        3
#define xKEY_UP_FLAG        4
#define xKEY_CTR_FLAG       5
```

Arguments: none
Return Value: xKEY_XX_FLAG, or 0xFF if there is an error

Warnings: This function should be called frequent enough before key buffer overflow.

Example : Please refer to Joystick_main.c

Line (31)

xKeyGetDownTime()
unsigned int xKeyGetDownTime(void);

xKeyGetKeyDownTime() returns the amount of time (in msec) that a key has been pressed.

Arguments: none
Return Value: The amount of time that the current key is being pressed.

Warnings: The key down time is not cleared when the pressed key is released

Example: please request if an example is required.

Line (32) – (34) are hardware dependent functions. They are not explicitly called in user application. Please refer to the source code for details.

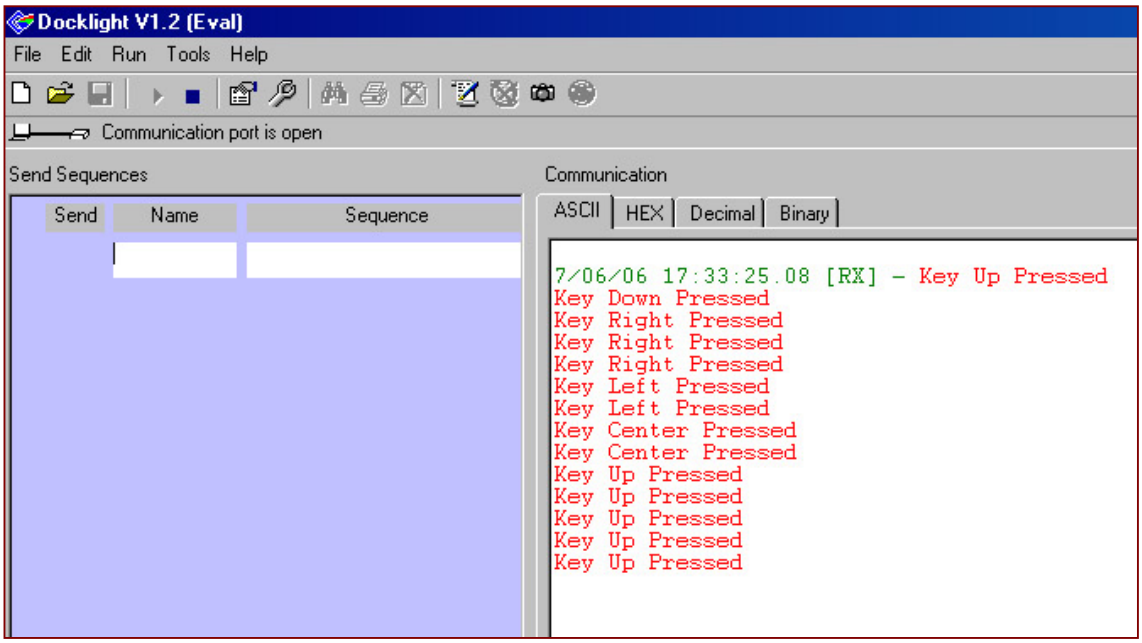
Listing 'Joystick_main.c' here shows the demo application.

```
#include <p18cxxx.h>
#include "delay.h"
#include "xKey.h"
#include <usart.h>

#pragma config FOSC   = HS              // HS oscillator
#pragma config CPUDIV = OSC4_PLL6       // OSC postscaler of 4 => CPU clock at 5MHz (20MHz/4)
#pragma config PWRT   = ON              // PowerUp timer ON
#pragma config WDT    = OFF             // watchdog timer off
#pragma config LVP    = OFF             // low level voltage OFF
#pragma config BOR    = OFF             // brown out detect OFF
#pragma config DEBUG  = ON              // Debug ON
#pragma config VREGEN = OFF             // Off 3.3V internal regulator
#pragma config PBADEN = OFF             // PORTB<4:0> as digital IO on reset
#pragma config MCLRE  = ON              // Enable MCLRE for reset function

void main(void)
{
        unsigned char keyCode, counter=0;
        ADCON1 = 0x0F;                  //PORTA all digital operation
        LATD   = 0x00;                  //use LED1 - LED4 as a simple counter
        TRISD  = 0x00;                  //configure PORTD for output
        OpenUSART(      USART_TX_INT_OFF &
                        USART_RX_INT_OFF &
                        USART_ASYNCH_MODE &
                        USART_EIGHT_BIT &
                        USART_CONT_RX &
                        USART_BRGH_HIGH, 64);
        //Fosc = 5 MHz, SPBRG = 64 => baud rate=5MHz/(16*(64+1))=4,808 bps

        putrsUSART("Joystick demo:\n");
        xKeyInit();                     //initialize the keypad module before using it

        for(;;){
                DelayMs(xKEY_SCAN_TASK_DLY);  //debounce the keys
                xKeyScanTask();               //scan for key press
                if(xKeyHit())
                {
                        LATD = (counter++)%15;

                        keyCode = xKeyGetKey();
                        switch (keyCode)
                                {
                                case xKEY_UP_FLAG:
                                        putrsUSART("Key Up Pressed");
                                        break;
                                case xKEY_DN_FLAG:
                                        putrsUSART("Key Down Pressed");
                                        break;
                                case xKEY_LT_FLAG:
                                        putrsUSART("Key Left Pressed");
                                        break;
                                case xKEY_RT_FLAG:
                                        putrsUSART("Key Right Pressed");
                                        break;
                                case xKEY_CTR_FLAG:
                                        putrsUSART("Key Center Pressed");
                                        break;
                                default:
                                        putrsUSART("Hey, there is something wrong!");
                                }
                        }
                }
}
```

Jotystick_main.c 5-way navigator joystick demonstration program

Program on Joystick_main.c checks for any key press on S1 (5-way navigator joystick). Key debounce achieved by simple software delay function DelayMs(xKEY_SCAN_TASK_DLY)[2].
If key UP or DOWN pressed, the corresponding text message is transmitted via UART for debug purpose. Similarly, text messages debug feature included for keys RIGHT, LEFT, and CENTER, with auto-repeat disabled. Baud rate is 4800bps.

Launch Docklight to see the result as below. Try clicking S1 up, down, right, etc. Now we know our xKey.c driver is at least working.



Docklight screen dump for Joystick_main.c running result

This is not the end, of course. As we have a nice 65k color LCD onboard, what we can do is to design a user interface with icons and manuals for a specific application. It can be a MP3 user interface, photo-viewer for SD card, or hydrograph data-logger user interface. We already have a fully working color LCD module driver, SD card driver with a file system, SHT10 sensor driver, and a joystick driver now. What we need to do next is to integrate these several modules together for a fully working application!

- END -

[2] Timer interrupt should be used for more serious applications