APPLICATION NOTE

ABSTRACT

This application note demonstrates how to connect a keypad matrix to the LPC9xx microcontroller family from Philips Semiconductors.

It explains a software example which implements the keyboard interrupt routine and it gives a schematic for connecting a keyboard to the keypad port of the 89LPC932.

In the end of this application note, a listing of the code is provided.

AN10184

Connecting a keyboard to the Philips LPC9xx microcontroller

Author: Paul Seerden
Cristi Ionescu-Catrina

2002 Sep 14

Philips Semiconductors

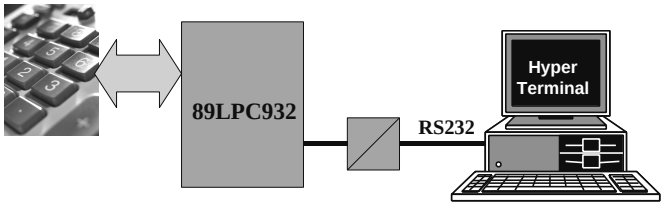
PHILIPS

Connecting a keyboard to the LPC932 microcontroller

AN10184

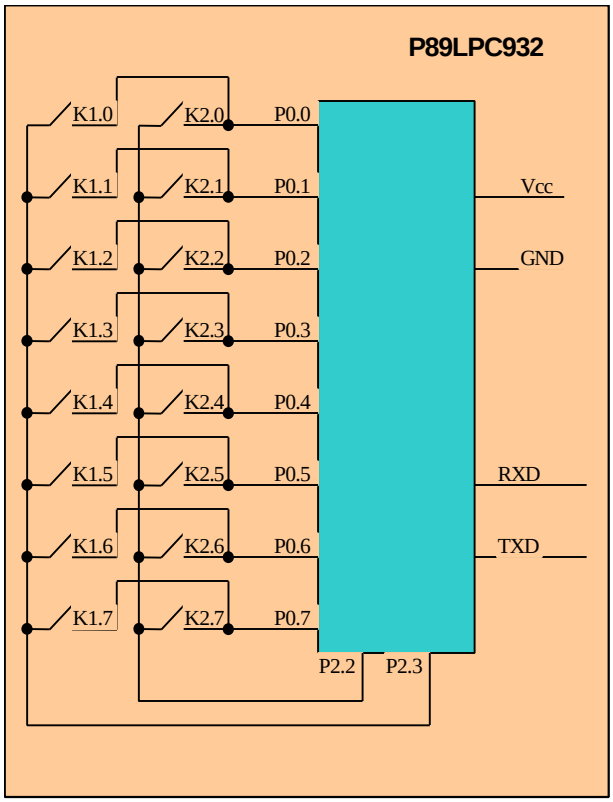
INTRODUCTION

This application note illustrates the use of a Philips P89LPC932 microcontroller to scan a keyboard directly connected to the port pins of the micro.
Received key info is transmitted to a terminal (PC) using an RS232 interface.



HARDWARE

The P89LPC932 has a special keypad interrupt function intended to generate an interrupt when Port 0 is equal or unequal to a configurable pattern. This feature is useful in applications like bus address or keypad recognition.
The user can configure the port through SFR's (special function registers) for different usage. These registers are: the Keypad Interrupt Mask Register (KBMASK), the Keypad Pattern Register (KBPATN) and the Keypad Interrupt Control Register (KBCON).
The first register (KBMASK) is used to define which input pin connected to Port 0 is enabled to generate an interrupt. This means that it is possible to enable individual pins to generate a keyboard interrupt; all other pins can be used for general purposes.
The KBPATN register is used to define a pattern that will be compared to the value of the selected pins of Port 0.
The KBCON register contains two bits: PATN_SEL (KBCON.1) and KBIF (KBCON.0). If the PATN_SEL bit is set, Port 0 has to be equal to the value programmed in the KBPATN register in order to generate the interrupt. If the PATN_SEL bit is clear, Port 0 has to be not equal to the value of the KBPATN in order to generate the interrupt. The KBIF bit is set when Port 0 matches the user-defined conditions in all the 3 registers and generates an interrupt if the Keyboard Interrupt Enable Bit (IEN1.1) is set (and also EA is set). KBIF needs to be cleared in software.



In our example we connected a keyboard with 16 keys in an 8 (column) x 2 (row) matrix to the LPC932. All eight return lines (columns) of the keyboard we connected to port 0. For the two scan (row) lines we used port pins P2.2 and P2.3.

Serial communication

Keyboard info (key n 'pressed' or 'released') is transmitted to the outside world using the on-chip serial port (UART) of the LPC932. In order to run this application one has to connect this port, via a RS232 level shifter, to for instance the COMx port of a PC and run a terminal emulation program like "Hyper Terminal", configured to 19200 Baud, 8 data bits and one stop bit.
After reset or Power up, the micro will send a start-up string to the serial port in order to check that the communication works. If the following message is displayed: "Keyboard Interrupt application", it means that the serial communication is okay. After this, at each key press the message "Key n was pressed" will be sent out and on a key release the message "Key n was released" will be sent.

Oscillator

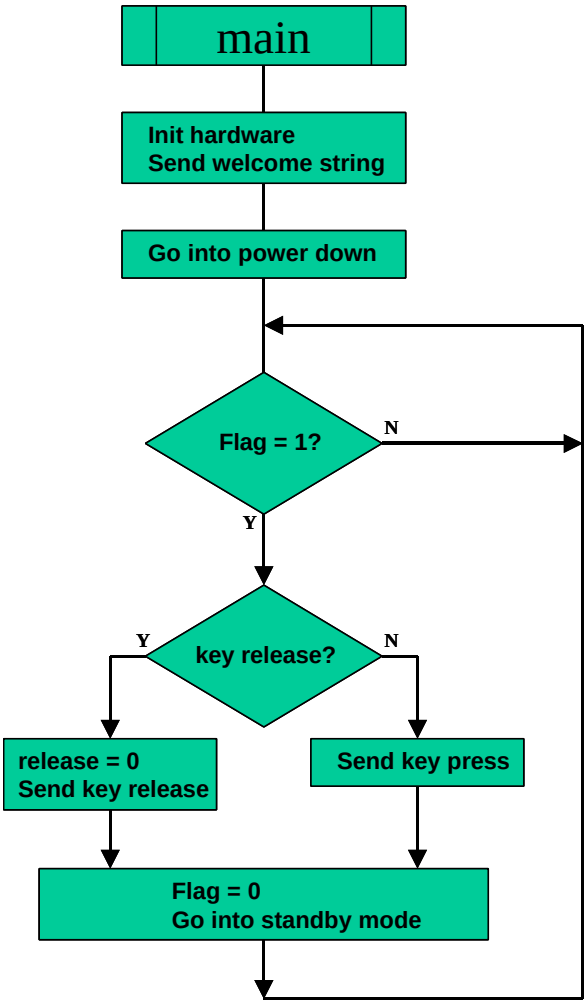
In our example, the micro's internal RC oscillator (7.3728 MHz ±2.5%) is used.

SOFTWARE

Main loop

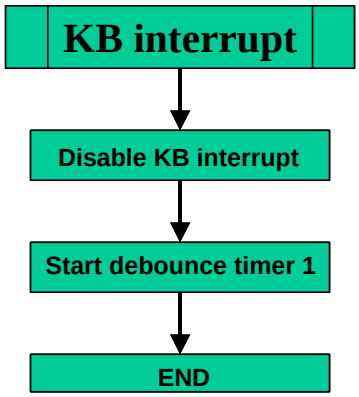
After setting the mode of operation for the port pins and programming the keyboard registers, the UART is initialised. Next, the two scan-lines (port pins P2.2 and P2.3 which select the 2 rows of the keyboard) are pulled low and the LPC932 is forced into power down mode.

If one of the 8 keys in a column is pressed a keyboard interrupt will be generated and the micro will wake up from power down mode. Then the main program is entered that contains an endless loop where the micro just waits for a valid key press or release detection. After that, it sends the key info out using the internal UART and enters power down mode again, waiting to wake up at the next keyboard interrupt.



Keyboard Interrupt

When a key is pressed or released, an interrupt is generated. Inside the Keyboard Interrupt routine the keyboard interrupt enable bit is cleared and the de-bounce Timer 1 is started. This is done in order to let enough time (17.7ms) for the key to stabilize so that we read a correct value from port 0.



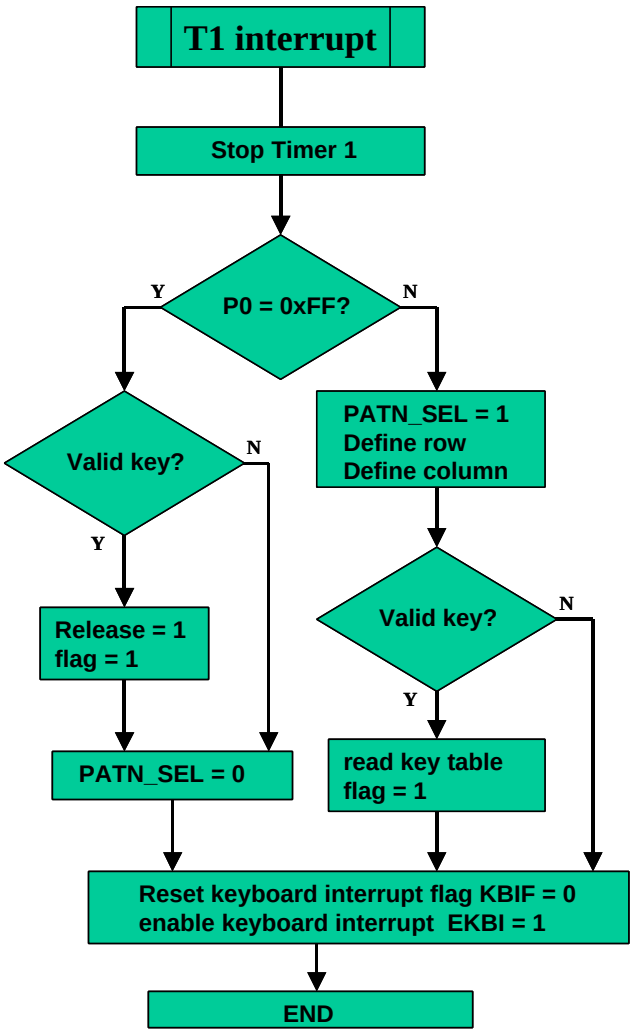
Timer 1 (de-bounce timer) Interrupt

At the very beginning of the Timer 1 interrupt routine, we inspect whether or not Port 0 is equal to 0xFF. This indicates a key press or a key release.

If a key was pressed (P0 unequal to 0xFF) we first set the PATN_SEL bit. This means that the next keyboard interrupt can only be generated if all keys are released (P0 = 0xFF). Next, we check to which row the key belongs by calling the function ***DefineRow***. After that, by reading the value of Port 0, we define to which column the key belongs. Both, the row and column numbers are used to read the key info from a two dimensional array. Finally, we set a flag to inform the main routine we had a valid key press.

If a key was released (P0 equal to 0xFF) we first check if previously we had a valid key press. If that is the case we set both the “Flag” and “release” bits to inform the main routine a valid key release. After that the PATN_SEL bit is cleared. This means that the next keyboard interrupt again will be generated if P0 is unequal to 0xFF (key press).

At the end of the timer interrupt the keyboard interrupt flag is cleared and the keyboard interrupt again is enabled.



KEYBOARD.C LISTING

```
/*****************************************************
* LPC932 Keyboard Interrupt demo program
* Sends the pressed key via UART at 19200kbps.
* Internal RC oscillator is used (7.3728MHz).
*****************************************************/
#include "sfr932.h"

extern void ua_outchar(char c);
extern void UART_Init(void);
extern void PrintString(rom char *s);

rom char key_tbl[][] = {
{'1' ,'A' ,'6' ,'9' ,'7' ,'8' ,'4' ,'5' },    // row 0
{'2' ,'0' ,'E' ,'D' ,'B' ,'C' ,'3' ,'F' }     // row 1
                      };

rom char startup[] = "LPC9xx Keyboard application\n\r";
rom char press[]   = " pressed\n\r";
rom char release[] = " released\n\r";

static unsigned char key,col,row,byte_read;

static bit flag = 0;
static bit rel  = 0;
static bit valid_key = 0;


static void KEYB_Init(void)
{
// P0 (keypad columns) setting
    P0M1 = 0x00;       // P0 is bidirectional
    P0   = 0xFF;

// P2 (keypad rows) setting
    P2M1 &= 0xF3;      // P2_2 and P2_3 are push pull
    P2M2 |= 0x0C;
    P2_2 = 1;
    P2_3 = 1;

    KBMASK = 0xFF;     // to generate an interrupt, P0
    KBPATN = 0xFF;     // should be NOT equal to the
                       // value from KBPATN(0xFF)
    EKBI = 1;          // enable keyboard interrupt

    TL1  = 0;          // Timer1 used to generate a de
                       // bounce delay of 17.7ms
    TH1  = 0;
    TMOD = 0x10;       // Timer 1 mode 1, 16 bit timer
    ET1  = 1;          // enable Timer 1 interrupt
}

static void DefineColumn(void)
{
    valid_key = 1;
    switch (byte_read)
    {
       case 0xFE: col = 0;    break;
       case 0xFD: col = 1;    break;
       case 0xFB: col = 2;    break;
       case 0xF7: col = 3;    break;
       case 0xEF: col = 4;    break;
       case 0xDF: col = 5;    break;
       case 0xBF: col = 6;    break;
       case 0x7F: col = 7;    break;
       default:   valid_key = 0;   break;
    }
}

static void DefineRow(void)
{
    P2_3 = 1;
    if (P0 != 0xFF)
    {
```

```
        P2_3 = 0;
        row = 0;
    }
    else
    {
        P2_3 = 0;
        row = 1;
    }
}

interrupt(3) void Timer1_interrupt(void)
{
    TR1 = 0;                // stop timer
    byte_read = P0;         // save value of port 0
    if (byte_read != 0xFF)  // If P0 |= FF, means
    {                       // that a key was pressed
                            // if not, key release
        KBCON |= 2;         // PATN_SEL = 1, next KB
                            // int on P0 = 0xFF
        DefineRow();
        DefineColumn();
        if (valid_key)
        {
            key = key_tbl[row][col];
            flag = 1;
        }
    }
    else
    {
        if (valid_key)
        {
            rel = 1;
            flag = 1;
        }
        KBCON &= 0xFD;     // PATN_SEL = 0, next inter
                           // rupt if P0 != 0xFF
    }
    KBCON &= 0xFE;         // reset keyb interrupt flag
    EKBI = 1;
}

interrupt(7) void Keyboard_Interrupt(void)
{
    EKBI  = 0;         // disable int.
    TR1 = 1;           // start timer1
}

void main(void)
{
    TRIM = 0x30;
    KEYB_Init();       // initialise keyboard hardware
    UART_Init();       // initialise on-chip UART
    EA = 1;            // enable interrupts

    PrintString(startup);

    P2_2 = 0;          // row 0 low
    P2_3 = 0;          // row 1 low

    PCON |= 2;         // get into power down mode
    #pragma asm
        nop
        nop
    #pragma endasm
    while(1)           // endless loop
    {
        if (flag)
        {
            if (!rel)
            {
                ua_outchar(key);
                PrintString(press);
            }
            else
            {
                rel = 0;
                ua_outchar(key);
                PrintString(release);
            }
            flag  = 0;
            PCON |= 2;       // go into power down mode
            #pragma asm
                nop
                nop
            #pragma endasm
        }
    }
}
```

UART.C LISTING

```
#include "sfr932.h"

rom char ascii[] = "0123456789ABCDEF";


void UART_Init(void)
{
    P1M1 &= 0xFC;     /* TXD (P1.0) -> quasi bidir   */

// BRG -> BRGR1,BRGR0 = (OSC / baudrate) - 16
// BRG -> BRGR1,BRGR0 = (7.373Mhz / 19200) - 16 = 0x170

    BRGR1  = 0x01;
    BRGR0  = 0x70;
    BRGCON = 3;
    AUXR1 |= 0x40;    /* enable break detect reset   */
    SCON = 0x52;      /* mode 1, receiver enable     */
}

void ua_outchar(char c)
{
    while (!TI)  ;
    SBUF = c;
    TI = 0;
}

void PrintString(rom char *s)
{
    while (*s)
    {
        if (*s == '\n')
            ua_outchar('\r');
        ua_outchar(*s);
         s++;
    }
}
```

REFERENCES

For further details please refer to the following publications:

- Datasheets: www.semiconductors.philips.com
- Example Programs: http://www.keil.com/download/c51.asp

Connecting a keyboard to the LPC932 microcontroller

AN10184

Definitions

Short-form specification – The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information, see the relevant datasheet or data handbook.

Limiting values definition – Limiting values given are in accordance with the Absolute Maximum Rating System (IEC134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information – Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support – These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes – Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Contact information

For additional information please visit
http://www.semiconductors.philips.com. Fax: **+31 40 27 24825**

For sales offices addresses send e-mail to:
sales.addresses@www.semiconductors.philips.com

Date of release: 09-02
Document order number: 9397 750 10405

Let's make things better.

Philips Semiconductors

PHILIPS