Keypad decoding technique using TSC80251G1

By François Fosse
Technical Marketing Engineer

1. Description

This application note demonstrates the use of a matrix keypad including wake up from power–down mode with TSC80251G1 microcontroller.

The demonstration software is divided in two parts, a first part that contains the keyboard interrupt drivers, and a second part that demonstrates how to use these drivers. All the software is written in C language.

2. TSC80251G1

In a battery–operated system, such as remote controls, current consumption is an issue. TSC80251G1 implements features to reduce power consumption: the keyboard interrupt feature and the power down mode.

2.1. Keyboard interrupt

TSC80251G1 provides interrupt capabilities on Port 1 especially designed for keyboard. Each input of this port can be independently programmed as high or low level interrupt. Figure 1 shows the keyboard input structure.

This feature has the advantage compared to a periodic polling algorithm that cpu power is only needed when a key is pressed down. This leads to a very low overhead when the keypad is used and no overhead at all when not used.

Moreover keyboard management by periodic polling introduces radio frequency emission (EMI), so interrupt management reduces EMI also.

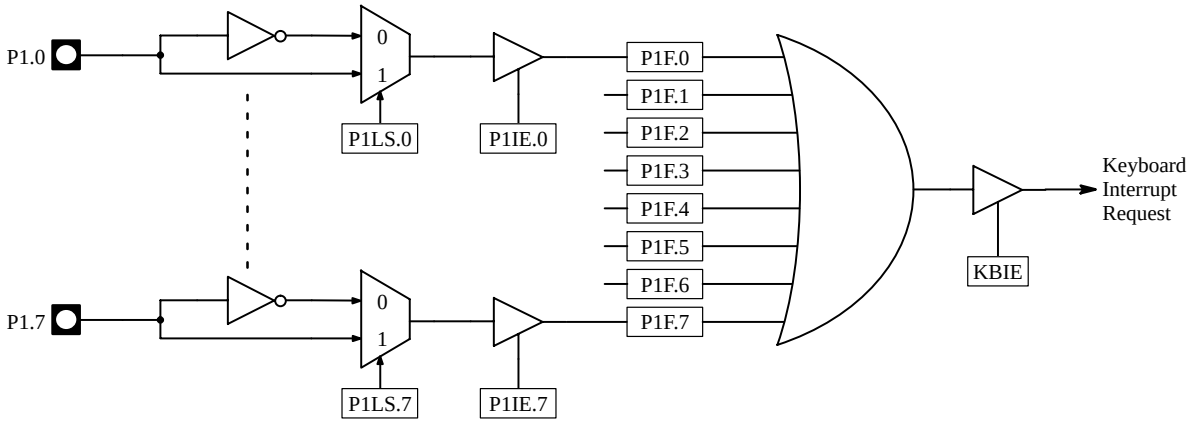


Figure 1. TSC80251G1 keyboard interface structure

2.2. Power down mode

TSC80251G1 provides the capability to exit from power down by using the keyboard input port.

The demonstration software included in this application note shows how to take advantage of this feature to wake up the system by a dedicated key.

2.3. Port Structure

TSC80251 input ports 1, 2 and 3 are internally equipped with pull–up resistors: a medium one when port is floating or connected to a high level and only a weak one when connected to a low level in order to reduce current consumption. These embedded pull–up resistors avoid the need of external ones when connecting a keyboard matrix.

3. Hardware

3.1. Matrix Keypad

3.1.1. Standard arrangement

As soon as number of key is greater than 4, a matrix keypad is more economical in term of I/O used. For example a 16–key pad arranged as a 4 x 4 matrix can be implemented with only eight port pins. To minimize the number of pins required, the keys should be arranged in as a square matrix as shown in Figure 2. Row lines are connected to microcontroller inputs whereas column lines are connected to microcontroller outputs.

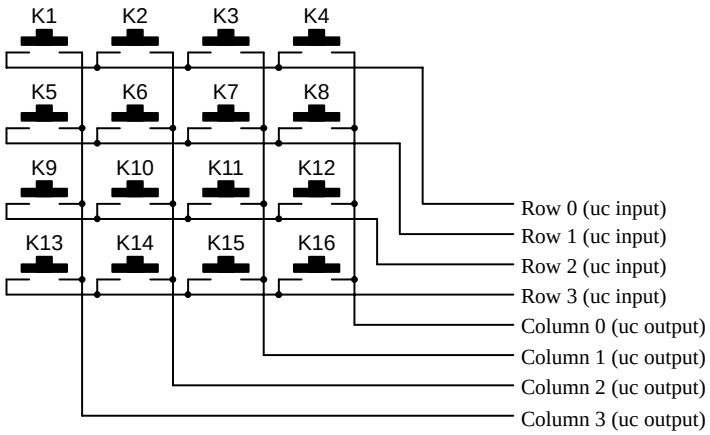


Figure 2. Standard matrix keypad arrangement

3.1.2. Enhanced arrangement

Figure 3 shows the optimum 4 x 4 matrix arrangement that permits to increase the number of key up to 36. This arrangement needs a more complex decoding and is not compatible with keyboard interrupt system. An intermediary arrangement is proposed in Figure 4 and connects permanently a column to GND. This solution has the advantage to save one output port from microcontroller and to remove the current consumption due to internal weak pull–ups during power down mode. The ‘ON’ key used to exit from power down will be selected in this column.

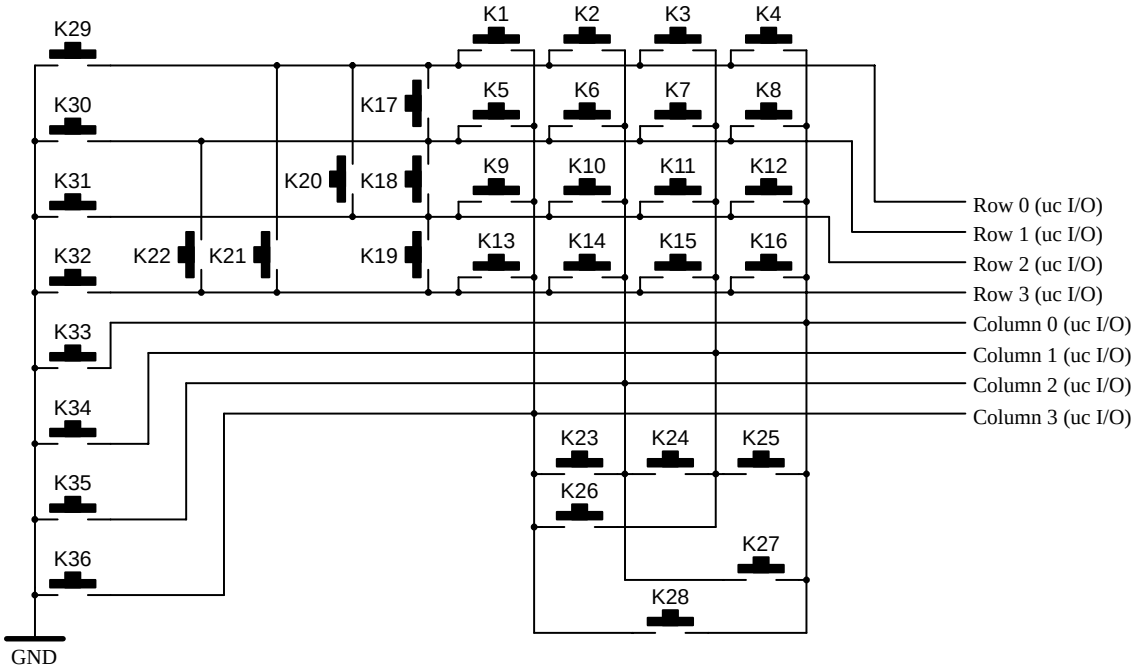


Figure 3. Enhanced matrix keypad arrangement

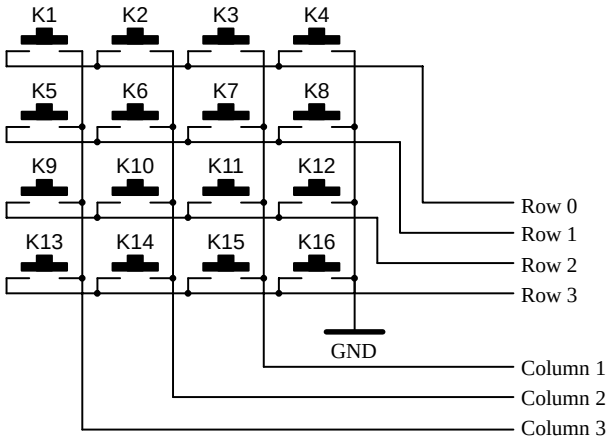


Figure 4. Intermediary matrix keypad arrangement

4. Software

4.1. Development tools

Software has been written using Keil development tools V2.0 and tests have been performed using Keil evaluation board proposed in the TSC80251G1 starter kit. Software can be downloaded from Atmel Wireless & Microcontrollers web site: http://www.atmel–wm.com

4.2. Demonstration software

After initialization, the application is put in power down mode. System wake up is effective after the user strikes the ON key. The main routine waits the user to enter a pin value (4 digits + VAL) and compares this pin value to an expected one. If the value matches, the system stops by entering into power down mode. Keypad connections and ASCII codes are shown in Figure 5. Rows must be connected to port 1 which is the keyboard interrupt port but columns can be taken from any other port. Port 1 is used in the demonstration software but can be easily modified by changing the columns definition in the keypad header file (keypad.h).

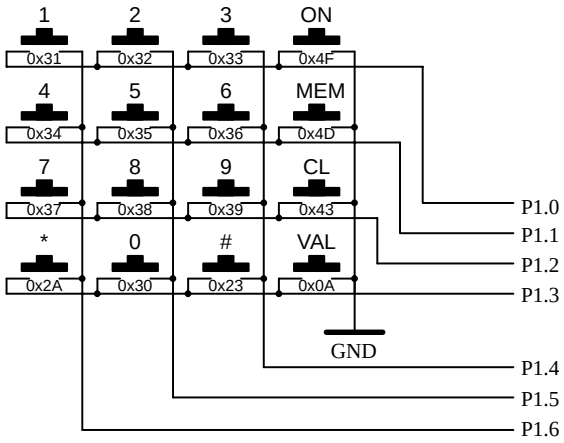


Figure 5. Key assignment

4.3. Algorithm

To detect and decode a key, only 4 interrupt routines are executed: the keyboard interrupt and the debounce timer routines both when the key is pressed down and released.

A sensible issue when writing routines for keypad management is the mechanical key bounce (see Figure 6). The algorithm must then use a debounce timer to avoid multiple key press detection. The debounce delay depends on the keypad technology (silicon rubber, mechanical...) and how the keypad is used.

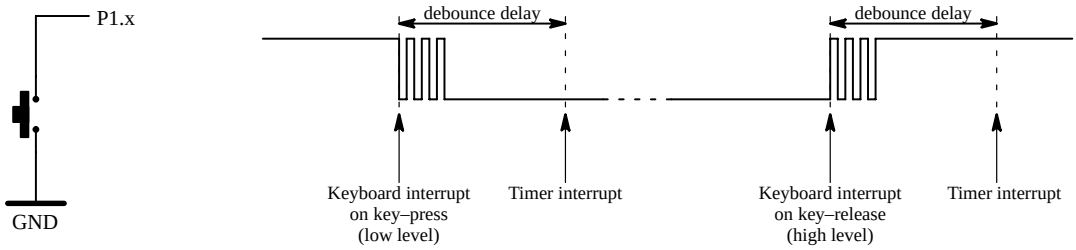


Figure 6. Key behavior

4.3.1. Key press detection

To detect a key, all row inputs are programmed in low level interrupt. Column outputs are set to a low level. When a key is pressed down, a keyboard interrupt is generated. Then interrupt is disabled and debounce timer is launched.

4.3.2. Key release detection

To detect a key release, active row input (the one on which key is pressed) is programmed in high level interrupt. When the key is released, a keyboard interrupt is generated. Then interrupt is disabled and debounce timer is launched.

4.3.3. Key debounce

As soon as a key press has been detected (via keyboard interrupt), a timer is started for the debounce delay. When the timer expires, an interrupt is generated and pressed key decoding is done (see paragraph 4.3.4.). Debounce delay can be easily modified by changing the debounce delay definition in the keypad header file (keypad.h).

In the demonstration software, timer 1 is used to manage the debounce delay. If the application needs to use the UART, TSC80251G1 embeds a baud rate generator that can be used instead of timer 1 to generate the serial clock. Any other timer than timer 1 can be used for the debounce delay: timer 0 or PCA timer.

4.3.4. Key decoding

A pressed down key is decoded by shifting a low level on columns and reading rows. If a logic 0 is read on a row then the key is decoded and converted to an ASCII value to be able to use it with ANSI C functions. Figure 5 shows the ASCII key assignment, the key value can be easily modified by changing the keypad table in the keypad header file (keypad.h).

4.4. Files

The written software implements the `_getkey()` function that waits for a key–press event and returns the ASCII code of this key. This function is the low–level character input routine used by the ANSI C I/O stream functions like scanf, gets... A number of keyboard specific functions has also been implemented like `kbd_install()`...

The following files are proposed in appendix:

- MAIN.C
  This file contains the main demonstration routine.

  `void main (void);` — main routine initializes keyboard interface and interrupts and put the system in power down mode. Wake up is done by pressing the ‘ON’ key and the system waits user entering a pin (4 digits) value followed by the ‘VAL’ key. If the pin value matches a secret one then the system returns in power down mode.

- KEYPAD.C
  This file contains the keyboard driver routines. The following global functions are included:

  `void kbd_install (void);` — keypad installation on keyboard port.

  `void t1_install (void);` — timer 1 installation for debounce delay.

  `void kbd_int (void);` — keyboard interrupt service routine launches the debounce timer.

  `void t1_int (void);` — timer 1 interrupt service routine test if key pressed and decodes it.

  `void sys_stop (void);` — system stop routine prepares wake up by ‘ON’ key and put microcontroller in power down mode.

  `void dec_key (char, char);` — row / column to ASCII key conversion routine.

  `char _getkey (void);` — low–level character input for the stream I/O routines. All stream routines that input character data (getchar, scanf...) do so through this routine.

- PUTCHAR.C
  This file contains the putchar routine. To avoid character output on serial port (standart output port) putchar routine is modified to do nothing. User has to modify this routine depending on its final application. Application note ANM069 gives an example on how to redirect output to a LCD display.
- KEYPAD.H
  This file is the keypad header file and contains constants that can be ajusted by the user depending on his application and keypad type (e.g. port column, ASCII key table, debounce period...).
- SYSTEM.H
  This file is the system header file and contains constants that can be ajusted by user depending on its application (e.g. oscillator frequency, interrupt priority...).
- REG251G1.H
  This file is the register header file and contains SFR and BIT addresses for the TSC80251G1.

5. References

- TSC80251G1 Design Guide, Atmel Wireless & Microcontrollers
- TSC80251 Programmer's Guide, Atmel Wireless & Microcontrollers
- MCB251 Evaluation Board – User's Guide, KEIL Software

6. Sites to visit

- Atmel Wireless & Microcontrollers web pages: http://www.atmel–wm.com

7. Appendix A: Software

7.1. MAIN.C

```
/*
** ---------------------------------------------------------------------------
** MAIN.C
** ---------------------------------------------------------------------------
** Company: Atmel Wireless & Microcontrollers
** Revision: 1.0
** ---------------------------------------------------------------------------
** Comments: This file contains the demonstration software for keypad routines
** ---------------------------------------------------------------------------
*/

/*
** Includes
*/
#include <reg251G1.h>                   /* special function registers 251G1 */
#include <stdio.h>                      /* prototype for I/O functions */

/*
** Defines
*/
#pragma REGISTERBANK (0)                /* use Register Bank 0 for coding */

#define SECRET_PIN    3710

/*
** Prototypes
*/
extern void kbd_install (void);
extern void t1_install (void);
extern void sys_stop (void);

/*
** ---------------------------------------------------------------------------
** main – main user routine
** ---------------------------------------------------------------------------
** Inputs:
**
** Outputs:
**
** ---------------------------------------------------------------------------
** Comments: demonstration routine
** ---------------------------------------------------------------------------
*/

void main (void) {                      /* main entry for program */
int Pin;
  kbd_install();
  t1_install();
  EA = 1;                               /* global interrupt enable */
  sys_stop();                           /* put system in power down */
  while (1)  {                          /* loop forever */
    while (scanf(”%4d”,&Pin) != 1) {    /* read pin value */
      getchar();                        /* flush of getchar buffer
                                           if number of input != 1 */
    }
    if ((getchar() == ’\n’) && (Pin == SECRET_PIN)) {
      sys_stop();                       /* stop system if pin matches */
    }
  }
}
```

7.2. KEYPAD.C

```
/*
** ------------------------------------------------------------------------
** KEYPAD.C
** ------------------------------------------------------------------------
** Company: Atmel Wireless & Microcontrollers
** Revision: 1.0
** ------------------------------------------------------------------------
** Comments:  This file contains the keypad driver and routines
** ------------------------------------------------------------------------
*/

/*
** Includes
*/
#include <reg251G1.h>                   /* special function register reg251G1 */
#include <keypad.h>                     /* define keypad constants */
#include <system.h>                     /* define system constants */

/*
** Defines
*/
#pragma iv(0)                           /* generates interrupt vectors */

/*
** Definitions
*/
bit bdata Key_Hit= 0;                   /* set to 1 if key pressed */
unsigned char data Last_Key = 0;        /* last key pressed value */

char code Key_Table[NB_COL*NB_ROW]=
         {K04, K08, K12, K16,
          K03, K07, K11, K15,
          K02, K06, K10, K14,
          K01, K05, K09, K13};          /* keypad is in code area, keys are
                                           define in keypad.h */
/*
** Prototypes
*/
void kbd_install (void);
void t1_install (void);
void sys_stop (void);
char _getkey (void)
void dec_key (char, char);

/*
** ------------------------------------------------------------------------
** kbd_install – Keyboard install function
** ------------------------------------------------------------------------
** Inputs:
**
** Outputs:
**
** ------------------------------------------------------------------------
** Comments: prepares the keyboard interface
** ------------------------------------------------------------------------
*/
void kbd_install (void) {

  Key_Hit = 0;                          /* no key pressed */

  P_COL |= ~MSK_COL;                    /* all columns active */
  P_ROW |= MSK_ROW;                     /* no active row */

  P1LS = ~MSK_ROW;                      /* low level int on rows */
  P1IE = MSK_ROW;                       /* enable row int */
  P1F = 0;                              /* no interrupt pending */

  IPH1 |= KB_PRIO_H << 0;               /* set priority */
  IPL1 |= KB_PRIO_L << 0;               /* defined in system.h */

  KBIE = 1;                             /* enable Keyboard interrupt */
}
```

```
/*
** ------------------------------------------------------------------------
** t1_install – timer 1 install function
** ------------------------------------------------------------------------
** Inputs:
**
** Outputs:
**
** ------------------------------------------------------------------------
** Comments: prepares the debounce timer
** ------------------------------------------------------------------------
*/
void t1_install (void) {

  TMOD |= 0x10;                         /* timer 1 in delay mode */
  TR1 = 0;                              /* timer 1 stopped */

  IPH0 |= T1_PRIO_H << 3;               /* set priority */
  IPL0 |= T1_PRIO_L << 3;               /* defined in system.h */

  ET1 = 1;                              /* enable timer interrupt */
}


/*
** ------------------------------------------------------------------------
** sys_stop – System stop function
** ------------------------------------------------------------------------
** Inputs:
**
** Outputs:
**
** ------------------------------------------------------------------------
** Comments : prepares wake up by ON key and puts
**             the microcontroller in power down mode
** ------------------------------------------------------------------------
*/
void sys_stop (void) {
  KBIE = 0;                             /* disable interrupt */
  P1IE = MSK_KEY_PD;                    /* enable ON key int */
  P1LS = ~MSK_ROW;                      /* low level int on rows */
  P_COL |= MSK_COL;                     /* only column 0 active*/
  P1F = 0;                              /* no interrupt pending */
  KBIE = 1;                             /* enable interrupt */
  PD = 1;                               /* enter power down mode */
}


/*
** ------------------------------------------------------------------------
** _getkey – wait a key press and return ASCII key value
** ------------------------------------------------------------------------
** Inputs:
**
** Outputs: key pressed value
**
** ------------------------------------------------------------------------
** Comments: _getkey is the low level character input routine for the stream
**           I/O routines. Default routine reads character from the serial
**           interface, new routine hereafter reads character from keypad
** ------------------------------------------------------------------------
*/
char _getkey (void) {
  while (!Key_Hit);
  Key_Hit = 0;
  return Last_Key;                      /* return key pressed */
}
```

```
/*
** ------------------------------------------------------------------------
** t1_int – Timer 1 interrupt function
** ------------------------------------------------------------------------
** Inputs:
**
** Outputs:
**
** ------------------------------------------------------------------------
** Comments: test if a key is pressed and decode it
**           test is done by shifting a 0 on columns and reading rows
**           column 0 is always set to GND and is the first tested
** ------------------------------------------------------------------------
*/
void t1_int (void) interrupt 3 using 1 {
char row;

  TR1 = 0;                              /* stop timer */

  /* here is the key decoding */
  P_COL |= MSK_COL;                     /* only colum 0 activated (always) */
  row = P_ROW;
  if ((row & MSK_ROW) != MSK_ROW) {
    dec_key(row, 0*NB_COL);
  }
  else {
    COL1 = 0;                           /* activate column 1 */
    row = P_ROW;
    if ((row & MSK_ROW) != MSK_ROW) {
      dec_key(row, 1*NB_COL);
    }
    else {
      COL1 = 1;
      COL2 = 0;                         /* activate column 2 */
      row = P_ROW;
      if ((row & MSK_ROW) != MSK_ROW) {
        dec_key(row, 2*NB_COL);
      }
      else {
        COL2 = 1;
        COL3 = 0;                       /* activate column 3 */
        row = P_ROW;
        if ((row & MSK_ROW) != MSK_ROW) {
          dec_key(row,3*NB_COL);
        }
        else {
          /* here there is no key pressed */
          COL3 = 1;
          P_COL &= ~MSK_COL;            /* activate all colums */
          Key_Hit = 0;                  /* no key pressed */
          P1LS = ~MSK_ROW;              /* low level int on all rows */
          P1IE = MSK_ROW;               /* enable all rows interrupt */
        }
      }
    }
  }
  KBIE = 1;                             /* enable Keyboard interrupt */
}
```

```
/*
** ------------------------------------------------------------------------
** kbd_int – Keyboard interrupt function
** ------------------------------------------------------------------------
** Inputs:
**
** Outputs:
**
** ------------------------------------------------------------------------
** Comments: launches the debounce timer
** ------------------------------------------------------------------------
*/
void kbd_int (void) interrupt 8 using 1 {

  KBIE = 0;                             /* disable Keyboard interrupt */
  P1IE = 0;                             /* disable row interrupts */
  P1F = 0;                              /* no interrupt pending */

  TL1 = KB_TEMPO;
  TH1 = KB_TEMPO >> 8;                  /* prepare debounce delay */
  TR1 = 1;                              /* launches timer */
}


/*
** ------------------------------------------------------------------------
** dec_key – Key decoding function
** ------------------------------------------------------------------------
** Inputs:  row: a value from 1, 2, 4, 8
**          col: a value from 0 to 3*NBCOL
**
** Outputs: Last_Key: set to decoded key
**
** ------------------------------------------------------------------------
** Comments: decodes the key pressed with its row and column
**           prepares interface to detect key release: interrupt on high level
** ------------------------------------------------------------------------
*/
void dec_key (char row, char col) {

  row = ~row & MSK_ROW;
  P1LS = row;                           /* high level int on key row */
  P1IE = row;                           /* enable only key row interrupt */
  while ((row >>= 1) != 0) col++;
  Last_Key = Key_Table[col];            /* read key value */
  Key_Hit = 1;
}
```

7.3. PUTCHAR.C

```
/*
** _____________________________________________________________________________
** PUTCHAR.C
** _____________________________________________________________________________
** Company: Atmel Wireless & Microcontrollers
** Revision: 1.0
** _____________________________________________________________________________
** Comments: This file contains the low level character output routine for
**           the stream I/O routines
** _____________________________________________________________________________
*/


/*
** _____________________________________________________________________________
** putchar – send a character to the standard output
** _____________________________________________________________________________
** Inputs: char to output
**
** Outputs: char output
**
** _____________________________________________________________________________
** Comments: putchar is the low level character output routine for the stream
**           I/O routines. Default routine sends character to the serial
**           interface, new routine hereafter does nothing
** _____________________________________________________________________________
*/
char putchar (char c) {
  return (c);
}
```

7.4. KEYPAD.H

```
/*
** ____________________________________________________________________________
** KEYPAD.H
** ____________________________________________________________________________
** Company: Atmel Wireless & Microcontrollers
** Revision: 1.0
** ____________________________________________________________________________
** Comments: This file contains the keypad definitions
** ____________________________________________________________________________
*/

#ifndef _KEYPAD_H_
#define _KEYPAD_H_

/*
** Header inclusion
*/
#include <system.h>                     /* define system constants */

/*
** Defines
*/
/* Keypad arrangement    K01---K02---K03---K04------ Row0
                          :     :     :     :
                         K05---K06---K07---K08------ Row1
                          :     :     :     :
                         K09---K10---K11---K12------ Row2
                          :     :     :     :
                         K13---K14---K15---K16------ Row3
                          :     :     :     :
                          :     :     :      -------- GND (Col0)
                          :     :     --------------- Col1
                          :     --------------------- Col2
                           ------------------------- Col3      */

#define P_ROW           P1              /* row port always P1 */
#define MSK_ROW         0x0F            /* mask for rows: 00001111 */
#define NB_ROW          4               /* number of rows */

#define P_COL           P1              /* columns port */
sbit COL1 = P1 ^ 4;                     /* col 0 set to 0 */
sbit COL2 = P1 ^ 5;
sbit COL3 = P1 ^ 6;

#define MSK_COL         0x70            /* mask for columns: 01110000 */
#define NB_COL          4               /* number of columns */

#define DEBOUNCE_PER    10              /* debounce period (ms) max 65 */
#define KB_TEMPO        (65535-(DEBOUNCE_PER*(FOSC/12)))

#define MSK_KEY_PD      0x01            /* mask for ON key row: 00000001 */

#define K01             0x31
#define K02             0x32
#define K03             0x33
#define K04             0x4F
#define K05             0x34
#define K06             0x35
#define K07             0x36
#define K08             0x4D
#define K09             0x37
#define K10             0x38
#define K11             0x39
#define K12             0x43
#define K13             0x2A
#define K14             0x30
#define K15             0x23
#define K16             0x0A

#endif /* _KEYPAD_H_ */
```

7.5. SYSTEM.H

```
/*
**_____________________________________________________________________________
** SYSTEM.H
**_____________________________________________________________________________
** Company: Atmel Wireless & Microcontrollers
** Revision: 1.0
**_____________________________________________________________________________
** Comments: This file contains the system definitions
**_____________________________________________________________________________
*/

#ifndef _SYSTEM_H_
#define _SYSTEM_H_

/*
** Defines
*/
#define FOSC            12000          /* oscillator frequency (KHz) */

#define KB_PRIO_H       0
#define KB_PRIO_L       0              /* keyboard priority= 0 */
#define T1_PRIO_H       0
#define T1_PRIO_L       0              /* timer 1 priority= 0 */


#endif /* _SYSTEM_H_ */
```

7.6. REG251G1.H

```
/*
** ------------------------------------------------------------------------
** REG251G1.H
**------------------------------------------------------------------------
** Company: Atmel Wireless & Microcontrollers
** Revision: 1.0
** ------------------------------------------------------------------------
** Comments: This file contains the registers definition for
**           TSC8x251G1 microcontroller
** ------------------------------------------------------------------------
*/

#ifndef _REG251G1_H_
#define _REG251G1_H_

/*
** Defines
*/

/* BYTE REGISTERS
   ------------- */
/* ports */
sfr P0     = 0x80;
sfr P1     = 0x90;
sfr P2     = 0xA0;
sfr P3     = 0xB0;

/* system */
sfr ACC    = 0xE0;
sfr B      = 0xF0;

sfr PSW    = 0xD0;
sfr PSW1   = 0xD1;

sfr SP     = 0x81;
sfr SPH    = 0xBE;

sfr DPL    = 0x82;
sfr DPH    = 0x83;
sfr DPXL   = 0x84;

sfr IE0    = 0xA8;
sfr IPL0   = 0xB8;
sfr IPH0   = 0xB7;
sfr IE1    = 0xB1;
sfr IPL1   = 0xB2;
sfr IPH1   = 0xB3;

sfr WCON   = 0xA7;

/* timers */
sfr TCON   = 0x88;
sfr TMOD   = 0x89;
sfr TL0    = 0x8A;
sfr TL1    = 0x8B;
sfr TH0    = 0x8C;
sfr TH1    = 0x8D;

sfr T2CON  = 0xC8;
sfr T2MOD  = 0xC9;
sfr RCAP2L = 0xCA;
sfr RCAP2H = 0xCB;
sfr TL2    = 0xCC;
sfr TH2    = 0xCD;

/* uart */
sfr SCON   = 0x98;
sfr SBUF   = 0x99;
sfr SADDR  = 0xA9;
sfr SADEN  = 0xB9;
sfr BRL    = 0x9A;
sfr BRDCON = 0x9B;
```

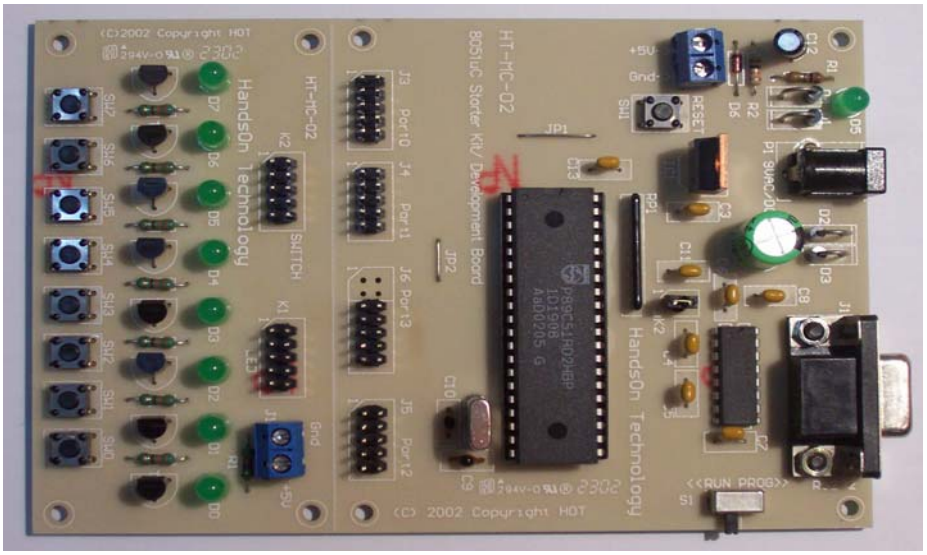
```
/* pca */
sfr CCON   = 0xD8;
sfr CMOD   = 0xD9;
sfr CL     = 0xE9;
sfr CH     = 0xF9;
sfr CCAPM0 = 0xDA;
sfr CCAPM1 = 0xDB;
sfr CCAPM2 = 0xDC;
sfr CCAPM3 = 0xDD;
sfr CCAPM4 = 0xDE;
sfr CCAP0L = 0xEA;
sfr CCAP1L = 0xEB;
sfr CCAP2L = 0xEC;
sfr CCAP3L = 0xED;
sfr CCAP4L = 0xEE;
sfr CCAP0H = 0xFA;
sfr CCAP1H = 0xFB;
sfr CCAP2H = 0xFC;
sfr CCAP3H = 0xFD;
sfr CCAP4H = 0xFE;

/* power management */
sfr PCON   = 0x87;
sfr PFILT  = 0x86;
sfr CKRL   = 0x8E;
sfr POWM   = 0x8F;

/* sslc */
sfr SSCON  = 0x93;
sfr SSCS   = 0x94;
sfr SDAT   = 0x95;
sfr SSADR  = 0x96;
sfr SSBR   = 0x92;

/* keyboard */
sfr P1LS   = 0x9C;
sfr P1IE   = 0x9D;
sfr P1F    = 0x9E;

/* watchdog */
sfr WDTRST = 0xA6;


/* BIT REGISTERS
   ------------- */
/* PSW  */
sbit CY    = 0xD7;
sbit AC    = 0xD6;
sbit F0    = 0xD5;
sbit RS1   = 0xD4;
sbit RS0   = 0xD3;
sbit OV    = 0xD2;
sbit UD    = 0xD1;
sbit P     = 0xD0;

/* TCON */
sbit TF1   = 0x8F;
sbit TR1   = 0x8E;
sbit TF0   = 0x8D;
sbit TR0   = 0x8C;
sbit IE1_  = 0x8B;
sbit IT1   = 0x8A;
sbit IE0_  = 0x89;
sbit IT0   = 0x88;

/* T2CON */
sbit TF2   = 0xCF;
sbit EXF2  = 0xCE;
sbit RCLK  = 0xCD;
sbit TCLK  = 0xCC;
sbit EXEN2 = 0xCB;
```

```
sbit TR2   = 0xCA;
sbit C_T2_ = 0xC9;
sbit CP_RL2= 0xC8;

/* IE0 */
sbit EA    = 0xAF;
sbit EC    = 0xAE;
sbit EADC  = 0xAD;
sbit ES    = 0xAC;
sbit ET1   = 0xAB;
sbit EX1   = 0xAA;
sbit ET0   = 0xA9;
sbit EX0   = 0xA8;

/* IPL0 */
sbit IPLC  = 0xBE;
sbit IPLT2 = 0xBD;
sbit IPLS  = 0xBC;
sbit IPLT1 = 0xBB;
sbit IPLX1 = 0xBA;
sbit IPLT0 = 0xB9;
sbit IPLX0 = 0xB8;

/* P1 */
sbit SDA   = 0x97;
sbit SCL   = 0x96;
sbit CEX4  = 0x97;
sbit CEX3  = 0x96;
sbit CEX2  = 0x95;
sbit CEX1  = 0x94;
sbit CEX0  = 0x93;
sbit ECI   = 0x92;
sbit T2EX  = 0x91;
sbit T2    = 0x90;

/* P3 */
sbit RD    = 0xB7;
sbit WR    = 0xB6;
sbit T1    = 0xB5;
sbit T0    = 0xB4;
sbit INT1  = 0xB3;
sbit INT0  = 0xB2;
sbit TXD   = 0xB1;
sbit RXD   = 0xB0;

/* SCON */
sbit FE    = 0x9F;
sbit SM0   = 0x9F;
sbit SM1   = 0x9E;
sbit SM2   = 0x9D;
sbit REN   = 0x9C;
sbit TB8   = 0x9B;
sbit RB8   = 0x9A;
sbit TI    = 0x99;
sbit RI    = 0x98;

/* CCON */
sbit CF    = 0xDF;
sbit CR    = 0xDE;
sbit CCF4  = 0xDC;
sbit CCF3  = 0xDB;
sbit CCF2  = 0xDA;
sbit CCF1  = 0xD9;
sbit CCF0  = 0xD8;

/*  PCON  */
sbit SMOD1 = PCON ^ 7;
sbit SMOD0 = PCON ^ 6;
sbit RPD   = PCON ^ 5;
sbit POF   = PCON ^ 4;
sbit GF1   = PCON ^ 3;
sbit GF0   = PCON ^ 2;
```

```
sbit PD    = PCON ^ 1;
sbit IDL   = PCON ^ 0;

/* IE1 */
sbit SSIE  = IE1 ^ 5;
sbit KBIE  = IE1 ^ 0;

/* IPH0 */
sbit IPHC  = IPH0 ^ 6;
sbit IPHT2 = IPH0 ^ 5;
sbit IPHS  = IPH0 ^ 4;
sbit IPHT1 = IPH0 ^ 3;
sbit IPHX1 = IPH0 ^ 2;
sbit IPHT0 = IPH0 ^ 1;
sbit IPHX0 = IPH0 ^ 0;

/* IPH1 */
sbit IPHSS = IPH1 ^ 5;
sbit IPHKB = IPH1 ^ 0;

/* IPL1 */
sbit IPLSS = IPL1 ^ 5;
sbit IPLKB = IPL1 ^ 0;

#endif /* _REG251G1_H_ */
```

Low Cost 8051μC Starter Kit/ Development Board *HT-MC-02*

HT-MC-02 *is an ideal platform for small to medium scale embedded systems development and quick 8051 embedded design prototyping.* ***HT-MC-02*** *can be used as stand-alone 8051μC Flash programmer or as a development, prototyping and educational platform*



Main Features:

- 8051 Central Processing Unit.
- On-chip Flash Program Memory with In-System Programming (ISP) and In Application Programming (IAP) capability.
- Boot ROM contains low level Flash programming routines for downloading code via the RS232.
- Flash memory reliably stores program code even after 10,000 erase and program cycles.
- 10-year minimum data retention.
- Programmable security for the code in the Flash. The security feature protects against software piracy and prevents the contents of the Flash from being read.
- 4 level priority interrupt & 7 interrupt sources.
- 32 general purpose I/O pins connected to 10pins header connectors for easy I/O pins access.
- Full-duplex enhanced UART **–** Framing error detection Automatic address recognition.
- Programmable Counter Array (PCA) & Pulse Width Modulation (PWM).
- Three 16-bits timer/event counters.
- AC/DC (9~12V) power supply – easily available from wall socket power adapter.
- On board stabilized +5Vdc for other external interface circuit power supply.
- Included 8x LEDs and pushbuttons test board (free with ***HT-MC-02*** while stock last) for fast simple code testing.
- Industrial popular window ***Keil*** C compiler and assembler included (Eval. version).
- Free ***Flash Magic*** Windows software for easy program code down loading.

PLEASE READ ***HT-MC-02 GETTING STARTED MANUAL*** BEFORE OPERATE THIS BOARD
INSTALL ACROBAT READER (AcrobatReader705 Application) TO OPEN AND PRINT ALL DOCUMENTS