Interfacing a 4x4 Keyboard to an AT91 Microcontroller

Introduction

This Application Note describes programming techniques implemented on the AT91 ARM-based microcontroller for scanning a 4x4 Keyboard matrix usually found in both consumer and industrial applications for numeric data entry.

AT91 Keyboard interface

In this application, a 4x4 matrix keypad requiring eight Input/Output ports for interfacing is used as an example.

Rows are connected to Peripheral Input/Output (PIO) pins configured as output. Columns are connected to PIO pins configured as input with interrupts. In this configuration, four pull-up resistors must be added in order to apply a high level on the corresponding input pins as shown in Figure 1. The corresponding hexadecimal value of the pressed key is sent on four LEDs.

![Keyboard Interface Diagram](image-url)
**AT91 Configuration**

**I/O configuration**

Rows are connected to four PIO pins configured as outputs.

Columns are connected to four PIO pins configured as inputs with interrupts. The idle state of these pins is high level due to four pull-up resistors. PIO interrupt is generated by a low level applied to these pins (caused by a key pressed).

Four additional PIO pins are configured as outputs to send the value of the pressed key to LEDs.

**Timer Counter Configuration**

The Timer Counter is configured in waveform operating mode with RC compare interrupt. The Timer Counter is initialized to be incremented on internal clock cycles. The debouncing time is programmable by initializing the RC compare register value according to the clock source selected. A software trigger is used to reset the timer counter and start the counter clock.

**Interrupt**

When a key is pressed, a low level is applied to the pin corresponding to the column associated to the key (pins configured as inputs with interrupts). A falling edge applied to a column pin creates a PIO interrupt. Then, the processor executes the PIO interrupt subroutine (debouncing) and comes back to its previous state (in the main program). After debouncing time, a RC compare timer interrupt occurs and the processor then executes the timer interrupt subroutine (decoding the pressed key) and comes back to its previous state (in the main program).

**Keyboard Scan**

The Keyboard used is a 4x4 matrixed Keyboard. Columns are connected to pins configured as inputs and having the input change interrupt enabled. The initial state of these pins is high level due to four external pull-up resistors.

The state machine is initialized to start with fast scan which outputs zeroes to all rows and detects all keys at the same time. When a key is pressed, a low level is applied to the corresponding column and causes a PIO interrupt to detect the first edge.

Once any key is detected, debouncing is started. The attempt to press a key on a physical keypad and have this activity detected can fail as a result of several noise sources, glitches, spikes, etc., to mention some of the possible causes of debounce problems. The timer is used to eliminate all noise of less than a few milliseconds. Normally this is dependent on the mechanical characteristics of the keys. In this application example, a 20ms programmable debouncing time is used.

After debouncing is completed, a detailed scan is executed. A second fast scan is done to assure that any detection made during the first fast scan stage was not just noise. (Refer to Figure 2 below.) Then, rows are configured as inputs. When a key is pressed a high level is applied in the corresponding row.

**Figure 2. Keyboard Scan Method**

1. First Edge Detection
2. Fast Scan and Start Debouncing
3. Encode Key Pressed

Debouncing time = 20 ms
Flow Charts

The flow charts shown in Figure 3 and in Figure 4, demonstrate the flow of initialization and interrupt service routine respectively.

Figure 3. Main Program

Figure 4. PIO and Timer Interrupts
Software Modules

This application example is written in C and Assembly language and has been validated on the AT91EB40A Evaluation Board using the AT91 software library V2.0.

Irq_pio.arm

The irq_pio.arm file defines the PIO and Timer 0 assembler interrupt handlers. The macros IRQ_ENTRY and IRQ_EXIT defined in the irq.mac file from the AT91 software library are used to save and restore the context respectively.

Software Delivery

The software is delivered "As Is" without warranty or condition of any kind, either express, implied or statutory. This includes without limitation any warranty or condition with respect to merchantability or fitness for any particular purpose, or against the infringements of intellectual property rights of others.

```
AREA Irg, CODE, READONLY, INTERWORK

INCLUDE ../../periph/aic/irq.mac
INCLUDE ../../periph/pio/pio.inc

IMPORT Keyboard_pioHandlerInt
EXPORT pio_asm_irq_handler
pio_asm_irq_handler

;mana Exception Entry
IRQ_ENTRY
;Call the PIO Interrupt C handler
ldr r0, =Keyboard_pioHandlerInt
mov r14, pc
bx r0
;Manage Exception Exit
IRQ_EXIT

IMPORTTC0_DESC

EXPORT timer0_asm_irq_handler
IMPORT Keyboard_timer0HandlerInt
```
timer0_asm_irq_handler

; - Manage Exception Entry
IRQ_ENTRY
; - Call the timer Interrupt C handler
ldr r1, =Keyboard_timer0HandlerInt
ldr r0, =TC0_DESC
mov r14, pc
bx r1
; - Manage Exception Exit
IRQ_EXIT
END
Keyboard.h

The Keyboard.h file defines the keyboard flags and variables.

```c
#include <stdio.h>

#define KB_COLUMNS 4
#define KB_ROWS 4

#define LEFT_UP 0
#define LEFT_DOWN 1
#define LEFT_LEFT 2
#define LEFT_RIGHT 3
#define DOWN_UP 4
#define DOWN_DOWN 5
#define DOWN_LEFT 6
#define DOWN_RIGHT 7
#define UP_UP 8
#define UP_DOWN 9
#define UP_LEFT 10
#define UP_RIGHT 11
#define STRAIGHT_UP 12
#define STRAIGHT_DOWN 13
#define STRAIGHT_LEFT 14
#define STRAIGHT_RIGHT 15

#define LEFT 0
#define RIGHT 1
#define UP 2
#define DOWN 3
#define STRAIGHT 4

#define LEFT_UP 0
#define LEFT_DOWN 1
#define LEFT_LEFT 2
#define LEFT_RIGHT 3
#define DOWN_UP 4
#define DOWN_DOWN 5
#define DOWN_LEFT 6
#define DOWN_RIGHT 7
#define UP_UP 8
#define UP_DOWN 9
#define UP_LEFT 10
#define UP_RIGHT 11
#define STRAIGHT_UP 12
#define STRAIGHT_DOWN 13
#define STRAIGHT_LEFT 14
#define STRAIGHT_RIGHT 15

#define LEFT_UP 0
#define LEFT_DOWN 1
#define LEFT_LEFT 2
#define LEFT_RIGHT 3
#define DOWN_UP 4
#define DOWN_DOWN 5
#define DOWN_LEFT 6
#define DOWN_RIGHT 7
#define UP_UP 8
#define UP_DOWN 9
#define UP_LEFT 10
#define UP_RIGHT 11
#define STRAIGHT_UP 12
#define STRAIGHT_DOWN 13
#define STRAIGHT_LEFT 14
#define STRAIGHT_RIGHT 15
```

The Keyboard.h file defines the keyboard flags and variables.
Interfacing a 4x4 Keyboard to an AT91 Microcontroller

```c
//* Keyboard Rows definition
#define KEYBOARD_ROW0(1<<1) /* on P1
#define KEYBOARD_ROW1(1<<2) /* on P2
#define KEYBOARD_ROW2(1<<3) /* on P3
#define KEYBOARD_ROW3(1<<4) /* on P4

#define KEYBOARD_ROW_MASK (KEYBOARD_ROW0 | KEYBOARD_ROW1 | KEYBOARD_ROW2 | KEYBOARD_ROW3)

//* Keyboard Columns definition
#define KEYBOARD_COLUMN0(1<<5) /* on P5
#define KEYBOARD_COLUMN1(1<<6) /* on P6
#define KEYBOARD_COLUMN2(1<<7) /* on P7
#define KEYBOARD_COLUMN3(1<<8) /* on P8

#define KEYBOARD_COLUMN_MASK (KEYBOARD_COLUMN0 | KEYBOARD_COLUMN1 | KEYBOARD_COLUMN2 | KEYBOARD_COLUMN3)

//* Keyboard translation
#define COLUMN00
#define COLUMN11
#define COLUMN22
#define COLUMN33
#define ROW00
#define ROW11
#define ROW22
#define ROW33
#define New_Key_Pressed 0x01
```

**Keyboard.c**

The Keyboard.c file is the main file. An interrupt method establishes the processor servicing activities beyond the control of the keypad program. When a key is pressed, an interrupt is called, and the key stroke is processed. After the interrupt, the processor is released to return to its own service routines.

```c
// File Name: keyboard.c
// Object: Keyboard 4x4 matrix

#include "parts/r40008/lib_r40008.h"
#include "parts/r40008/reg_r40008.h"
#include "targets/eb40a/eb40a.h"
#include 'keyboard.h'

extern void pio_asm_irq_handler (void);
extern void timer0_asm_irq_handler (void);

/* Global Variables */
u_char Keyboard_Row;
```

2669A–ATARM–01/03
```c
u_char Key_Pressed;

// * define translation table
const u_char KeyboardTable[NB_ROW][NB_COLUMN] =
{
    {'A','7','4','1'},
    {'0','8','5','2'},
    {'B','9','6','3'},
    {'F','E','D','C'}
};

const int led_mask[NB_ROW][NB_COLUMN] =
{
    {LED1|LED3, LED2|LED3|LED4, LED2, LED4},
    {0, LED1, LED2|LED4, LED3},
    {LED1|LED3|LED4, LED1|LED4, LED2|LED3, LED3|LED4}
};

//*-------------------------------------------------------------------------
//* Function Name: Get_Keyboard_Column
//* Object: Translate the Key buffer column
//* Input Parameters: read- PIO read value
//* Output Parameters: col- Active column value
//*-------------------------------------------------------------------------

u_char Get_Keyboard_Column(u_int read)
{
    u_char col;
    col = 0;
    if ( (~read & KEYBOARD_COLUMN0) == KEYBOARD_COLUMN0)
    {
        col = COLUMN0;
    }
    else if ( (~read & KEYBOARD_COLUMN1) == KEYBOARD_COLUMN1)
    {
        col = COLUMN1;
    }
    else if ( (~read & KEYBOARD_COLUMN2) == KEYBOARD_COLUMN2)
    {
        col = COLUMN2;
    }
    else if ( (~read & KEYBOARD_COLUMN3) == KEYBOARD_COLUMN3)
    {
        col = COLUMN3;
    }
    return col;
}//* End

//*-------------------------------------------------------------------------
//* Function Name: Get_Keyboard_Row
```
Interfacing a 4x4 Keyboard to an AT91 Microcontroller

```c
u_char Get_Keyboard_Row(u_int read)
{
    u_char row;
    row = 0;
    if ( (read & KEYBOARD_ROW0) == KEYBOARD_ROW0 )
    {
        row = ROW0;
    }
    else if ( (read & KEYBOARD_ROW1) == KEYBOARD_ROW1 )
    {
        row = ROW1;
    }
    else if ( (read & KEYBOARD_ROW2) == KEYBOARD_ROW2 )
    {
        row = ROW2;
    }
    else if ( (read & KEYBOARD_ROW3) == KEYBOARD_ROW3 )
    {
        row = ROW3;
    }
    return row;
}
```

```c
void Read_Keyboard(void)
{
    if (~at91_pio_read(&PIO_DESC) & KEYBOARD_COLUMN_MASK) != 0)
    {
        Keyboard_Column = Get_Keyboard_Column(at91_pio_read(&PIO_DESC));
        /* Rows configured as PIO input
        at91_pio_open(&PIO_DESC, KEYBOARD_ROW_MASK, PIO_INPUT);
        */
    }
    if (-at91_pio_read(&PIO_DESC) & KEYBOARD_COLUMN_MASK) != 0)
    {
        /* All PIO Rows are actived
        Keyboard_Column = Get_Keyboard_Column(at91_pio_read(&PIO_DESC));
        */
    }
    /* Columns configured as PIO input
    at91_pio_open(&PIO_DESC, KEYBOARD_COLUMN_MASK, PIO_INPUT);
    */
    /* All PIO columns are actived
    */
```
Keyboard_Row = Get_Keyboard_Row (at91_pio_read (&PIO_DESC));

//* Initialise PIO for next Keyboard scan
at91_pio_open ( &PIO_DESC, KEYBOARD_ROW_MASK, PIO_OUTPUT );
at91_pio_write ( &PIO_DESC, KEYBOARD_ROW_MASK, PIO_CLEAR_OUT );

//* Encode and Display Key pressed
Key_Pressed = KeyboardTable[Keyboard_Row][Keyboard_Column];at91_pio_write
( &PIO_DESC,LED1|LED2|LED3|LED4,LED_OFF);
at91_pio_write ( &PIO_DESC, led_mask[Keyboard_Row][Keyboard_Column],
LED_ON);
}
}//* End

//*-------------------------------------------------------------------------
//* Function Name: Keyboard_timer0HandlerInt
//* Object: C Interrupt Handler called by assembly timer
//* interrupt handler.
//* Input Parameters: none
//* Output Parameters: none
//*-------------------------------------------------------------------------

void Keyboard_timer0HandlerInt (void)
{//* Begin
    u_char dummy;

    //* acknowledge interrupt status
dummy = TC0_SR;

    Read_Keyboard();

    //* Disable RC compare interrupt
    TC0_IDR = TC_CPCS;

}//* End

//*-------------------------------------------------------------------------
//* Function Name: KeyBoard_pioHandlerInt
//* Object: C Interrupt Handler called by assembly PIO interrupt
//* handler.
//* Input Parameters: none
//* Output Parameters: none
//*-------------------------------------------------------------------------

void KeyBoard_pioHandlerInt (void)
{//* Begin

    //* Check if Keyboard PIO interrupt
    u_int tmp;

if ( (~at91_pio_read (&PIO_DESC) & KEYBOARD_COLUMN_MASK) != 0)
{
    /* Trig the timer
    TC0_CCR = TC_SWTRG;
    /* Enable RC compare interrupt
    TC0_IER = TC_CPCS;
    */
    /* enable the next PIO IRQ
    tmp = PIO_ISR;
}

好吧，快告诉我，键盘的初始化函数的代码是如何实现的？
TC0_RC = 40000; /* MCKI=66MHz, TCCLKS= MCKI/32, debouncing time:20ms

    /* LEVEL sensitive interrupt!!
    at91_irq_open(TC0_ID, 5, AIC_SRCTYPE_INT_LEVEL_SENSITIVE,
    timer0_asm_irq_handler);

    /* Enable the clock
    TC0_CCR = TC_CLKEN;

    */ End

    /*--------------------------------------------------------------
    */ Function Name: main
    */ Object: main program
    */ Input Parameters: none
    */ Output Parameters: TRUE
    */--------------------------------------------------------------
    int main( void )
    */ Begin
    {
    //*/LEDs Initialization
    at91_pio_open ( &PIO_DESC, LED1|LED2|LED3|LED4, PIO_OUTPUT ) ;
    at91_pio_write (&PIO_DESC, LED1|LED2|LED3|LED4, LED_OFF ) ;

    Keyboard_Initialization();

    /* Loop forever
    while(1)
    {
    /* Wait for interrupt
    }
    return(TRUE);
    */ End
    }